Software Testing And Quality Assurance

(STQA)

IT-Engineering (Semester 7)

Index

Module 1 || Testing Methodology

Index

1.1 Software Testing

Software testing is nothing but testing/ investigating software to ensure its quality with respect to requirements of clients.
It is carried out in a systematic manner with the intent of finding defects in the system.It is required for evaluating the system.
One small defect can cause a lot of financial loss. It is for this reason that software testing is now emerging as a very powerful field in IT.

1.2 What are Software Testing Methodology

Software testing methodology in software engineering are testing strategies approaches or methods used to test specific product to ensure its usability
It makes us that the application or the product works as per given specifications and has no side-effect when used outside the design parameter
It encompasses everything from unit testing to integration, testing and specialized forms of testing like security, testing or performance testing.

1.3 Common software testing terms

1. Failure
   →The inability of the system or Component to perform a required function according to its specifications
2. Fault/Defect/Bug
   →It is a condition that actual causes a system to produce failure
3. Error
   →Whenever a member of the development team makes any mistake in any phase of SDLC, errors are produced.
   Thus error is a very general term used for human mistakes
   It might be a typographical error, misleading specifications, misleading off what a subroutine does and so on.
   (In general, a mistake in coding is called error)

4. Testware

→The documents created during the testing are known as testware

5. Incident
   →The symptoms associated with a failure, that alerts the user to the occurrence of a failure
6. Test Oracle
   →To judge the success or failure of a test.

1.4 Software Testing Life Cycle (STLC)

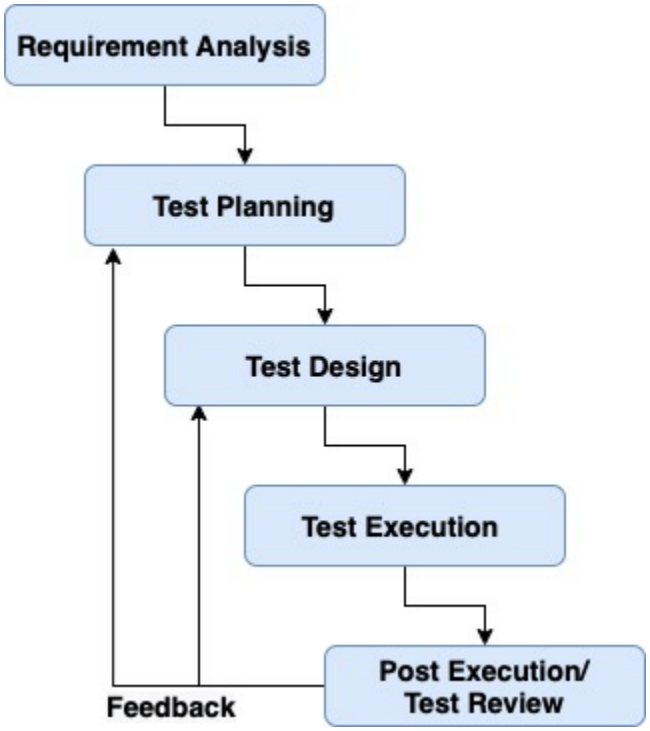


Fig 1.1 STLC

1. **Requirement Analysis →**
   In this testing team studies the requirements from a testing point of view. Requirements could be functional or non-functional. Automation feasibility for the testing project is also done in this stage.

2. **Test Planning →**

It's the phase in which a senior QA manager determines the test plan strategy along with efforts & cost estimation for the project. Moreover the resources, test environment, Test limitations and test schedule.

3. **Test design →**
   Involves the creation and verification of test cases & test - scripts after the test plan is ready. It also involves creation of test data
4. **Test execution →**
   After test-case design and environment setup, the test execution phase gets started. In this phase the Testing team starts executing test cases based on test cases prepared in earlier phases.
5. **Post Execution/ Test review →**
   Analysis is done & Feedback is sent…

1.5 Exhaustive and Effective Software testing

Exhaustive Software testing	**Effective Software testing**
● Complete or Exhaustive software testing means there are no undiscovered faults at the end of the test phase. (All problems must be known at the end of the testing phase).	● Effective software testing provides experienced based practices and key concepts that can be used by an organization to implement a successful and efficient testing program.
● It's not possible to perform complete testing (practically not possible)	● Practically possible.
● It is not feasible because → ➢ Achieving deadlines ➢ Various possible outputs ➢ Time constraints ➢ Number of possible test environments	● It is feasible because → ➢ It checks for software reliability and no bugs in final product ➢ It tests in each phase ➢ It uses constrained resources

