UNIT IV

4 Software Testing and Maintenance

Syllabus

Testing - Unit testing - Black box testing - White box testing - Integration and System testing - Regression testing - Debugging - Program analysis - Symbolic execution - Model Checking- Case Study.

Contents

4.1 Introduction to Testing

AU : May-07,16,Dec.-04,05,08,10, Marks 8

Definition : Software testing is a critical element of software quality assurance and represents the ultimate review of specification, design and coding.

The purpose of software testing is to ensure whether the software functions appear to be working according to specifications and performance requirements.

4.1.1 Testing Objectives

According to **Glen Myers** the testing objectives are,

1. Testing is a process of executing a program with the intend of finding an error.
2. A good test case is one that has high probability of finding an undiscovered error.
3. A successful test is one that uncovers an as-yet undiscovered error.

The major testing objective is to design tests that systematically uncover types of errors with minimum time and effort.

4.1.2 Testing Principles

Every software engineer must apply following testing principles while performing the software testing.

1. All tests should be traceable to customer requirements.
2. Tests should be planned long before testing begins.
3. The Pareto principle can be applied to software testing - 80 % of all errors uncovered during testing will likely be traceable to 20 % of all program modules.
4. Testing should begin "in the small" and progress toward testing "in the large".
5. Exhaustive testing is not possible.
6. To be most effective, testing should be conducted by an independent third party.

4.1.3 Why Testing is Important ?

- Generally, testing is a process that requires more efforts than any other software engineering activity.
- Testing is a set of activities that can be planned in advance and conducted systematically.
- If it is conducted haphazardly, then only time will be wasted and more even worse errors may get introduced.
- This may lead to have many undetected errors in the system being developed. Hence performing testing by adopting systematic strategies is very much essential in during development of software.

Example 4.1.1 *Why does software testing need extensive planning ? Explain.* AU : May-16, Marks 8

Solution : Following are the issues that explain why the testing need extensive planning.

1. Testing is a process of executing the program with the intent of finding errors. Although it is an expensive activity, yet it is essential to perform testing rigorously especially in systems where human safety is involved.
2. Testing requires the developers to find errors from their software. But it is very difficult for developers to point out errors from their own design or code. Hence many organizations have made distinction between development and testing phase by making different people responsible for each phase.
3. The testing is performed on program's response to every possible input. That means, we should test for all valid and invalid inputs. Although practically testing each and every input can not be tested, the important cases of valid and invalid inputs must be tested.
4. Another aspect of testing is for execution of all possible paths of program. A program path can be traced through the code from the start of the program to program termination. Although it is just difficult to test each and every path of the program, it is expected to test those areas where probability of getting a fault is maximum.

Thus organizations should develop strategies and policies for choosing effective testing techniques which requires extensive planning.

Example 4.1.2 *Distinguish among Error/Fault/Failure.* AU : CSE, Dec.-04, Marks 6

OR

Distinguish between software fault and failure. AU : CSE, Dec.-08, Marks 2

Solution : Error : It is a state that can lead to a system behaviour that is unexpected by the system user.

Fault : It is a characteristic of a software system that can lead to system error.

Failure : It is an event that occurs at some point in time when the system does not deliver a service as per user's expectations.

Example 4.1.3 *Distinguish between defects and errors.* AU : Dec.-05, Marks 6

Solution : Error is a state that can lead to a system behaviour that is unexpected by the system user. The software team performs the formal technical reviews to test the software developed. In this review errors are identified and corrected.

Any errors that remain uncovered and are found in later tasks are called defects. Error removal is software development activity and defect removal is a software quality assurance activity.

Example 4.1.4 *When do you stop testing ? Justify your answer with two illustrations.* **AU : CSE, May-07, Marks 8**

Solution : Testing is a complex activity in the software systems. It is said that testing is an endless process and complete testing not possible for almost all the projects. But there are some common factors that are required to decode when to stop testing.

1. When the testing cost is increasing and if it is more than the project cost then it is enough to test.
2. If the project deadline and testing deadline is already crossed.
3. After completion of critical or key test cases one can stop testing.
4. If the project is meeting functional coverage, code coverage or satisfying the client requirements at some point.
5. When high priority bugs are resolved and defect rates fall below certain specified level.
6. When project progresses through alpha and beta testing.

For example :

Case 1 :

In the development of Windows operating system the Internet Explorer is a famous web browser that we use. There are many security patches that need to be applied. Some patches are already introduced with newer versions of operating systems. But still this product specially shows how testing is an **endless process**.

Case 2 :

For sorting a list of elements typical test cases can be -

Test data	Test case name	Expected result
2, 1, 3	Testing unsorted list.	1, 2, 3
2, 3, 2	Testing when equal elements are present.	2, 2, 3
	Testing empty list.	Print message "List is empty".
–4, –5, 0	Testing with negative numbers.	–5, –4, 0

This much testing is sufficient because it satisfies the major requirement of the function. Thus unlike case 1 we can **stop testing** here.

Example 4.1.5 *What are the characteristics of good tester ?* **AU : Dec.-10, Marks 2**

Solution : Following are the characteristics of good tester -

1. The tester must be able to understand the software. He should be in a position to find out high probability errors.

2. The tester must not conduct two different tests for the same purpose.
3. The tester must be able to write simple test cases.
4. The tester should conduct the tests which should have highest likelihood of uncovering errors.

4.2 Internal and External Views of Testing

There are two views of the testing. The internal view and external view. The internal view is also known as white box testing and the external view is also known as black box testing.

1. Black box testing

The black box testing is used to demonstrate that the **software functions** are operational. As the name suggests in black box testing it is tested whether the **input** is accepted properly and **output** is correctly produced.

The major focus of black box testing is on **functions, operations, external interfaces, external data** and **information.**

2. White box testing

In white box testing the **procedural details** are closely examined. In this testing the internals of software are tested to make sure that they operate according to specifications and designs. Thus major focus of white box testing is on **internal structures, logic paths, control flows, data flows, internal data structures, conditions, loops,** etc.

4.3 Black Box Testing

AU : May-07,12,15,16,17,19,22, Dec.-03,04,11,13,15,16,20, Marks 16

- The black box testing is also called as **behavioural testing.**
- Black box testing methods focus on the functional requirements of the software. Test sets are derived that fully exercise all functional requirements.
- The black box testing is not an alternative to white box testing and it uncovers different class of errors than white box testing.

Why to perform black box testing ?

Black box testing uncovers following types of errors.

1. Incorrect or missing functions
2. Interface errors
3. Errors in data structures
4. Performance errors
5. Initialization or termination errors

4.3.1 Equivalence Partitioning

- It is a black box technique that divides the input domain into classes of data. From this data test cases can be derived.
- An ideal test case uncovers a class of errors that might require many arbitrary test cases to be executed before a general error is observed.
- In equivalence partitioning the equivalence classes are evaluated for given input condition. Equivalence class represents a set of valid or invalid states for input conditions.
- Equivalence class guidelines can be as given below :
  - If input condition specifies a range, one valid and two invalid equivalence classes are defined.
  - If an input condition requires a specific value, one valid and two invalid equivalence classes are defined.
  - If an input condition specifies a member of a set, one valid and one invalid equivalence class is defined.
  - If an input condition is Boolean, one valid and one invalid equivalence class is defined.

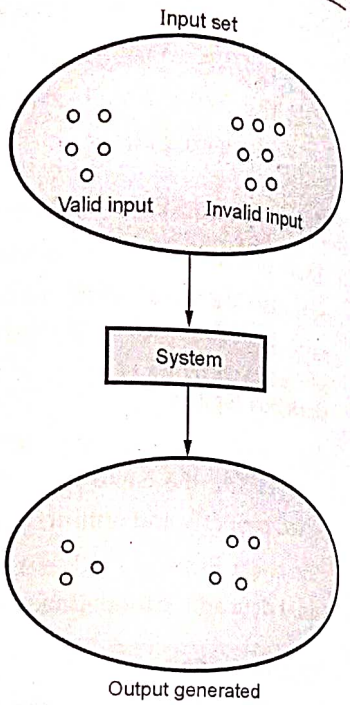


Fig. 4.3.1

For example :

Area code : Input condition, Boolean - The area code may or may not be present.

Input condition, range - Value defined between 200 and 700.

Password : Input condition, Boolean - A password may or may not be present.

Input condition, value - Seven character string.

Command : Input condition, set - Containing commands noted before.

Example 4.3.1 *Describe black box testing. Design the black-box test suit for the following program. The program computes the intersection point of two straight lines and displays the result. It reads two integer pairs (m_1, c_1) and (m_2, c_2) defining the two straight lines of the form $y' = mx + c$.*

AU : May-17, Marks 5

Solution : Black box testing : Refer section 4.3.

The equivalence classes for given program are as follows :

1) Parallel lines if $m_1 = m_2$, $c_1 \neq c_2$

2) Intersecting lines if $m_1 \neq m_2$

3) Coincident lines if $m1 = m_2$, $c_1 = c_2$

The representative values can be selected from each equivalence class.

Thus the test suit obtained can be

(5, 5) (5, 7)

(7, 10) (8, 12)

(15, 15) (15, 15)

4.3.2 Boundary Value Analysis (BVA)

- Boundary value analysis is done to check boundary conditions.
- A boundary value analysis is a testing technique in which the elements at the edge of the domain are selected and tested.
- Using boundary value analysis, instead of focusing on input conditions only, the test cases from output domain are also derived.
- Boundary value analysis is a test case design technique that complements equivalence partitioning technique.
- Guidelines for boundary value analysis technique are,

1. If the input condition specified the range bounded by values x and y, then test cases should be designed with values x and y. Also test cases should be with the values above and below x and y.
2. If input condition specifies the number of values then the test cases should be designed with minimum and maximum values as well as with the values that are just above and below the maximum and minimum should be tested.
3. If the output condition specified the range bounded by values x and y, then test cases should be designed with values x and y. Also test cases should be with the values above and below x and y.
4. If output condition specifies the number of values then the test cases should be designed with minimum and maximum values as well as with the values that are just above and below the maximum and minimum should be tested.
5. If the internal program data structures specify such boundaries then the test cases must be designed such that the values at the boundaries of data structure can be tested.

For example :

Integer D with input condition [– 2, 10],

Test values : – 2, 10, 11, – 1, 0

If input condition specifies a number values, test cases should developed to exercise the minimum and maximum numbers. Values just above and below this min and max should be tested.

Enumerate data E with input condition : {2, 7, 100, 102}

Test values : 2, 102, – 1, 200, 7

Example 4.3.2 *Find the boundary value test cases for the following :*
If x is less than level 1 go to 100 else 200.
If y is greater than level 2 go to 300 else 400. **AU : May-07, Marks 8**

Solution :

Sr. No.	Test case name	Test data	Expected result
1.	Testing lower boundary of x	If x = level 0 If x = level 1	go to 100 go to 200
2.	Testing upper boundary of x	If x = level 2 or more	go to 200
3.	Testing lower boundary of y	If y = level 1 or level 0 If y = level 2	go to 400 go to 400
4.	Testing upper boundary of y	If y = level 3 or more	go to 300

Example 4.3.3 *Design a black box testing for an under water submarine.*

AU : Dec.-11, Marks 8

Solution : Inside submarine there are containers called **ballast tanks.** If ballast tanks are full of air then, the submarine will float otherwise, if water is pumped into the ballast tank then submarine will sink.

To make submarine work underwater it is necessary to fill up ballast tank with water. The submarine will float, sink or rise by adjusting water and air level in ballast tank. The **rudder** of submarine is turned left or right.

Sr. No.	Test case	Expected result
1.	The ballast tank is filled with air above threshold level.	It will float on the surface of water.
2.	The ballast tank is filled with water with water above threshold.	It will sink under water by letting some water out.
3.	Fin is centered.	Submarine can be moved forward.

4.	Fin to right side.	Direction changes.
5.	Fin to left side.	Direction gets changed.
6.	Creating adequate amount of oxygen.	The water is separated out as and O by releasing oxygen.

Example 4.3.4 *A program specs state the following for an input field : The program shall accept an input value of 4-digit integer equal or greater than 2000 and less than or equal 8000. Determine the test cases using.*
i) Equivalence class partitioning. ii) Boundary value analysis. **AU : Dec.-15, Marks 16**

Solution : i) Equivalence class partitioning

We can have set of test cases which are based on input domain

I_1 = { Integer < 2000 }

I_2 = { Integer = 2000 }

I_3 = { Integer > 2000 and < 8000 }

I_4 = { Integer = 8000 }

I_5 = { Integer > 8000 }

The test cases are

Test case	Integer	Expected output
1	1900	Invalid input
2	2000	Valid input
3	5000	Valid input
4	8000	Valid input
5	9000	Invalid input

ii) Boundary Value Analysis :

The boundary values can be,

i) Less than 2000

ii) Greater than and / or equal to 2000

iii) Less than and / or equal to 8000

iv) Greater than 8000.