• It is not cost effective • Complex and time consuming • It concentrates all test cases	• It is cost effective • Less complex and time consuming • It is adopted such that critical test cases are concerned first

1.6 Goals of Software testing

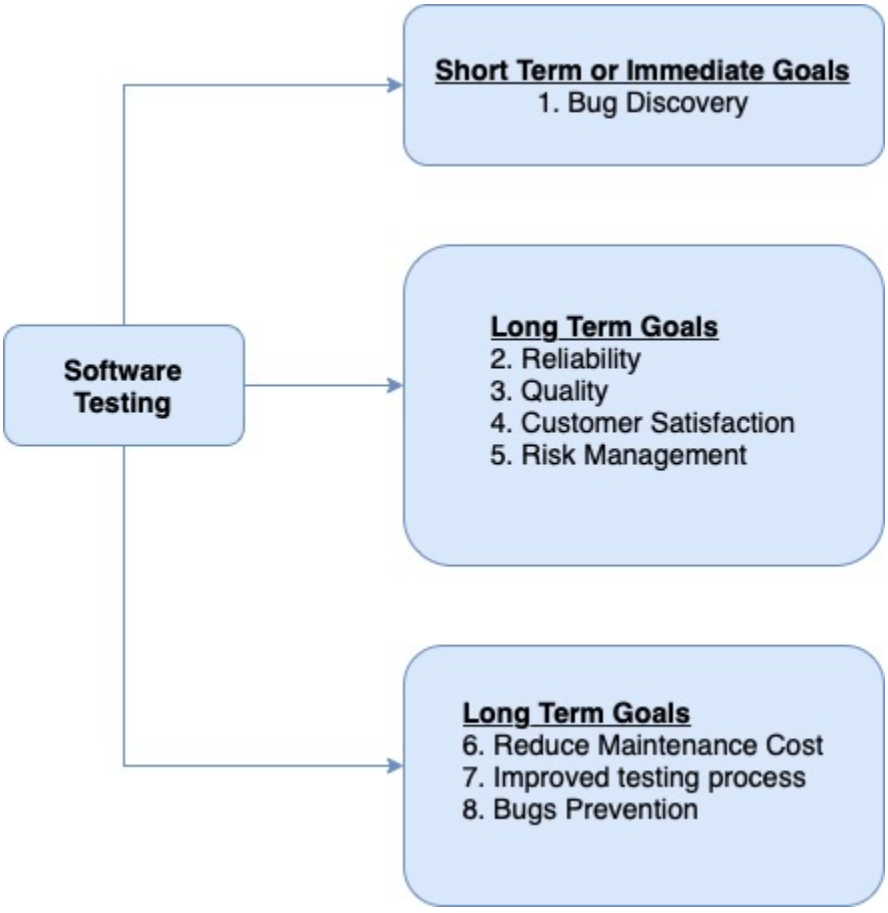


Fig 1.2 Software Testing goals

1. **Bug discovery**

The immediate goal of software testing is to find errors at any stage of software development. more the book discovery at early stage, battery will be the success rate of software testing.

2. **Quality**
   Since the software is also a product it's quality is primary from the user's point of view. Though quality depends on various factors such as correctness, integrity, efficiency, etc , **reliability** is the major factor to achieve quality. So the software should be passed through a reliability Analysis to attend high quality standards.
3. **Customer satisfaction**
   If we want customers to be satisfied with the software product, then testing should be complete and throughout . Testing should be completed in the sense that it must satisfy the user For all the specified and the unspecified requirements which are otherwise do understood.
4. **Risk Management**
   Risk is the probability that undesirable events will occur in a system. These undesirable events will prevent the organization from successful implementation of business activities.
5. **Reduced maintenance cost**
   The maintenance cost of any software product is not its physical cost as software does not wear out, the only maintenance cost in a software product is its failure due to errors.
   Post-release, errors are costlier to fix as they are difficult to detect .
   If software testing is done effectively, then chances of fillers are minimized and intern maintenance cost is reduced .
6. **Improved Software Testing Processes**
   Software testing processes are different for different software products, continuously getting updated as there is always a scope for improvement.
7. **Bug prevention**
   It's the consequent action of bug discovery. As everyone in software development teams gets to learn how to code safely such that the same book discovery is not repeated in later stages, or future projects.

1.7.1 Verification in software testing

It is a process of checking documents, design, code and program in order to check if the software has been built according to the requirements or not. The main goal of the verification process in software testing is to ensure quality of software application, design, architecture etc.

1.7.2 Validation in software testing

It is a dynamic mechanism of testing and validating if the software product meets the exact needs of the customer or not.
This process helps to ensure that the software fulfills the desired use in an appropriate environment.

1.7.3 DIFFERENCE

Verification	**Validation**
● The verification process includes checking of documents, design, code and program ● It does not involve executing the code ● Verification uses methods like reviews, walkthrough, inspections,... etc ● Checks whether software conforms to specifications ● It finds bugs early in development cycle	● It is a dynamic mechanism of testing and validating the actual product ● It always involves executing the code ● Validation uses methods like black-box, white-box and non-functional testing ● Checks whether software meets requirements and expectations ● It finds bugs that verification process can not catch

● Target in application and software architecture, specifications, complete design, high level and database design,… etc ● It comes before validation	● Target is an actual product ● It comes after verification

1.8.1 Verification in High Level Design (HLD)

It is a general system design (means it refers to the overall system design).
It describes the overall description/ architecture of the application.
It includes the description of system architecture, database design, description of system, services, platforms and relationship among modules.

1.8.2 Verification in Low Level Design (LLD)

It is like detailing HLD (means it refers to the component level design process).
It describes details of each and every module in detail
It includes actual logic for every system component and goes deep into each module's specification.

1.8.3 DIFFERENCE

HLD	**LLD**
● High Level design ● It is the general system design (means it refers to overall system design). ● Also known as macro level system design ● Gives description of overall	● Low Level Design ● Its like detailed HLD (means it refers to component level design process) ● Also known as micro level system design ● Gives description of each and

application architecture ● Brief functionality of each module ● It is created first, means before LLD ● In HLD the input criteria is software-requirement-specification (SRS) ● It converts client requirements into HLD	every module. ● Detailed functionality of each module ● It is created second, after LLD ● In LLD input criteria is reviewed High-Level-design (HLD) ● It converts high level solution into detailed solution

1.9 V-Model

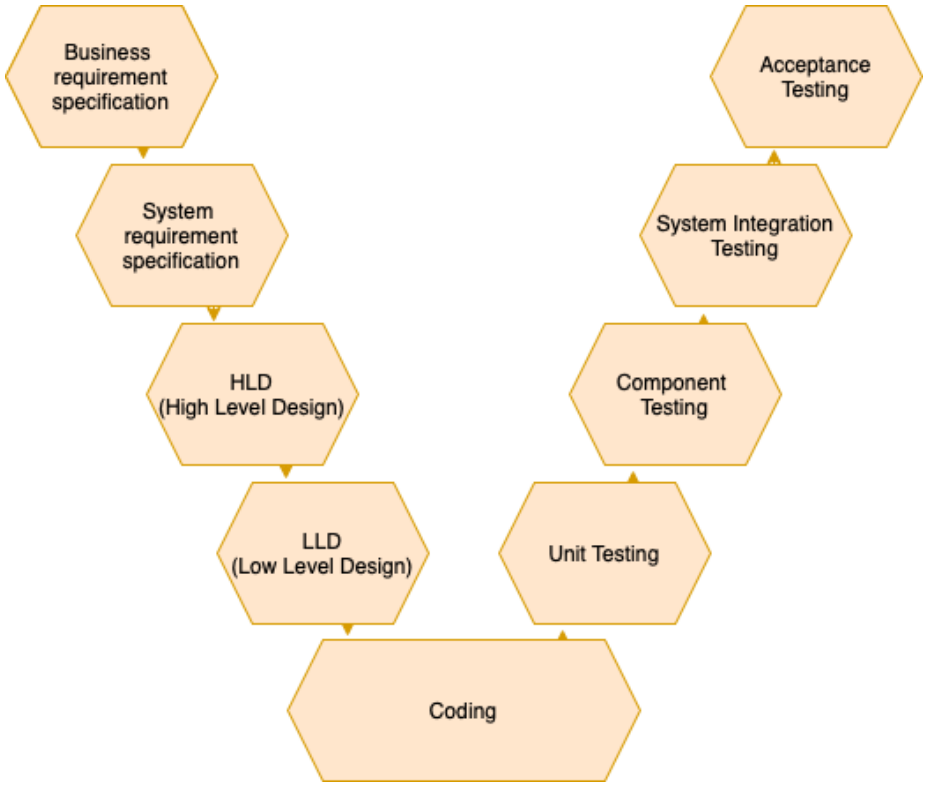


Fig 1.3 V-V Model

- Also referred to as the verification and validation model.
- Verification steps include → business requirements specification, system requirements specification, HLD, LLD and coding.