Test case	Integer	Expected output
1	1000	Invalid input
2	2000	Valid input
3	4000	Valid input
4	8000	Valid input
5	9000	Invalid input

Example 4.3.5 *Consider a program for determining the previous date. Its input is a triple of day, month and year with the values in the range 1 month 12, 1 day 31, 1990 year 2014. The possible outputs would be previous date or invalid input date. Design the boundary value test cases.* **AU : May-16, Marks 8**

Solution : This program takes current date as input and returns previous date as its output.

There are three variables for the program – month, day and year. Hence there would be $4n + 1 = (4 \times 3) + 1 = 13$ test cases that can be designed. The boundary value test cases are.

Test Case	Month	Day	Year	Expected output
1	5	16	1990	15 May 1990
2	5	16	1992	15 May 1992
3	10	1	1995	30 September 1995
4	4	15	1985	Invalid date
5	7	2	2001	1 July 2001
6	3	11	2003	10 March 2003
7	3	29	2005	28 March 2005
8	11	12	2007	11 November 2007
9	11	15	2008	14 November 2008
10	1	15	2008	14 January 2008
11	6	17	2010	16 June 2010
12	8	16	2014	15 August 2014
13	8	16	2015	Invalid date

Example 4.3.6 *Given the requirements for an Automated Teller Machine (ATM) system (see below), design the following :*

i) Use case diagram. ii) Activity diagram detailing each use case.

iii) List test cases for any one functionality from your use case diagram.

The ATM will service one customer at a time. A customer will be required to insert an ATM card and enter a Personal Identification Number (PIN) - both of which will be sent to the bank for validation as part of each transaction. The customer will then be able to perform one or more transactions..

The ATM must be able to provide the following services to the customer : A.customer must be able to make a cash withdrawal from any suitable account linked to the card, in multiples of $20.00. Approval must be obtained from the bank before cash is dispensed.

A customer must be able to make a deposit to any account linked to the card, consisting of cash and/or checks in an envelope. The customer will enter the amount of the deposit into the ATM, subject to manual verification when the envelope is removed from the machine by an operator. Approval must be obtained from the bank before physically accepting the envelope.

A customer must be able to make a transfer of money between any two accounts linked to the card.

A customer must be able to make a balance inquiry of any account linked to the card.

AU : May-19, Marks 15

Solution : i) Use case diagram

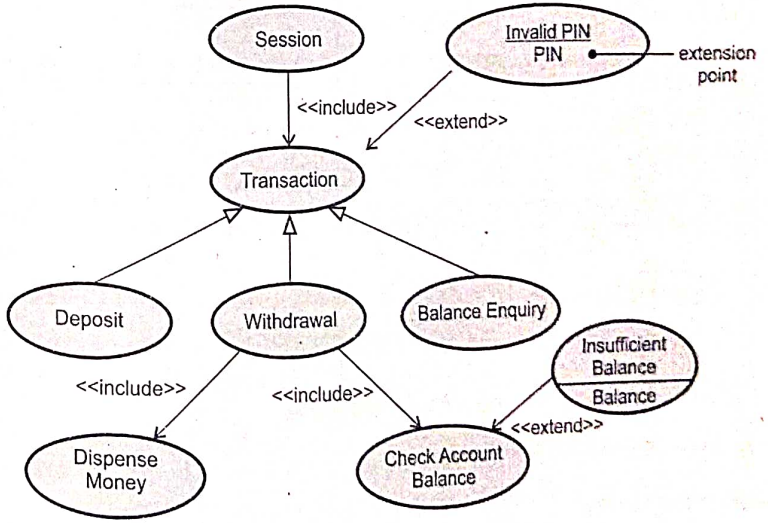


Fig. 4.3.2 Withdrawal of money from ATM

ii) Activity diagram

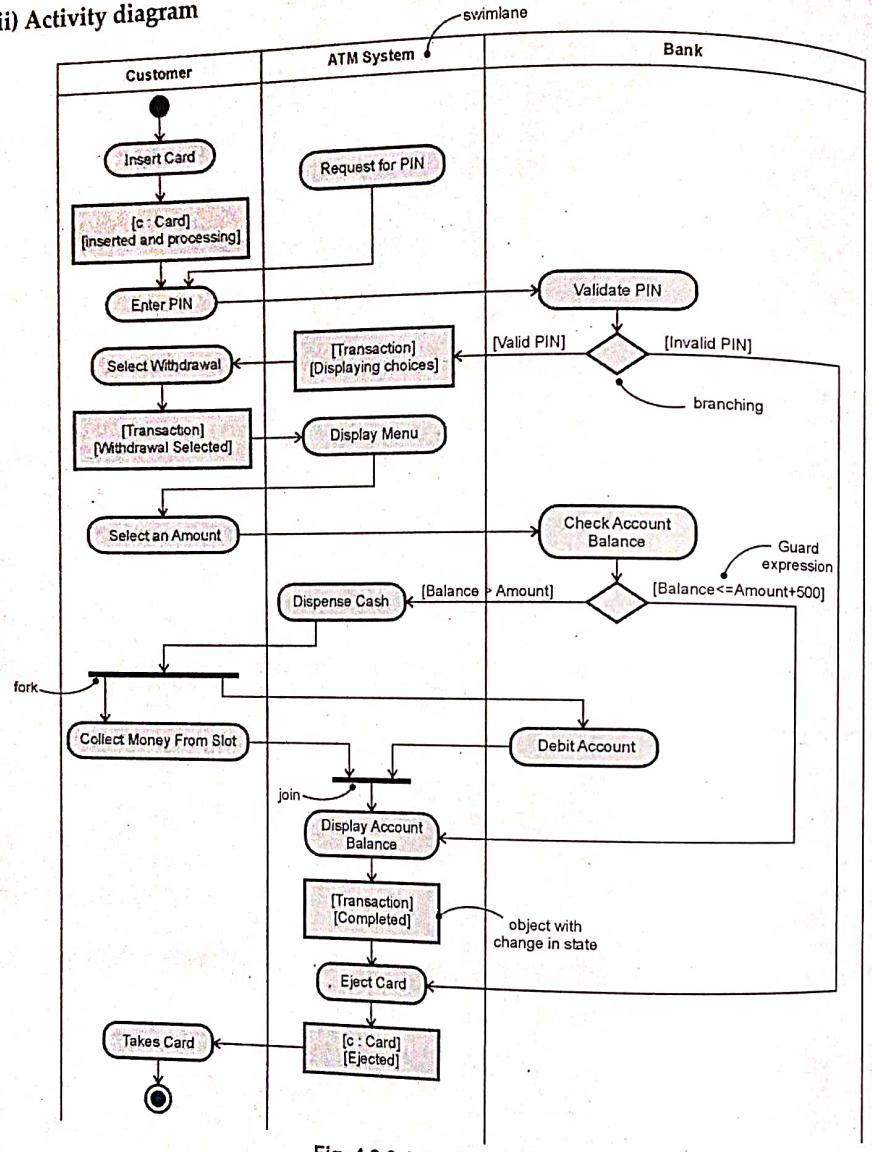


Fig. 4.3.3 Activity model

iii) The list of test cases is as follows -

Cash withdrawal operation

1) Machine does not accept the card as card is expired.
2) User enters wrong PIN.

3) Verify that user is presented with different account type options like - saving, current etc.
4) Verify that user is only allowed to enter amount in multiples of denominations as per the specifications.
5) Verify that user is prompted to enter the amount again in case amount entered is not as per the specification and proper message should be displayed for the same.
6) Verify that user cannot withdraw more amount than the balance.

Money transfer from one account to another

1) Verify that the correct selection of the choice of transfer money from Menu.
2) Verify that the sufficient amount is present in the source account.
3) Verify that the correct account number is mentioned for transfer of amount'.
4) Verify that the correct amount is mentioned for transfer.
5) Verify that other account is linked to the card.

Review Questions

1. *What is black box testing ? Explain the different types of black box testing strategies. Explain by considering suitable examples.* **AU : Dec.-16. Marks 16**
2. *How do you test boundary conditions ?* **AU : May-12, Dec.-03, Marks 2, Dec.-04, Marks 8**
3. *What is boundary value analysis ? Explain the technique specifying rules and its usage with the help of an example.* **AU : Dec.-13. Marks 7**
4. *Describe the various black box and white box testing techniques. Use suitable examples for your explanation.* **AU : May-15, Marks 16**
5. *Explain equivalence partitioning technique with suitable example.* **AU : Dec.-20, Marks 13**
6. *With suitable example, explain boundary value analysis.* **AU : May-22, Marks 6**

4.4 White Box Testing

AU : May-04,07,08,15,17,18,19, Dec.-07,11,13,14,16,17, Marks 10

- The white box testing is also called as structural testing.
- In white box testing derivation of test cases is according to program structure. Hence knowledge of the program is used to identify additional test cases.
- Objective of white box testing is to exercise all program statements.

4.4.1 Condition Testing

- To test the logical conditions in the program module the condition testing is used. This condition can be a Boolean condition or a relational expression.
- The condition is incorrect in following situations.
  i) Boolean operator is incorrect, missing or extra.
  ii) Boolean variable is incorrect.

iii) Boolean parenthesis may be missing, incorrect or extra.

iv) Error in relational operator.

v) Error in arithmetic expression.

- The *condition testing* focuses on each testing condition in the program.
- The *branch testing* is a condition testing strategy in which for a compound condition each and every true or false branches are tested.
- The *domain testing* is a testing strategy in which relational expression can be tested using three or four tests.

4.4.2 Loop Testing

Loop testing is a white box testing technique which is used to test the loop constructs. Basically there are four types of loops.

1. Simple loops :

The tests can be performed for n number of classes.

if,

i) n = 0 that means skip the loop completely.

ii) n = 1 that means one pass through the loop is tested.

iii) n = 2 that means two passes through the loop is tested.

iv) n = m that means testing is done when there are m passes where m < n.

v) Perform the testing when number of passes are n – 1, n, n + 1.

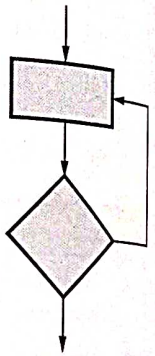


Fig. 4.4.1 Simple loop

2. Nested loops :

The nested loop can be tested as follows.

i) Testing begins from the innermost loop first. At the same time set all the other loops to minimum values.

ii) The simple loop test for innermost loop is done.

iii) Conduct the loop testing for the next loop by keeping the outer loops at minimum values and other nested loops at some specified value.

iv) This testing process is continued until all the loops have been tested.

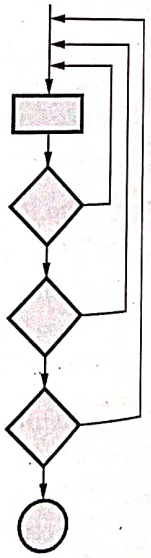


Fig. 4.4.2 Nested loops

3. Concatenated loops :

The concatenated loops can be tested in the same manner as simple loop tests.

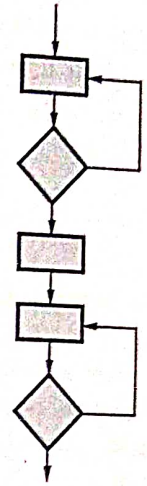


Fig. 4.4.3 Concatenated loops

4. Unstructured loops :

The testing cannot be effectively conducted for unstructured loops. Hence these types of loops needs to be redesigned.

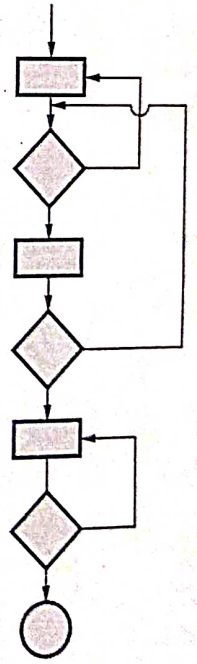


Fig. 4.4.4 Unstructured loops

4.4.3 Basis Path Testing

Path testing is a structural testing strategy. This method is intended to exercise every independent execution path of a program atleast once.

Following are the steps that are carried out while performing path testing.

Step 1 : Design the flow graph for the program or a component.
Step 2 : Calculate the cyclomatic complexity.
Step 3 : Select a basis set of path.
Step 4 : Generate test cases for these paths.

Let us discuss each in detail.

Step 1 : Design the flow graph for the program or a component.