- Validation steps include → unit testing, component testing, system integration testing, acceptance testing.

1.10 Bug Life Cycle

Bug life cycle in software status is the specific set of states that bug goes through in its entire life.

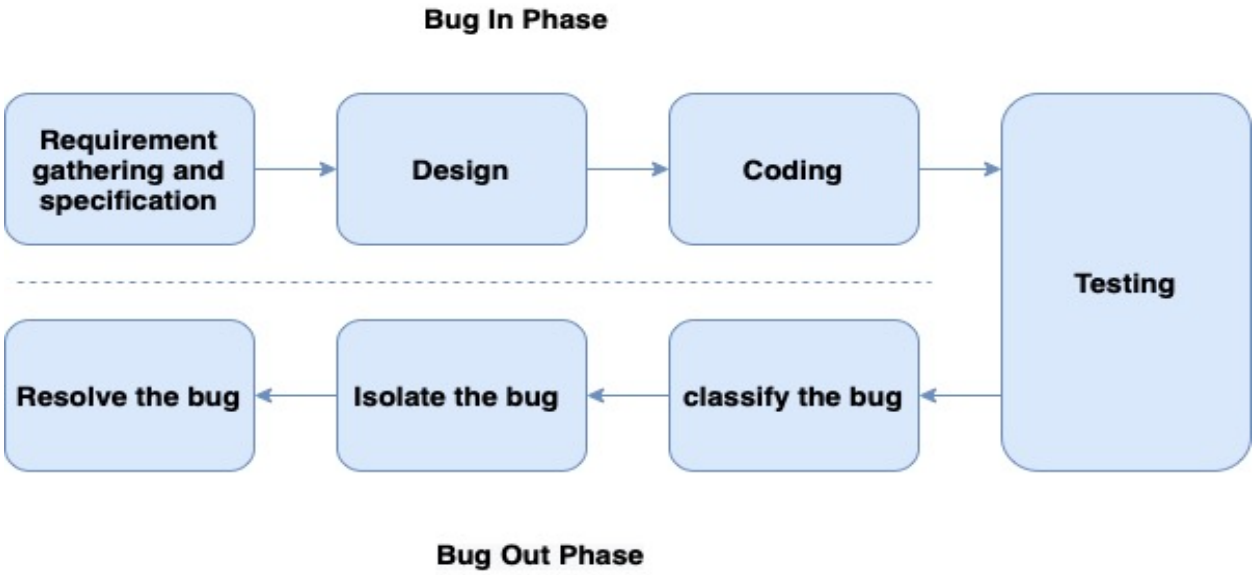


Fig 1.4 Bug Life Cycle

1.11 Bug Classification

1. Based on Criticality

- Critical Bugs → Worst effect on the functionality of software such that it hangs or stops normal functioning of the software.
- Major Bugs → This type of bugs does not stop the functioning of software but fails to meet its requirements as expected.
- Medium Bugs/ Minor Bugs → These bugs are less critical in nature as compared to major bugs.

2. Based on SDLC

- Design Bug → Control Flow Bugs, Logic Bugs, Processing Bugs, Data Flow bugs.
- Coding Bug → Interface and Integration Bugs, System Bugs, Testing Bugs.