Flow graph is a graphical representation of logical control flow of the program. Such a graph consists of circle called a ***flow graph node*** which basically represents one or more procedural statements and arrow called as ***edges*** or ***links*** which basically represent control flow. In this flow graph the areas bounded by nodes and edges are called *regions*. Various notations used in flow graph are

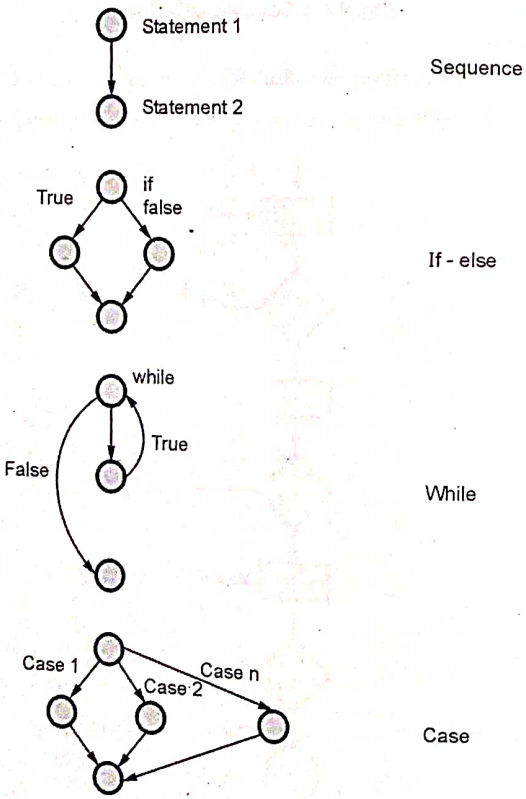


Fig. 4.4.5

For example : Following program is for searching a number using binary search method. Draw a flow graph for the same.

```
void search (int key, int n, int a [ ])
 {
  int mid;
1) int bottom = 0;
2) int top = n – 1;
3) while (bottom <=top)
4) { mid = (top + bottom) / 2;
5) if (a [mid] = = key)
   {
6) printf ("Element is present");
7) return;
  } // end of if
  else
   {
8) if (a [mid] < key)
9) bottom = mid + 1;
  else
10)     top = mid – 1;
  } // end of else
  } // end of while
11)    } // end of search
```

The flow graph will be

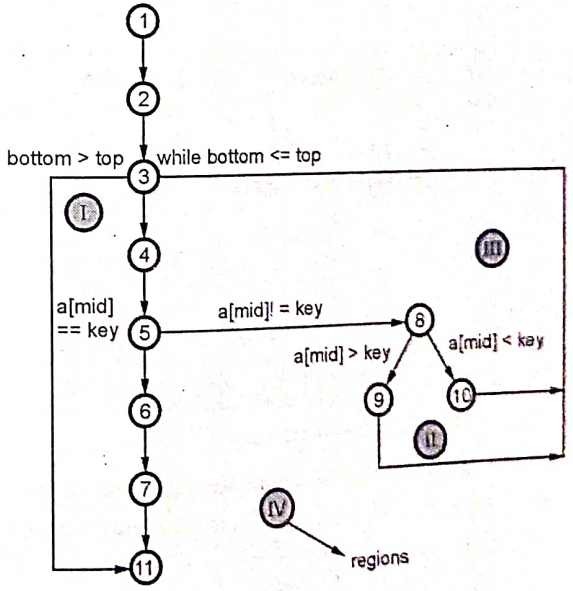


Fig. 4.4.6

Step 2 : Calculate the cyclomatic complexity.

The cyclomatic complexity can be computed by three ways.

1) Cyclomatic complexity = Total number of regions in the flow graph = 4 (note that in above flow graph regions are given by shaded roman letters).

2) Cyclomatic complexity = E – N + 2 = 13 edges – 11 nodes + 2
= 2 + 2 = 4

3) Cyclomatic complexity = P + 1 = 3 + 1 = 4. There are 3 predicate (decision making) nodes : Nodes 3, 5 and 8.

Step 3 : Select a basis set of path.

The basis paths are

Path 1 : 1, 2, 3, 4, 5, 6, 7, 11

Path 2 : 1, 2, 3, 11

Path 3 : 1, 2, 3, 4, 5, 8, 9, 3 ...

Path 4 : 1, 2, 3, 4, 5, 8, 10, 3 ...

Step 4 : Generate test cases for these paths.

After computing cyclomatic complexity and finding independent basis paths, the test cases has to be executed for these paths. The format for test case is -

Preconditions :

Test case id	Test case name	Test case description	Test steps			Test case status (Pass/Fail)	Test priority	Defect severity
			Step	Expected	Actual			
1								
2								
n								

The test case for binary search can be written as -

Precondition : There should be list of elements arranged in ascending order. The element to be searched from the list, its value should be entered and will be stored in variable *'key'*.

Test case id	Test case name	Test case description	Test steps			Test case status	Test status (P / F)	Test priority	Defect severity
			Steps	Expected	Actual				
1.	Validating the list boundary	Checking the bottom and top values for the list of elements	Set bottom = 0 top = n – 1 check if bottom < = top by while loop. This condition defines the length of the list from which the key is searched.	Initially bottom < = top will be true. But during iterations list's length will be reduced and if entire list gets scanned at one point (bottom > = top) will be reached then return to main		Design			
2.	Checking list element with key	Checking if middle element of array is equal to key value	Set mid = (top + bottom) / 2 Then compare if a[mid] is equal to key	If a[mid] = key value then print message "Element is present" and return to main.		Design			
		If a[mid] is < key value if a[mid] is > key value	Set bottom = mid + 1 and then goto "while" set top = mid + 1 and go back to "while" loop	The right sublist will be searched The left sublist will be searched for key element					

Example 4.4.1 *Write a program for sorting of n numbers. Draw the flowchart, flowgraph, find out the cyclomatic complexity.* **AU : Dec.-11, Marks 16**

Solution :

```
#include<stdio.h>
#include<conio.h>
int n;
1.  void main( )
2.  {
3.   int i,A[10];
4.   int j,temp;
5.   printf("\n\t\t Bubble Sort\n");
6.   printf("\n How many elements are there?");
7.   scanf("%d",&n);
8.   printf("\n Enter the elements\n");
9.   for(i=0;i<n;i++)
10.    scanf("%d",&A[i]);
11.  for(i=0;i<=n-2;i++)

12. {
13.    for(j=0;j<=n-2-i;j++)
14.    {
15.    if(A[j]>A[j+1])
16.    {
17.     temp=A[j];
18.     A[j]=A[j+1];
19.     A[j+1]=temp;
20.    }
21.    }
22. }
23. printf("\n The sorted List is ...\n");
24. for(i=0;i<n;i++)
25.    printf(" %d",A[i]);
26. }
```

The flow graph for above program is as shown in Fig. 4.4.7. (See Fig. 4.4.7 on next page)

Example 4.4.2 *Given a set of numbers 'n', the function. Find Prime(a[ ],n) prints a number - if it is a prime number. Draw a control flow graph, calculate the cyclomatic complexity and enumerate all paths. State how many test case-s are needed to adequately cover the code in terms of branches, decisions and statement ? Develop the necessary test cases using sample values for 'a' and 'n'.* **AU : Dec.-13, Marks 16**

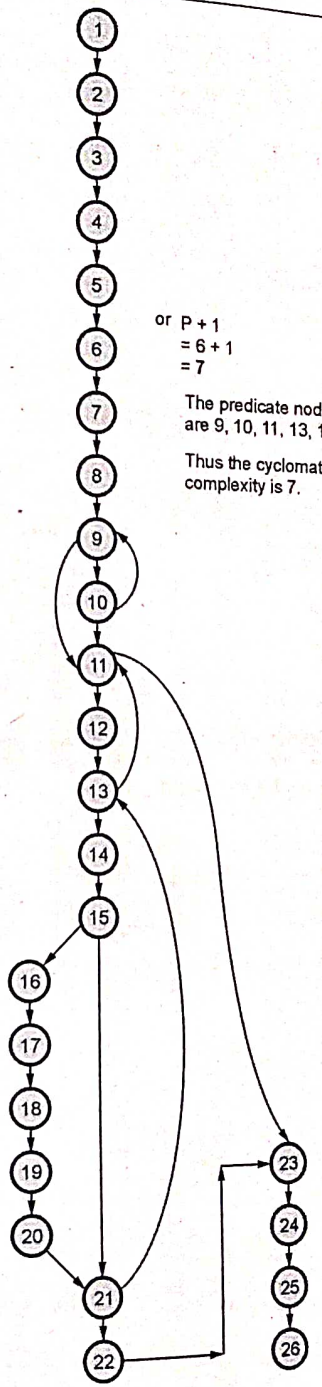


Fig. 4.4.7

Solution :

```
Void Find Prime (int a[], int n)
{
1. i=0
2. while (i < n)
   {
3.   flag = 0
4.   j = 2
5.   while (j < a[i])
     {
6.     rem = a[i]% j;
7.     if (rem = = 0)
       {
8.         flag = 1 ;
9.         break ;
       }
10.        j++;
11.  } //end of while
12.  if (flag = = 0)
13.        Print f ("%d", a[i]);
14.  i++;
15. }//end of while
}//end of function
```

The flow graph will be - (See Fig. 4.4.8 on next page)

Number of edges E = 20

Number of nodes N = 15

Cyclomatic complexity = E – N + 2

= 20 – 15 + 2

= 7

Basis set of path :

Path 1 : 1, 2, 3, 4, 5, ... 15

Path 2 : 1, 2, 15

Path 3 : 1, 2, 3, 4, 5, 6, 7, 10, ... 15

Path 4 : 1, 2, 3, 4, 5, 11, 12, 13, ... 15

Path 5 : 1, 2, 3, ... 12, 14, 15

Path 6 : 1, 2, 3, 4, 5, 11, 12, 14, 15

Path 7 : 1, 2, 3, 4, 5, 11, 12, 13, 14, 15.

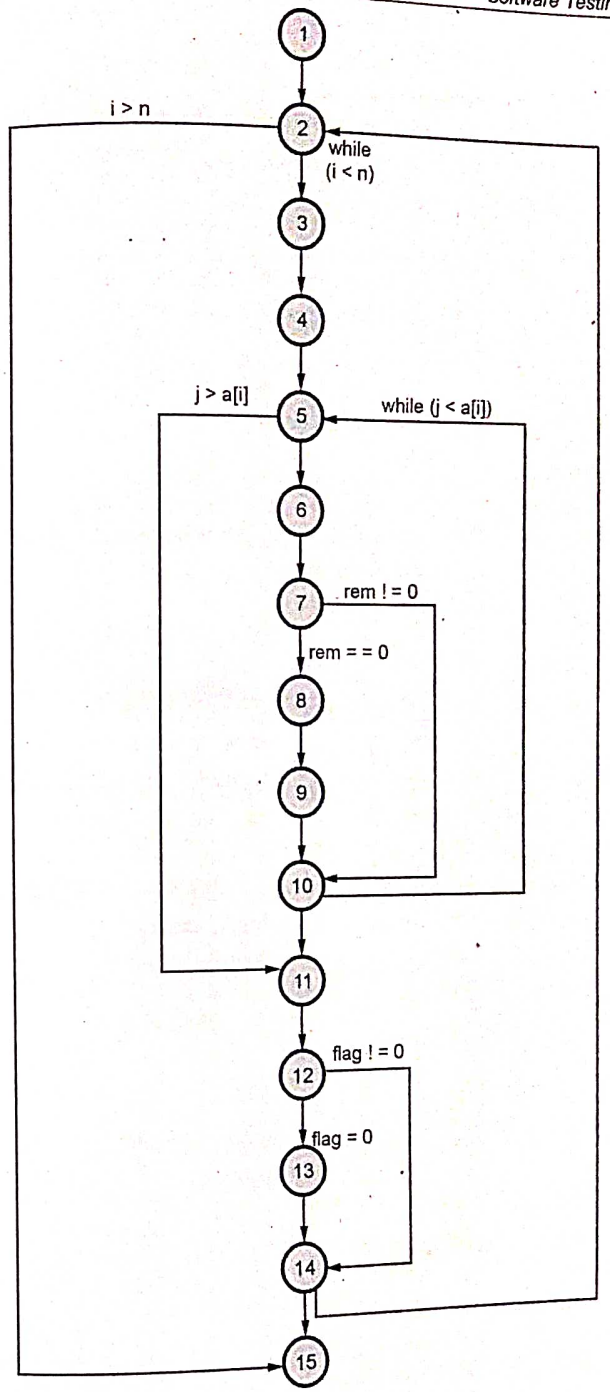


Fig. 4.4.8

Test case :

Pre condition : i) a[ ] stores number to be tested.
ii) j denotes any number within a range to test divisibility.

Test case id	Test case name	Description	Step	Test case status (P/F)

1.	Divisibility by other number	remainder is zero when number in a[ ] is divisible by other than 1 or itself	rem = a[i] % j
			where j < n
			if (rem = = 0)
			set flag = 1
			If (flag ! = 0)
			then
			"Number is not Prime"
				P

2.	Divisibility by 1 or self	remainder is not zero when number in a [ ] is not divisible by a number other than 1 or self	rem = a[i]%j
			where j<n
			if (rem ! = 0)
			set flag = 0
			if (flag = 0)
			then
			"Number is Prime"
				P

Example 4.4.3 *Given a set of 'n' numbers, write an algorithm that finds whether the given number is positive, negative, zero, even or odd. Finally, the total number in each category is also printed. Draw the flow graph and enumerate paths for testing. Determine the number of independent paths using cyclomatic complexity.* **AU : Dec.-14, Marks 8**

Solution : Step 1 : We will first write an algorithm using 'C' code.

```
void main()
{
1. int a[ ] = {10, – 4, 1, 7, 8, 6, 4 };
2 int i;
3. int nz, np, nn, ne, no;
4. nz = np = nn = ne = no = 0;
5. for(i = 0; i < n; i++)
   {
6.   if (a[i] = = 0)
7.   nz++; /* number of zeros */
8.   else if(a[i]>0)
9.   np++; /* number of positive numbers */
10.  else
11.  nn++;* number of negative numbers */
   }
```



```
12.  for(i = 0; i<n; i++)
     {
13.  if(a[i]%2 = = 0)
14.      ne++;
15.  else
16.      no++;
     }
17.  printf ("Number of zeros %d", nz);
18.  printf ("Number of positive %d", np);
19.  printf ("Number of negatives %d", nn);
20.  printf ("Number of even %d", ne);
21.  printf ("Number of odd %d", no);
}
```

Step 2 : We will draw the flow graph from above code. It is as follows -

(See Fig. 4.4.9 on next page.)

Step 3 : Some of the independent paths for testing are -

Path I : 1, 2, 3, 4, 5, 6, 7, 5, 12, 13, 15, 16, 12, 17, 18, 19, 20, 21.

Path II : 1, 2, 3, 4, 5, 6, 8, 9, 5, 12, 13, 15, 16, 12, 17, 18, 19, 20, 21.

Path III : 1, 2, 3, 4, 5, 6, 8, 10, 11, 5, 12, 13, 15, 16, 12, 17, 18, 19, 20, 21.

Path IV : 1, 2, 3, 4, 5, 12, 17, 18, 19, 20, 21.

Step 4 : The cyclomatic complexity can be computed as follows -

Method 1 : Number of edges (E) = 25

Number of vertices (N) = 21

Cyclomatic complexity = E – N + 2

= 25 – 21 + 2

= 6

Method 2 : The total number of regions in Fig. 4.4.9 = 6.

Hence cyclomatic complexity = 6.

Note that the area outside the graph is also treated as one region. Hence VI is marked as a region, along with all other closed loops.

Method 3 : Number of predicate nodes (P) = 5

Because node 5, 6, 8, 12, 13 are predicate nodes

∴ Cyclomatic complexity = P + 1

= 5 + 1

= 6

Thus the cyclomatic complexity is **6.**

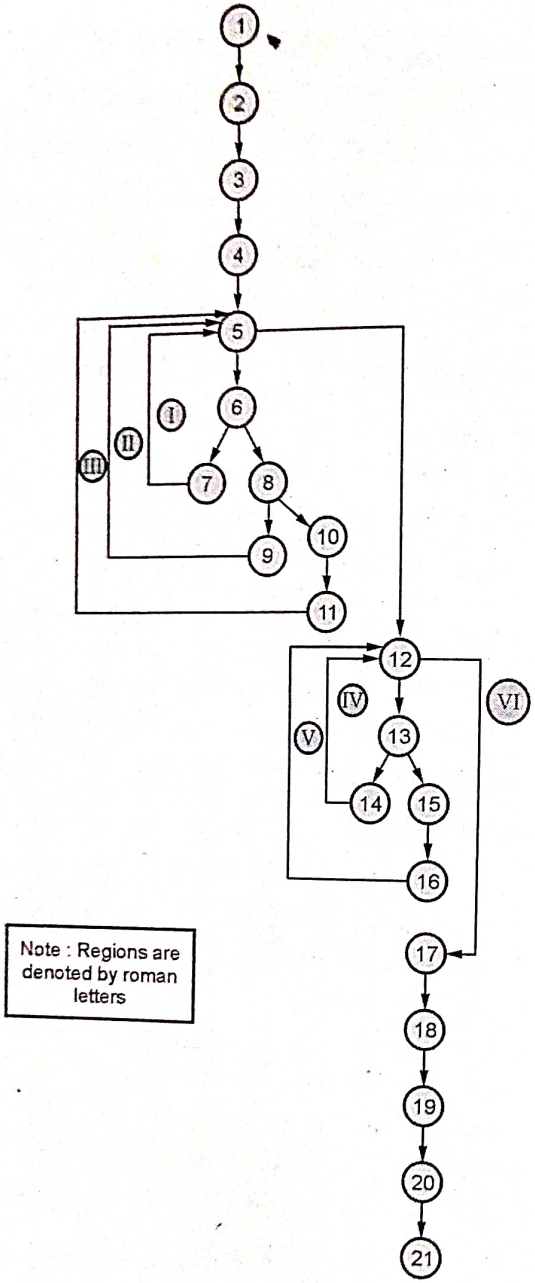


Fig. 4.4.9

Example 4.4.4 *Consider the following program segment.*

```
/* num is the number the function searches in a presorted integer array arr*/
int bin_search (int num)
{
int min, max; min = 0; max = 100;
while (min! = max)
if (arr[min + max]/2) > num
max = (min + max) /2;
else if (arr[(min + max)/2]
min = (min + max)/2;
else return ((min + max)/2);
}
return (- 1);
}
```

i) Draw the control flow graph for this program segment.

ii) Define cyclomatic complexity.

iii) Determine the cyclomatic complexity for this program. Show the intermediate steps in your computation. Writing only the final result is not sufficient.

AU : Dec.-17. Marks (2+2+9)

Solution : i) We will number the program statements as follows

```
int bin_search (int num)
{
1)  int min, max; min = 0; max = 100;
2)  while (min! = max)
3)  if (arr[min + max]/2) > num
4)  max = (min + max) /2;
5)  else if (arr[(min + max)/2]
6)  min = (min + max)/2;
7)  else return ((min + max)/2);
8)  }
9)  return (- 1);
}
```

The Control Flow Graph(CFG) is as follows - (Refer Fig. 4.4.10)

ii) Cyclomatic complexity : The cyclomatic complexity is defined as the number of independent paths in the basis set of programs that provides the upper bound for the number of tests that must be conducted to ensure that all the statements have been executed atleast once.

iii) Cyclomatic complexity = E – N + 2 = 14 – 9 + 2 = 7

Total number of regions = 7

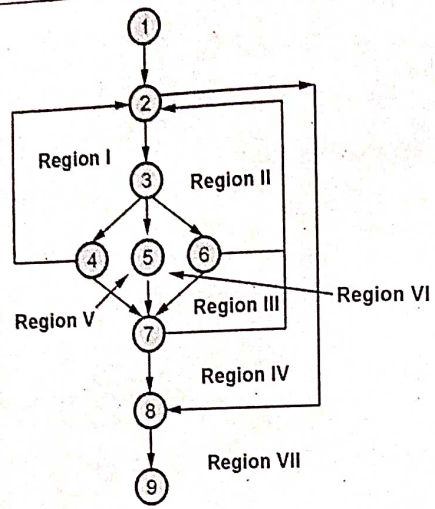


Fig. 4.4.10

Example 4.4.5 *Design test case for the following program :*

GCD(X,Y); if X = Y, then X
else if X>Y, then GCD(X–Y,Y)
else GCD(X,Y–X)
end if
end if

AU : May-07, Marks 8

Solution : The typical format of writing the test is as given below.

Precondition : If any precondition is required then mention it.

Test case id	Test case name	Test case description	Testing steps			Test case status (Pass/ Fail)	Test priority	Test severity
			Step	Expected result	Actual result			

Precondition : The two non-zero numbers should be entered for obtaining the GCD value.

Test case id	Test case name	Test case description	Testing steps			Test case status (Pass/ Fail)	Test priority	Test severity
			Step	Expected result	Actual result			
1.	X = Y	If the value of both the numbers is same.	Enter two numbers such that X = Y.	The GCD is value of X.				
2.	X > Y	If first number is greater than the second one.	Enter two numbers such that value of X is > Y.	Call recursive routine by setting X = X – Y and Y as it is.				
3.	X<Y	If first number is less than the second one.	Enter two numbers such that value of X < Y.	Call recursive routine by setting X as it is and Y = Y – X.				

Example 4.4.6 *What are the attributes of a "good" test ? Explain the test case design.*

AU : CSE. May-08. Marks 10

Solution : Attributes of "good" test :

1. Finding high probability errors should be the goal of testing. For that purpose it is necessary to understand the software. The developer must have a judgement of where the software might fail.
2. Good test is not redundant. That means two different tests must not be conducted for the same purpose.
3. A good test is said to be "best of breed". The test should have highest likelihood of uncovering errors.
4. A good test must not be too simple not too complex.

Test case design : Refer section 4.4.3.

Example 4.4.7 *Consider the pseudo code for simple subtraction given below*

1) Program 'Simple substraction'
2) Input (x, y)
3) Output (x)
4) Output (y)
5) If x > y then D0
6) x – y = z
7) Else y – x = z
8) EndIf
9) Output (z)
10) Output "End Program"

Perform basis path testing and generate test cases.

AU : Dec.-16, Marks 10, May-17, Marks 6; May-18, Marks 9

Solution : Step 1 : We will first draw the flow graph for given pseudo code. This flow graph is as shown in Fig. 4.4.11.

Step 2 : We will compute cyclomatic complexity using following formula = e – V + 2

where e is total number of edges, V is total number of vertices

$\therefore \quad 9 - 9 + 2 = 2$

Now we will find two independent paths for basis path testing.

Step 3 :

Path 1 = 2 – 3 – 4 – 5 – 6 – 8 – 9 – 10

Path 2 = 2 – 3 – 4 – 5 – 7 – 8 – 9 – 10

Step 4 : The test cases for these paths are as given below

Independent path	x	y	Expected Result (z)
Path 1 2 – 3 – 4 – 5 – 6 – 8 – 9 – 10	10	5	5 End Program
Path 2 2 – 3 – 4 – 5 – 7 – 8 – 9 – 10	5	10	5 End Program
Path 2 2 – 3 – 4 – 5 – 7 – 8 – 9 – 10	5	5	0 End Program

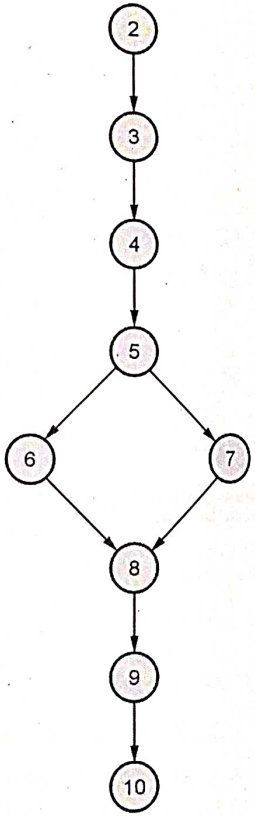


Fig. 4.4.11

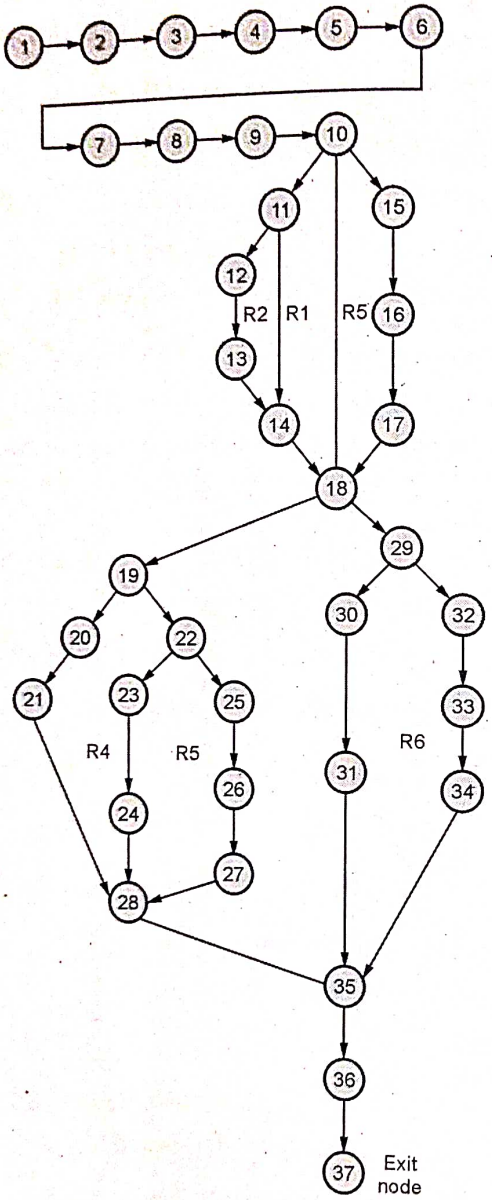
Example 4.4.8 *Write a procedure for the following : Given three sides of a triangle, return the type of triangle i.e. equilateral, isosceles and scalene triangle. Draw the control flow graph and calculate cyclomatic complexity to calculate the minimum number of paths. Enumerate the paths to be tested.*

AU : May-19, Marks 9

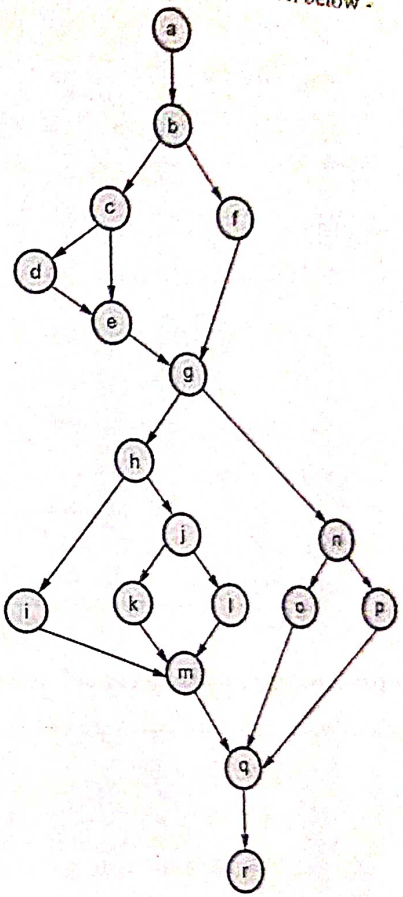
Solution : Step 1 : We will write the procedure for finding the type of triangle. Each important step is numbered so that we can draw the flow graph in the next step

```
(1) int main ( )
(2) {
(3) int a, b, c, status = 0;
(4) printf ("nt Enter side a :");
(5) scanf ("%d", & a);
(6) printf("nt Enter side b :");
(7) scanf ("%d", & b);
(8) printf ("nt Enter side c:");
(9) scanf ("'%d", &c);
(10) if ((a > 0) && (a < - 100) && (b > 0) && (b < . 100) && (c > 0) && (c < =100)) {
(11) if ((a + b) > c) && ((c + a) > b) && ((b + c) > a)) {
(12) status = 1;
(13) }
(14) }
(15) else {
(16) status = -1;
(17) }
(18) if (status = = 1) {
(19) if ((a = =b) && (b = =c)) {
(20) printf ("Triangle is equilateral");
(21) }
(22) else if ((a = =b) I I (b = = c) I I (c = = a)) {
(23) print ("Triangle is isosceles");
(24) }
(25) else {
(26) printf("Triangle is scalene");
(27) }
(28) }
(29) else if (status = = 0) {
(30) printf ("Not a triangle");
(31) }
(32) else
(33) printf ("Invalid input range");
(34) }
(35) getch ( );
(36) return -1;
(37) }
```

Step 2 : Now the flow graph is designed as follows -



Step 3 : The Decision to Decision path(DD Path) is drawn from the above flow graph. In this graph we only concentrate on decision nodes, the sequential nodes are combined together into one node. The DD path graph is as shown below -



Step 3 : Computing the cyclomatic complexity as follows -

Method 1 : Cyclomatic complexity = E – N + 2 where E is number of edges and N are number of nodes

= 23 – 18 + 2

= 7

Method 2 : Cyclomatic complexity = P + 1 where P is a predicate node

= 6 + 1 where b, c, g, h, j and n are predicate nodes

= 7

Method 3 : Cyclomatic complexity = Number of regions +1
= 6 + 1
= 7

Thus the cyclomatic complexity is 7

Step 4 : The independent paths to be tested are -

(1) abfgnpqr

(2) abfgnoqr

(3) abcegnpqr

(4) abcdegnoqr

(5) abfghimqr

(6) abfghjkmqr

(7) abfghjlmqr

Review Questions

1. *What is white box testing ? Explain.* **AU : May-17, Marks 7**
2. *Explain how the various types of loops are tested.* **AU : Dec.-17, Marks 9**

4.5 Testing Strategies

AU : May-05, 06, 22, Marks 16

- We begin by **'testing-in-the-small'** and move toward **'testing-in-the-large'**.
- Various testing strategies for conventional software are
  1. Unit testing 2. Integration testing
  3. Validation testing 4. System testing

1. **Unit testing** - In this type of testing techniques are applied to detect the errors from each software component individually.
2. **Integration testing** - It focuses on issues associated with verification and program construction as components begin interacting with one another.
3. **Validation testing** - It provides assurance that the software validation criteria (established during requirements analysis) meets all functional, behavioural and performance requirements.
4. **System testing** - In system testing all system elements forming the system is tested as a whole.

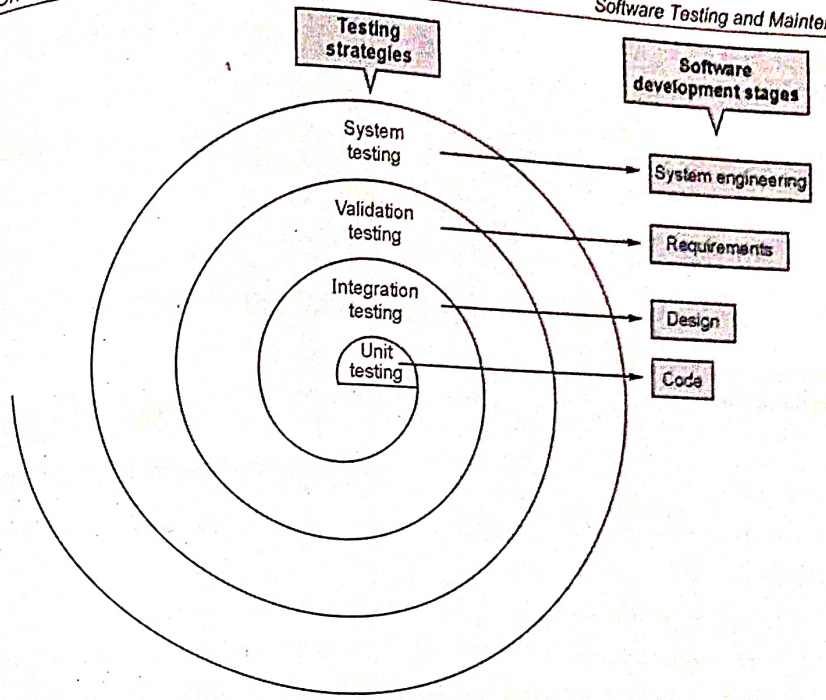


Fig. 4.5.1 Testing strategy

➢ **Difference between black box testing and White box testing**

Sr. No.	Black box testing	White box testing
1.	Black box testing is the software testing method which is used to test the software **without knowing the internal structure** of code or program.	White box testing is the software testing method in which **internal structure** is being **known** to tester who is going to test the software.
2.	This type of testing is **carried out by testers.**	Generally, this type of testing is carried out **by software developers.**
3.	**Implementation knowledge is not required** to carry out black box testing.	**Implementation knowledge is required** to carry out white box testing.
4.	Programming knowledge is not required to carry out black box testing.	Programming knowledge is required to carry out white box testing.
5.	Testing is applicable on **higher levels of testing** like system testing, acceptance testing.	Testing is applicable on **lower level of testing** like unit testing, integration testing.

6.	Black box testing means functional test or **external testing.**	White box testing means structural test or **interior testing.**
7.	Black box testing can be started based on **requirement specifications documents.**	White box testing can be started based on **detail design documents.**
8.	The functional testing, behavior testing, **close box testing** is carried out under black box testing.	The Structural testing, logic testing, path testing, loop testing, code coverage testing, open box testing is carried out under white box testing.

Review Question

1. *Explain the various levels of software testing with suitable examples.* **AU : May-22, Marks 7**

4.6 Unit Testing

AU : Dec.-06,08,15, May-03,09, Marks 16

- In unit testing the individual components are tested independently to ensure their quality.
- The focus is to uncover the errors in design and implementation.
- The various tests that are conducted during the unit test are described as below.
  1. Module interfaces are tested for proper information flow in and out of the program.
  2. Local data are examined to ensure that integrity is maintained.
  3. Boundary conditions are tested to ensure that the module operates properly at boundaries established to limit or restrict processing.
  4. All the basis (independent) paths are tested for ensuring that all statements in the module have been executed only once.
  5. All error handling paths should be tested.

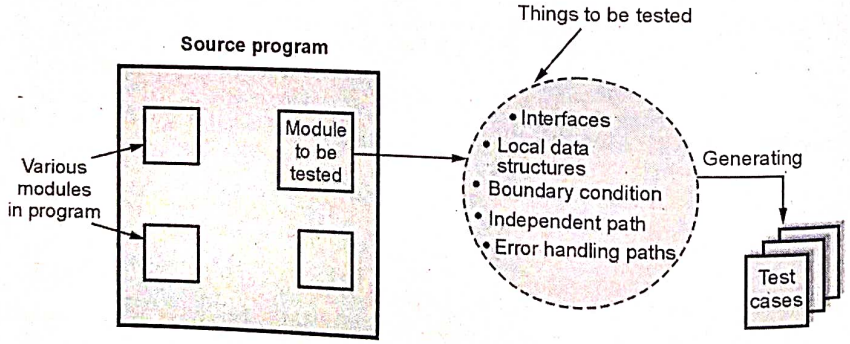


Fig. 4.6.1 Unit testing

6. Drivers and stub software need to be developed to test incomplete software. The "driver" is a program that accepts the test data and prints the relevant results. And the "stub" is a subprogram that uses the module interfaces and performs the minimal data manipulation if required. This is illustrated by following Fig. 4.6.2.

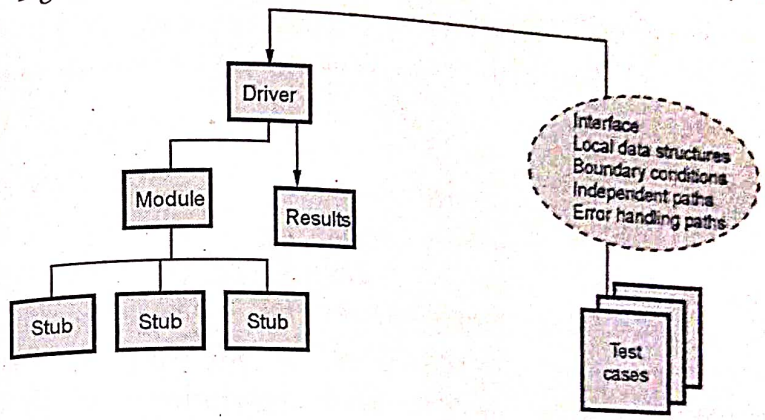


Fig. 4.6.2 Unit testing environment

7. The unit testing is simplified when a component with high cohesion (with one function) is designed. In such a design the number of test cases are less and one can easily predict or uncover errors.

4.7 Integration Testing

AU : Dec.-06,08,15,19, May-03,04,05,09,13,14,15,17,18, Marks 16

- A group of dependent components are tested together to ensure their quality of their integration unit.
- The objective is to take unit tested components and build a program structure that has been dictated by software design.
- The focus of integration testing is to uncover errors in :
  - Design and construction of software architecture.
  - Integrated functions or operations at subsystem level.
  - Interfaces and interactions between them.
  - Resource integration and/or environment integration.
- The integration testing can be carried out using two approaches.

1. The non-incremental integration 2. Incremental integration.

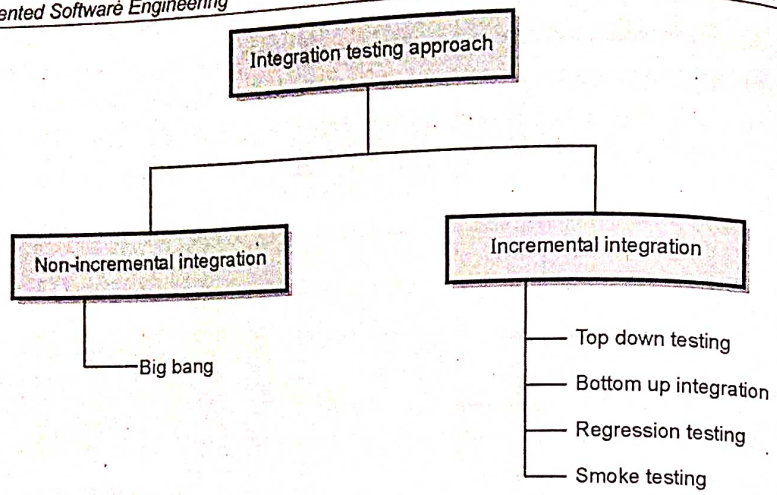


Fig. 4.7.1 Integration testing approach

- The non-incremental integration is given by the **"big bang"** approach. All components are combined in advance. The entire program is tested as a whole. And chaos usually results. A set of errors is tested as a whole. Correction is difficult because isolation of causes is complicated by the size of the entire program. Once these errors are corrected new ones appear. This process continues infinitely.

Advantage of big-bang : This approach is simple.

Disadvantages :

1. It is hard to debug.
2. It is not easy to isolate errors while testing.
3. In this approach it is not easy to validate test results.
4. After performing testing, it is impossible to form an integrated system.

- An incremental construction strategy includes,

  Top down integration

  Bottom up integration

  Regression testing

  Smoke testing.

The outcome of integration testing is

- Errors in design and construction of software architecture.
- Errors from integrated functions or operations at sub-system level.
- Errors from interfaces and interactions between them.
- Errors in resource integration and environment integration.

- The system test is a series of tests conducted to fully exercise the computer based system.

4.7.1 Top Down Integration Testing

- Top down testing is an incremental approach in which modules are integrated by moving down through the control structure.
- Modules subordinate to the main control module are incorporated into the system in either a depth first or breadth first manner.
- Integration process can be performed using following steps.
  1. The main control module is used as a test driver and the stubs are substituted for all modules directly subordinate to the main control module.
  2. Subordinate stubs are replaced one at a time with actual modules using either depth first or breadth first method.
  3. Tests are conducted as each module is integrated.
  4. On completion of each set of tests, another stub is replaced with the real module.
  5. Regression testing is conducted to prevent the introduction of new errors.

For example :

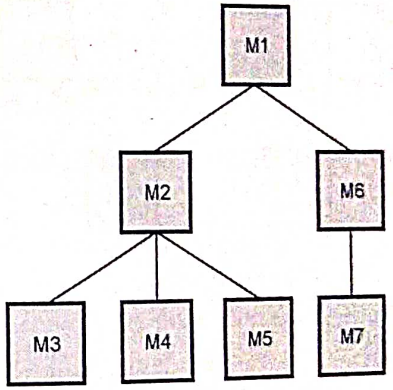


Fig. 4.7.2 Program structure

In top down integration if the depth first approach is adopted then we will start integration from module M1 then we will integrate M2 then M3, M4, M5, M6 and then M7.

If breadth first approach is adopted then we will integrate module M1 first then M2, M6. Then we will integrate module M3, M4, M5 and finally M7.

4.7.2 Bottom Up Integration Testing

In bottom up integration the modules at the lowest levels are integrated at first, then integration is done by moving upward through the control structure.

The bottom up integration process can be carried out using following steps.

1. Low-level modules are combined into clusters that perform a specific software subfunction.
2. A driver program is written to co-ordinate test case input and output.
3. The whole cluster is tested.
4. Drivers are removed and clusters are combined moving upward in the program structure.

For example :

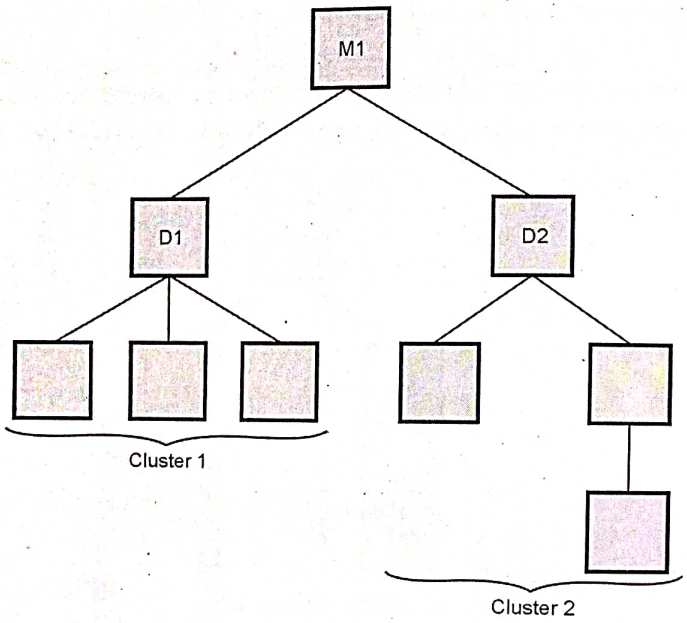


Fig. 4.7.3 Bottom up integration testing

First components are collected together to form cluster 1 and cluster 2. Then each cluster is tested using a driver program. The clusters subordinate the driver module. After testing the driver is removed and clusters are directly interfaced to the modules.

Difference between top down and bottom up integration testing :

Sr. No.	Top-down Integration	Bottom-up Integration
1.	The major controls or decisions are verified at early stage itself.	After integrating all the components at the bottom level, major controls or decisions can be verified.
2.	Testing can be performed from the early stage.	Individual components can be tested at higher level, but after integration of low level components into cluster testing is required.
3.	In top-down testing, testing stubs are created.	In botom-up testing, test drivers are required.
4.	Working system can be available at early stage.	Working system is available only after integrating all the components.

4.7.3 Regression Testing

- Regression testing is used to check for defects propagated to other modules by changes made to existing program. Thus regression testing is used to reduce the side effects of the changes.
- There are three different classes of test cases involved in regression testing -
  - Representative sample of existing test cases is used to exercise all software functions.
  - Additional test cases focusing software functions likely to be affected by the change.
  - Tests cases that focus on the changed software components.
- After product had been deployed, regression testing would be necessary because after a change has been made to the product an error that can be discovered and it should be corrected. Similarly for deployed product addition of new feature may be requested and implemented. For that reason regression testing is essential.

4.7.4 Smoke Testing

- The smoke testing is a kind of integration testing technique used for time critical projects wherein the project needs to be assessed on frequent basis.
- Following activities need to be carried out in smoke testing -
  1. Software components already translated into code are integrated into a "build". The "build" can be data files, libraries, reusable modules or program components.
  2. A series of tests are designed to expose errors from build so that the "build" performs its functioning correctly.

3. The "build" is integrated with the other builds and the entire product is smoke tested daily.

Smoke testing benefits

1. Integration risk is minimized.
2. The quality of the end product is improved.
3. Error diagnosis and correction are simplified.
4. Assessment of progress is easy.

Review Questions

1. *What is meant by integration testing and system testing ? Explain. Discuss on their outcomes.* **AU : May-03, 04, 05, Marks 16**
2. *What is integration testing ? Discuss any one method in detail.* **AU : May-17, Marks 8**
3. *Explain unit testing and integration testing process with an example.* **AU : Dec.-15, Marks 16**
4. *Write short note on - Regression testing.* **AU : May-13, Marks 8, May-18, Marks 6**
5. *Elaborate path testing and regression testing with an example.* **AU : Dec.-19, Marks 13**
6. *Discuss about the integration techniques with suitable example.* **AU : Dec.-22, Marks 7**

4.8 System Testing

AU : May-03,04,05, Dec.-14, Marks 16

The system test is a series of tests conducted for fully the computer based system.

Various types of system tests are

1. Recovery testing
2. Security testing
3. Stress testing
4. Performance testing

The main focus of such testing is to test

- System functions and performance.
- System reliability and recoverability (recovery test).
- System installation (installation test).
- System behaviour in the special conditions (stress test).
- System user operations (acceptance test/alpha test).
- Hardware and software integration and collaboration.
- Integration of external software and the system.

4.8.1 Recovery Testing

- Recovery testing is intended to check the system's ability to recover from failures.
- In this type of testing the software is forced to fail and then it is verified whether the system recovers properly or not.
- For automated recovery then reinitialization, checkpoint mechanisms, data recovery and restart are verified.

4.8.2 Security Testing

- Security testing verifies that system protection mechanism prevent improper penetration or data alteration.
- It also verifies that protection mechanisms built into the system prevent intrusion such as unauthorized internal or external access or willful damage.
- System design goal is to make the penetration attempt more costly than the value of the information that will be obtained.

4.8.3 Stress Testing

- Determines breakpoint of a system to establish maximum service level.
- In stress testing the system is executed in a manner that demands resources in abnormal quantity, frequency or volume.
- A variation of stress testing is a technique called sensitivity testing.
- The sensitive testing is a testing in which it is tried to uncover data from a large class of valid data that may cause instability or improper processing.

4.8.4 Performance Testing

- Performance testing evaluates the run time performance of the software, especially real time software.
- In performance testing resource utilization such as CPU load, throughput, response time, memory usage can be measured.
- For big systems (e.g. banking systems) involving many users connecting to servers (e.g. using internet) performance testing is very difficult.
- Beta testing is useful for performance testing.

Review Question

1. *What is system testing ? Discuss types of system tests.* **AU : Dec.-14, Marks 12**

AU : May-05,16, Marks 16

4.9 Validation Testing

- The integrated software is tested based on requirements to ensure that the desired product is obtained.
- In validation testing the main focus is to uncover errors in
  - System input/output
  - System functions and information data
  - System interfaces with external parts
  - User interfaces
  - System behaviour and performance.
- Software validation can be performed through a series of **black box tests**.
- After performing the validation tests there exists two conditions.
  1. The function or performance characteristics are **according** to the **specifications** and are accepted.
  2. The requirement specifications are derived and the deficiency list is created. The **deficiencies** then can be **resolved** by establishing the proper communication with the customer.
- Finally in validation testing a review is taken to ensure that all the elements of software configuration are developed as per requirements. This review is called configuration review or audit.

4.9.1 Acceptance Testing

The acceptance testing is a kind of testing conducted to ensure that the software works correctly in the user work environment.

The acceptance testing can be conducted over a period of weeks or months.

The types of acceptance testing are

1. **Alpha test** - The alpha testing is a testing in which the version of complete software is tested by the customer under the supervision of developer. This testing is performed at developer's site. The software is used in natural setting in presence of developer. This test is conducted in controlled environment.

Advantages

1. The users of the system get systematic training to use the new system.
2. The acceptance of the system can be immediately known to the developers.

3. The primary problems in the identified and fixed in immediately before delivery of the product to the customer.
4. Any mis-understanding about the scope of the project or in requirements given by the customer can be cleared during this type of testing.

Disadvantages

1. In depth functionality cannot be tested as software is still under development stage. Sometimes developers and testers are dissatisfied with the results of alpha testing.
2. Alpha testing is conducted in lab environment or testing environment which may not be similar to real-life environment.
3. Data used during this testing may not be similar to the business data or actual data. Hence false image of working module can be created and actual problems remain hidden.

2. **Beta test** - The beta testing is a testing in which the version of software is tested by the customer without the developer being present. This testing is performed at customer's site. As there is no presence of developer during testing, it is not controlled by developer. The end user records the problems and report them to developer. The developer then makes appropriate modification.

Advantages :

1. Beta testing improves the product quality via customer feedback.
2. Any training requirement for use of the product can be identified during beta testing.
3. The testing is done in actual work environment without the interference of developer. Hence addition in requirements, changes in the requirements can be identified.
4. The security loopholes can be identified by the customer itself.
5. Beta testing allows the organization to test post-launched infrastructure.
6. Due to customer feedback product failure risk can be reduced.
7. Usability of the application can be identified and verified by the customer.

Disadvantages :

1. Finding right beta user and maintaining his participation can be real challenge.
2. Tests can not be management because beta testing is conducted in actual working environment and not in the organization.

Review Question

1. *Compare and contrast alpha and beta testing.* **AU : May-16, Marks 8**

4.10 Debugging Techniques

AU : Dec.-10,14, May-15, 18, Marks 4

Debugging is a process of removal of a defect. It occurs as a consequence of successful testing.

Debugging process starts with execution of test cases. The actual test results are compared with the expected results. The debugging process attempts to find the lack of correspondence between actual and expected results. The suspected causes are identified and additional tests or regression tests are performed to make the system to work as per requirement.

Common approaches in debugging are :

Brute force method - The memory dumps and run-time traces are examined and program with write statements is loaded to obtain clues to error causes.

In this method "Let computer find the error" approach is used.

This is the least efficient method of debugging.

Backtracking method - This method is applicable to small programs.

In this method, the source code is examined by looking backwards from symptom to potential causes of errors.

Cause elimination method - This method uses binary partitioning to reduce the number of locations where errors can exist.

Why debugging is so difficult ?

Following are some reasons that reveal why debugging is so difficult -

1. The symptoms of bug may be present at some part(module) of the program and its effect might be seen in some other module of the program. Hence tracing out the location of symptom becomes difficult.
2. Symptoms may be caused by software developers during the development process. Such symptoms are difficult to trace out.
3. The symptom may appear due to timing problems instead of processing problem.
4. If the developer corrects some error then the symptom may disappear temporarily.
5. The symptom can appear if some in-accuracies in the program are simply rounded off.

6. In real time systems, it is not possible to accurately reproduce the input conditions and this may lead to symptoms of bugs.

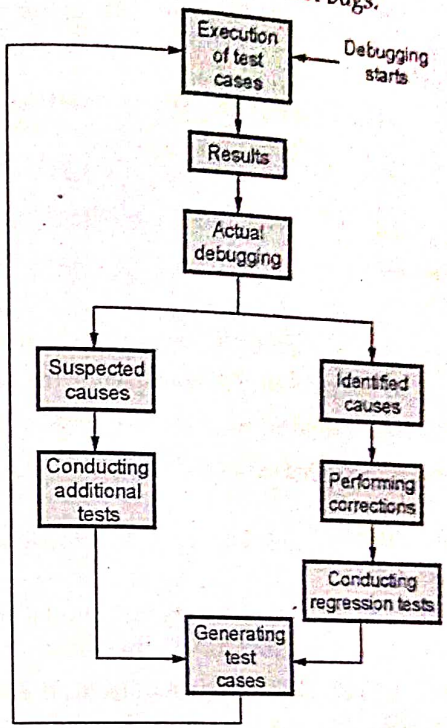


Fig. 4.10.1 Debugging process

7. The symptom may appear periodically. Such things normally occur in embedded systems in which hardware and software is coupled.
8. In distributed systems number of tasks are running on several distinct processors which may lead to symptoms.

4.10.1 Testing Vs. Debugging

Sr. No.	Testing	Debugging
1.	Testing is a process in which the bug is identified.	Debugging is the process in which the bug or error is corrected by the programmer.
2.	In testing process, it is identified where the bug occurs.	In debugging the root cause of error is identified.
3.	Testing starts with the execution results from the test cases.	Debugging starts after the testing process.

Review Question

1. *Write short note on - Debugging.* **AU : May-18, Marks 6**

4.11 Program Analysis

There are two ways by which the program analysis is carried out -

1) Static analysis
2) Dynamic analysis

➢ 1) Static analysis

- **Definition :** Static testing is a testing technique in which software is tested without executing the code. As the code, requirement documents, and design documents are tested manually in order to find errors, it is called static.
- This kind of testing is also called verification testing.

Static analysis testing techniques :

- **Informal reviews :**
  - In this technique, the team of reviewers just checks the documents and gives comments.
  - The purpose is to maintain the quality from the initial stage. It is non-documented in nature.
- **Formal reviews :**
  - It is well structured and documented and follows six main steps : Planning, kick-off, preparation, review meeting, rework follow-up.

  Technical reviews :

  The team of technical experts will review the software for technical specifications. The purpose is to pin out the difference between the required specification and product design and then correct the flaws. It focuses on technical documents such as test strategy, test plan and requirement specification documents.
- **Walk-through :**
  - The author explains the software to the team and teammates can raise questions if they have any. It is headed by the author and review comments are noted down.
- **Inspection process :**
  - The meeting is headed by a trained moderator. A formal review is done, a record is maintained for all the errors and the authors are informed to make rectifications on the given feedbacks.

- **Static code review :**
  - Code is reviewed without execution, it is checked for syntax, coding standards and code optimization. It is also referred to as white box testing.

Advantages :

1) It is a fast and easy technique used to fix errors.
2) It helps in identifying flaws in code.
3) With the help of automated tools it becomes very easy and convenient to scan and review the software.
4) With static testing it is possible to find errors at an early stage of the development life cycle.

Disadvantages :

1) It takes a lot of time to conduct the testing procedure if done manually.
2) Automated tools work for a restricted set of programming languages.
3) The automated tools simply scan the code and can not test the code deeply.

➢ 2) Dynamic analysis

Definition : Dynamic testing is a process by which code is executed to check how software will perform in a runtime environment. As this type of testing is conducted during code execution it is called dynamic. It is also called **validation testing.**

Dynamic testing techniques

- **Unit testing :** As the name suggests individual units or modules are tested. The source code is tested by the developers.
- **Integration testing :** Individual modules are clubbed and tested by the developers. It is performed in order to ensure that modules are working in the right manner and will continue to perform flawlessly even after integration.
- **System testing :** It is performed on a complete system to ensure that the application is designed according to the requirement specification document.

Advantages

1) It identifies weak areas in a runtime.
2) It helps in performing detailed analysis of code.
3) It can be applied with any application.

Disadvantages

1) It is not easy to find a trained software tester to perform dynamic testing.
2) It becomes costly to fix errors in dynamic testing.

Difference between static and dynamic analysis :

Sr. No.	Static analysis	Dynamic analysis
1.	Static analysis is a testing process that is done at the early stage of the development life cycle.	Dynamic analysis is a testing process done later stage of the development life cycle.
2.	It involves a walkthrough and code review.	It involves functional and non-functional testing.
3.	This testing is a verification process.	This testing is a validation process.
4.	This type of testing is done before the execution of code.	This type of testing is done during code execution.
5.	It is about prevention.	It is about a cure.
6.	It is cost-effective.	It is costly.

4.12 Symbolic Execution

- Symbolic execution is a testing technique in which we use symbols to see what is the output of the code without actually running the code. This helps to find and fix the errors in the software without doing a real test.
- Symbolic execution is a technique for testing software by symbolically evaluating the program's control flow and data flow. It is a type of dynamic analysis, which means that it tests the program by executing it with symbolic input values instead of concrete input values.
- Symbolic execution involves analyzing the code without running it with actual values.
- For example -

```
function divide(int a, int b)
{
   if(b!=0)
     result = a/b;
   else
     result = -1;
   return result
}
```


In above code,

The **result** is a symbolic variable we can analyze the code's behavior in both cases without having specific values for 'a' and 'b' at this stage. This helps us understand how the code will react to various scenarios.

- Symbolic execution is often used in more complex programs to find the major bugs.
- This technique also helps us to understand how the code responds to different inputs without running it with actual data.
- It is an important technique for debugging and testing software to identify possible problems and ensure robustness.

Review Question

1. *Write a short note on - Symbolic execution.*

4.13 Model Checking

Model checking is a technique for verifying that a software system satisfies the desired properties. In this technique, a model of the system is created and then using a model checker we exhaustively explore all possible states of the model to check the properties of the system.

Model checking can be used in both software testing and debugging. In software testing, it can be used to verify that the system meets its requirements. In software debugging, it can be used to identify the root cause of a bug.

Once the model and the desired properties have been specified, a model checker can be used to verify that the model satisfies the properties.

For example - Consider a ATM system in which we have to design a function for withdrawal of money say **withdraw()**. This function allows the user to withdraw money from his/her account. In order to implement this function, following requirements need to be satisfied -

1. The user must have a positive balance in their account in order to withdraw money.
2. The user cannot withdraw more money than they have in their account.
3. The user's balance must be updated after each withdrawal.

Before actually executing the code, we can use model checking technique to verify that the withdraw() function meets these requirements. In this technique, we first create a model of the **withdraw()** function. We must specify the above requirements as **properties** of that model. The model checker would explore all possible states of the function to

verify that the function satisfies all these properties. For instance - If there is a negative balance, then there must not be a path to 'withdraw amount' state. Similarly if the balance amount greater than the withdrawal amount then only there should be a path to 'withdraw amount' state. If the model checker finds a state in which the function does not update the user's balance after a withdrawal then it should report a bug. By identifying this issue in our model, we can locate the corresponding code in our software and correct the logic to prevent this scenario from happening.

Model checking is particularly valuable for complex systems, where manually testing all possible combinations and paths would be impractical.

4.14 Two Marks Questions with Answers

Q.1 State the guidelines for debugging. AU : Dec.-04, 05, May-03

Ans. : Following are some debugging guideline suggested by Myers

1. Debugging can be effectively carried out by mental analysis.
2. If an error cannot be located by yourself, then describe the problem to someone else.
3. Use debugging tool only if no alternative is present.
4. Avoid experimentation.
5. If one bug occurs in one area of code then there are chances of occurrence of more bugs.
6. Fix the error and not its symptoms.
7. For correcting the existing program, if a new code is added to it then test it rigorously than the original program.
8. There are chances of introducing new errors on correction of older ones.
9. The process error repair must be conducted in parallel during design phase.
10. Change the source code and not the object code/machine code.

Q.2 What are static and dynamic software testing ? AU : Dec.-05

Ans. : The verification activities fall into the category of static testing. During static testing, you have a checklist to check whether the work you are doing is going as per the set standards of the organization.

The dynamic testing is a method in which the actual testing is done to uncover all possible errors. Various testing strategies such as unit testing, integration testing, validation testing, system testing can be applied while performing dynamic testing.

Q.3 What is stress testing ? AU : May-03

Ans. : Stress testing is a testing for a system which is executed in a manner that demands resources in abnormal quantity, frequency or volume.

A variation of stress testing is a technique called sensitivity testing.

Q.4 What is partial integration testing ? AU : Dec.-03

Ans. : Partial integration testing verifies correct data movement from one external system to another.

For instance : If there is one system that brings data from single external system and then sends the data to several other external systems. Then partial test of this system would be to test whether the data can be sent to one of the external systems.

Q.5 Why testing is important with respect to software ? AU : May-04

Ans. : The purpose of software testing is to ensure whether the software functions appear to be working according to specifications and performance requirements. This ultimately helps in improving the overall quality of the software.

Q.6 How regression and stress tests are performed ? AU : Dec.-04

Ans. : Regression testing is used to check for defects propagated to other modules by changes made to existing program. Thus regression testing is used to reduce the side effects of the changes.

Stress testing determines breakpoint of a system to establish maximum service level. In stress testing the system is executed in a manner that demands resources in abnormal quantity, frequency or volume.

Q.7 Define cyclomatic complexity. AU : May-06, 14

Ans. : Cyclomatic complexity is kind of software metric that gives the quantitative measure of logical complexity of a program. In basis path testing method the cyclomatic complexity defines the number of independent paths of the program structure.

Q.8 Write the types of system tests. AU : May-06

Ans. : Various types of system tests are

1. Recovery testing
2. Security testing
3. Stress testing
4. Performance testing

Q.9 Write short note on equivalence partitioning. AU : May-06

Ans. : In equivalence partitioning the equivalence classes are evaluated for given input condition. Equivalence class represents a set of valid or invalid states for input conditions.

It is a black-box technique that divides the input domain into classes of data. From this data test cases can be derived.

Q.10 What is white box testing and what is the difficulty while exercising it ? AU : May-05

Ans. : In white box testing the procedural details are closely examined. In this testing the internals of software are tested to make sure that they operate according to specifications and designs. Thus major focus of white box testing is on internal structures, logic paths, control flows, data flow, internal data structures, conditions, loops, etc.

This kind of testing will be exhaustive testing. This type of testing will keep the processor continuously busy. And software developer will find it difficult to manage the schedule of the project with white box testing. The white box testing is impossible for large system.

Q.11 What is behavioural testing ? AU : May-06

Ans. : The black box testing is also called as behavioural testing.

Black box testing methods focus on the functional requirements of the software. Tests sets are derived that fully exercise all functional requirements.

Q.12 What is cyclomatic complexity? How to calculate this ? AU : Dec.-06

Ans. : The cyclomatic complexity defines the number of independent paths in the basis set of the program that provides the upper bound for the number of tests that must be conducted to ensure that all the statements have been executed atleast once.

The cyclomatic complexity can be computed by one of the following ways.

1. The number of regions of the flow graph correspond to the cyclomatic complexity.
2. Cyclomatic complexity, V(G), for a flow graph, G, is defined as :

   $V(G) = E - N + 2$,

   E = Number of flow graph edges

   N = Number of flow graph nodes
3. $V(G) = P + 1$

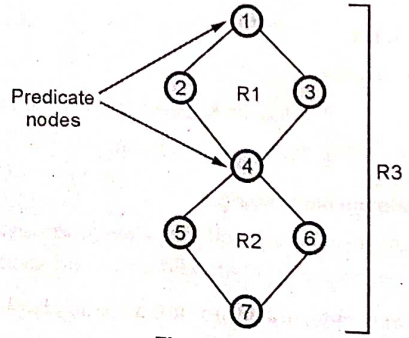


Fig. 4.14.1

where P is the number of predicate nodes contained in the flow graph G.

Consider the flow graph as shown below.

The cyclomatic complexity can be computed as -

1. The flow graph has 3 regions.
2. V(G) = 8 edges – 7 nodes + 2 = 3
3. V(G) = 2 predicate nodes + 1 = 3

Hence cyclomatic complexity is 3.

Q.13 Write the steps involved in testing real time systems. AU : May-07

Ans. : Following are the steps involved in testing real time systems.

1. **Task testing** : This is the first step in testing real time systems. In this step, each task is independently tested. Task testing uncovers the errors in logic and function. But it does not find errors from behaviour.
2. **Behavioural testing** : With the help of automated tool the behaviour of the real time system is simulated. Then it becomes possible to examine the behaviour of the system as a series of external events.
3. **Inter task testing** : After finding and removing the errors from individual task and behaviour of the system the next step is to obtain errors from asynchronous tasks that may communicate with one another. Such type of testing is called inter task testing.
4. **System testing** : When all the required hardware and software is integrated then a full range of testing is carried out in order to uncover errors at hardware and software interface. As real time system is based on interrupts therefore testing the Boolean events is necessary.

Q.14 What is the objective of unit testing ? AU : Dec.-06

Ans. : The objective of unit testing is to test individual components independently to ensure their quality. Thus the focus is to uncover the errors in design and implementation.

Q.15 Assume a program for computing the roots of a quadratic equation. List out the test cases using equivalence partitioning method. AU : May-07

Ans. : To write the test cases for roots of quadratic equations we will first write a program for computing the roots for quadratic equation -

```
Void Roots_of_QE (float a, float b, float c)
 {
   float d;
   d = (b * b) – (4 * a * c);
   if (d < 0)
   {
     printf ("Roots are complex");
     r1 = (– b + sqrt (d)) / (2 * a);
```

```
    r2 = (– b – sqrt (d)) / (2 * a);
    printf ("%f %f ", r1, r2);
   }
   else if (d = = 0)
   {
    printf (" Single real root");
    r1 = – b / 2 * a;
    printf ("%f ", r1);
   }
   else
   {
    printf (" Two real roots");
    r1 = (– b + sqrt (d)) / (2 * a);
    r2 = (– b – sqrt (d)) / (2 * a);
    printf (" %f %f ", r1, r2);
   }
}
```

Now the test cases can be written as follows.

Precondition : The quadratic equation $ax^2 + bx + c$ is to be entered. This will set a, b and c.

Test case Id	Test case name	Test case description	Test steps			Test case status (P/F)
			Steps	Expected	Actual	
1.	Check for complex roots	For a quadratic equation, complex roots exist if $d = b^2 - 4ac$ is < 0.	Compute d = (b * b) – (4 * a * c) check if (d < 0)	If (d < 0) the message "Roots are complex" will be displayed. And then roots r1 and r2 will be printed.	For equation : $x^2 + x + 2 = 0$ $r1 = \frac{-1+\sqrt{-7}}{2}$ $r2 = \frac{-1-\sqrt{-7}}{2}$ Then solve (x – r1) * (x – r2)	Pass
2.	Check for single real root	For a quadratic equation, single real root exists if $d = b^2 - 4ac$ is equal to 0.	Compute d = (b * b) – (4 * a * c) check if (d = = 0)	If (d is equal to 0) then the message "Single real root" will be displayed and only one root r1 will be printed.	For equation : $9x^2 - 24x + 16 = 0$ r1 = 4/3 Then solve $(x - r1)^2 = 0$ we will get $9x^2 - 24x + 16 = 0$	Pass
3.	Check for two real roots	For a quadratic equation, two real roots exist if $d = b^2 - 4ac$	Compute d = (b * b) – (4 * a * c) check if (d > 0)	If (d > 0) then the message "Two real roots" will be	For equation : $x^2 + 2x - 2 = 0$ $r1 = -1 + \sqrt{3}$ $r2 = -1 - \sqrt{3}$	Pass

is greater than 0.	printed.	Then solve (x – r1) * (x – r2) and we will get $x^2 + 2x - 2$

Q.16 What are the steps for top-down integration ? AU : Dec.-07

Ans. : Following are the steps in top-down integration.

1. The main control module is used as a test driver and stubs are substituted for all the modules directly subordinate to the main control module.
2. Subordinate stubs are replaced one at a time with actual modules using either depth first or breadth search method.
3. Tests are conducted as each module is integrated.
4. On completion of each set of tests, another stub is replaced with the real module.
5. Regression testing is conducted to prevent the introduction of new errors.

Q.17 Mention any two characteristics of software testing. AU : May-08

OR Write down generic characteristics of software testing. AU : May-13

Ans. : Following are the characteristics of software testing :

1. Testing should be conducted by the developer as well as by some independent group.
2. The testing must begin at the component level and must work outwards towards the integration testing.
3. Different testing must be appropriate at appropriate times.

Q.18 Which is called as a glass box testing ? What is the objective of this ? (Refer section 4.4) AU : Dec.-07

Q.19 List out the data structure errors identified during the unit testing. AU : Dec.-07

Ans. : Various data structure errors that can be identified during the unit testing are -

1. Incorrect arithmetic precedence.
2. Mixed mode operations.
3. Precision inaccuracy.
4. Comparison of different data types.

Q.20 Calculate the cyclomatic complexity for the following program. Explain your approach.

```
int temp
(a>b) temp a
else temp b
```

```
if (c>temp)
temp=c
return temp
```

AU : May-08

Ans. : The flow graph for given program is -

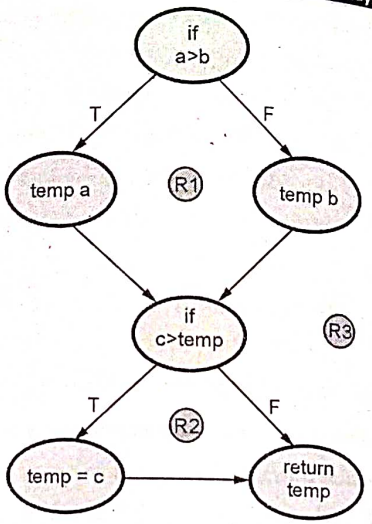


Fig. 4.14.2

Cyclomatic complexity
= E – N + 2
= 7 – 6 + 2 = 3

Cyclomatic complexity
= P + 1
= 2 + 1 = 3

Cyclomatic complexity
= Regions encountered
= 3

Hence cyclomatic complexity of given program is **3.**

Q.21 What is a critical module and why should we identify it ? AU : Dec.-08

Ans. : The module which possess following characteristics is called critical module.

1. The critical module addresses several software requirements.
2. This module has definite performance requirements.
3. It has high level of control.
4. This module is complex and error prone.

Because of above mentioned characteristics, it is necessary to identify and test such module in order to make the system errorfree and as per user requirements.

Q.22 What is meant by smoke testing ? *(Refer section 4.7.4)* AU : Dec.-08

Q.23 What is the difference between alpha testing and beta testing ? AU : May-09

Sr. No.	Alpha testing	Beta testing
1.	This testing is done by a developer or by a customer under the supervision of developer in company's premises.	This testing is done by the customer without any interference of developer and is done at customer's place.
2.	Sometime full product is not tested using alpha testing and only core functionalities are tested.	The complete product is tested under this testing. Such product is usually given as free trial version.

Q.24 Will the exhaustive testing guarantee that the program is 100 % correct ? AU : May-09, 16

Ans. : No the exhaustive testing will not guarantee that the program is 100 % correct. This is because there are several parameters that affect the correctness of the program. For instance -

User acceptance - It is not guaranteed that the program is strictly as per users' expectation and will work as per users' design.

Performance testing - It is necessary to test whether the program performs to a level that was satisfactory. Similarly the performance should not get degraded if volume load gets increased.

Integration testing - This testing needs to be done to check whether the program works properly if it gets integrated with some other software component.

Besides these testing there are many other types of testing that needs to be carried out on the program, and there are chances of a program getting failed in one or more testing.

Q.25 What is meant by integration testing ? *(Refer section 4.7)* AU : May-09

Q.26 State any four software testing principles. *(Refer section 4.1.2)* AU : Dec.-09

Q.27 What is an unit ? AU : May-10

Ans. : Unit is a smallest testable part in the application. In procedural programming, unit may be an individual program, function or procedure. While in object oriented programming, the smallest unit is a method, which may belong to a base class, abstract class or derived class.Ideally independent cases can be written for each unit(function) of a source program.

Q.28 In unit testing of a module, it is found for a set of test data, at maximum 90 % of the code alone were tested with the probability of success 0.9. What is the reliability of the module ? AU : Dec.-11

Ans. : The reliability of the module is at the most 0.81. For the given set of test data the 90 % of the code is tested and probability of success is given as 0.9 i.e. 90 %.

Hence Reliability = (90/100)*0.9 = 0.81

Q.29 With a simple example establish that 100 % testing is impossible.

Ans. : It is not possible to test all inputs or all permutations and combinations of all inputs. It is not even possible to test all paths of even a moderate system. For example - Suppose we have to test a function that handles verification of user password. Each user on each

computer must have a password. Each password might be atleast 6 to 8 characters long. It might be in capital letters, alphanumeric or in lower case. Then it is just impossible to consider various permutations and combinations for password verification.

Q.30 Distinguish between stress and load testing. AU : May-12

Ans. : Load testing is conducted to measure system performance based on volume of users. On the other hand the stress testing is conducted for measuring the breakpoint of a system when extreme load is applied. In load testing the expected load is applied to measure the performance of the system whereas in stress testing the extreme load is applied.

Q.31 What is a big-bang approach ? AU : Dec.-12

Ans. : Big-bang approach is used in non incremental integration testing. In this approach of integration testing, all the components are combined in advance and then entire program is tested as a whole.

Q.32 How are the software testing results related to the reliability of the software ? AU : Dec.-12

Ans. : During the software testing the program is executed with the intention of finding as much errors as possible. The test cases are designed that are intended to discover yet-undiscovered errors. Thus after testing, majority of serious errors are removed from the system and it is turned into the model that satisfied user requirements.

Q.33 What are the levels at which the testing is done ? AU : Dec.-13

Ans. : Testing can be done in the two levels as given below -

1. **Component level testing :** In the component level testing the individual component is tested.
2. **System level testing :** In system testing, the testing of group of components integrated to create a system or subsystem is done.

Q.34 What are the classes of loops that can be tested ? AU : May-14

Ans. : Various classes of loop testing are -

1. Simple loop
2. Nested loop
3. Concatenated loop
4. Unstructured loop.

Q.35 What is validation testing ? AU : Dec.-14

Ans. : Validation testing is a testing which is meant to validate the requirements as defined in the Software Requirements Specification (SRS). It ensures that the application as developed matches with the requirements defined and is fit for intended use.

Q.36 What is the need for regression testing ? AU : May-15

Ans. : Regression testing is required to test the defects that get propagated from one module to another when changes are made to existing program. Thus regression testing is used to reduce the side effects of changes.

Q.37 How will you test simple loop ? AU : Dec.-15

Ans. : The simple tests can be performed for n number of classes.

i) If n = 0 then skip the loop completely.

ii) If n = 1 then one pass through the loop is tested.

iii) If n = m then testing should be done for m passes where m < n.

iv) Perform the testing when number of passes are n – 1, n, n + 1.

Q.38 Why does software fail after it has passed from acceptance testing ? AU : May-16

Ans. : During acceptance testing, the random input is used for testing. This may lead to the situation that some input values that may cause failure go unhandled. The practical problem with acceptance testing is that it is time consuming. Hence in order to keep testing cost low, there are restricted number of test cases.

Hence there are high chances of software failure even after passing the acceptance testing.

Q.39 What is validation testing ? AU : Dec.-14

Ans. : Validation testing is a testing which is meant to validate the requirements as defined in the Software Requirements Specification(SRS). It ensures that the application as developed matches with the requirements defined and is fit for intended use.

Q.40 What is the need for regression testing ? AU : May-15

Ans. : Regression testing is required to test the defects that get propagated from one module to another when changes are made to existing program. Thus regression testing is used to reduce the side effects of changes.

Q.41 What is smoke testing ? AU : May-17

Ans. : The smoke testing is a kind of integration testing technique used for time critical projects where in the projects need to be assessed on frequent basis.

Q.42 Mention the purpose of stub and driver used for testing. AU : Dec.-17

Ans : Driver : The "driver" is a program that accepts the test data and prints the relevant results.

Stub : The "stub" is a subprogram that uses the module interfaces and performs the minimal data manipulation if required.

Q.43 What are the testing principles the software engineer must apply while performing the software testing ? AU : May-18

Ans. :

1) All tests must be traceable to customer requirements.
2) Tests should be planned long before testing begins.
3) Testing should begin in small and progress towards testing in large.
4) Exhaustive testing is not possible.

5) Testing should be done independent third party.

Q.44 Identify the type of maintenance for each of the following :

a) Correcting the software faults

b) Adapting the change in environment.

AU : May-18

Ans. : a) Corrective maintenance **b)** Adaptive maintenance

AU : May-19

Q.45 List the levels of testing.

Ans. : 1) Unit testing 2) Integration testing 3) Validation testing 4) System testing.

AU : Dec.-19

Q.46 What is test case ?

Ans. : Test case is a document or a specification of inputs, execution conditions, testing procedure and expected results while executing every single test.

AU : Dec.-19

Q.47 Outline the need for system testing.

Ans. : The system testing is needed for testing all the system elements so that the system as a whole can be tested.

Q.48 How black box testing is different from white box testing ? *(Refer section 4.4)*

AU : Dec.-20

AU : May-22, Dec.-22

Q.49 Define regression testing.

Ans. : Regression testing is used to check for defects propagated to other modules by changes made to an existing program. This testing is used to reduce the side effects of the changes.

❑❑❑