1.12 States of a Bug

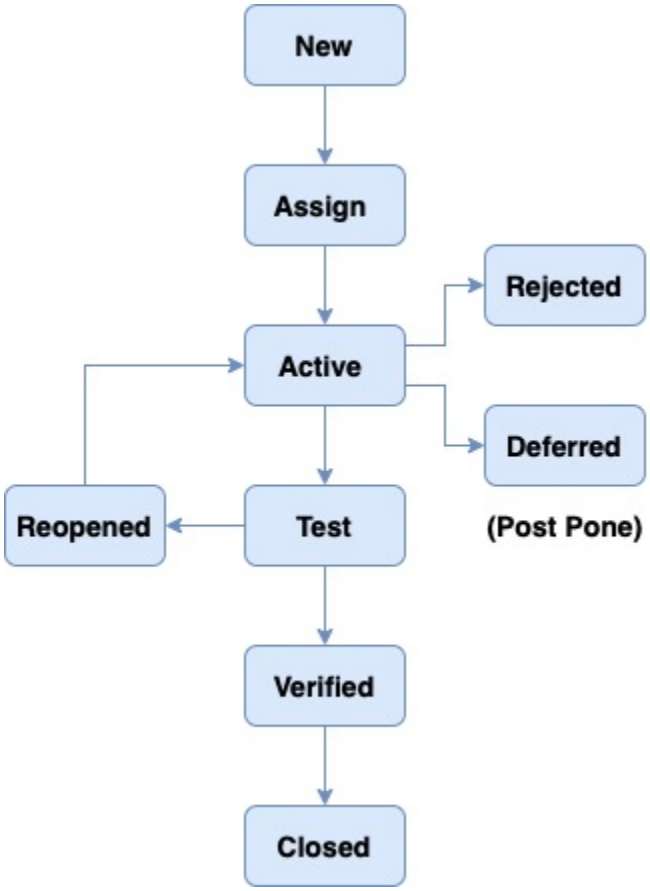


Fig 1.5 States of Bug

(1) New → Potential defect that is raised & yet to be validated
(2) Assigned → Assigned against a development team to address it
(3) Active → Bug is addressed by developer & investigation is under process. At this stage there are two possible outcomes; Deferred or Rejected
(4) Test → The defect is fixed & ready for testing.

(5) Verified → The defect that is retested & test has been verified by quality Assurance team
(6) Closed → Closed if the defect is duplicate or considered as not a defect anymore.
(7) Reopened → when the defect is not fixed.
(8) Rejected → for 3 reasons: duplicate defect, not a defect, non reproducible.
(9) Deferred → Post pore due to some reason.

1.13 Risk Identification in different stages

1. **Planning Stage Risk factors**

- Incomplete requirements:- It was found that users cannot describe more than 60% of the requirements at the beginning of the project; hence, requirements continue to change through the SDLC.

- Unrealistic Schedule:-The estimated time for the project as a whole may exceed the delivery date agreed upon previously.

- Unrealistic Budget:-The estimated budget depends on the required time, efforts and resources. The project may be out of funds early in SDLC and fail.

- Unclear Project Scope:-To manage the project scope (i.e. goals and requirements) is the most important task in planning.

- Insufficient Resources:-Sometimes available resources are not enough to complete the project. In other cases the system cannot be implemented using current technology where the project involves use of newer technology.

2. **Design Stage Risk factors**

- Incomplete Design:- The design document must be detailed enough to allow the programmers to work independently. If the design

document lacks these important details then the programmer may not work independently

- Large Design Document:- Although the design document must be detailed enough to ease the work of programmers, it should avoid extensive unimportant specification, which cause the design document to become large and thus non-readable

- Difficulties in verifying design to requirement:- In order to make sure that the design is a correct solution, the design must be verified against requirements to ensure that users' needs are reflected in the design. This is the case when the developer found it difficult to check whether the resulting design meets the users requirements

- Incorrect design:- When verifying the design, it might be found that the design does not match some, or even all, of the requirements. Worse, it might be different design other than the intended one

3. Testing Stage Risk factors

- Unqualified testing team:- Testing team experience has a significant influence on the testing process. Unqualified testers may destroy the whole process, since they might misuse the available tools, resources and techniques. Also, testing teams often lack skilled programmers, since testing is considered a trivial activity and can be performed by anybody else.

- Limited testing resources:-Time, budget, tools and other testing resources can hinder the testing process; either in their unavailability or in their misuse.

- Testing cannot cope with requirements change:- In most cases, users' needs continuously change. Tests are designed according to the initial description of requirements and cannot change to cope with requirement

- Wasting time in Building testing tool:- Testers sometimes distracted from testing in building testing tools, they waste time in Building testing tools instead of doing testing, which may will negatively affects the testing process

---

Question Bank

1. Define error, mistake, failure and bug.
2. Explain the term validation and verification.
3. Draw and explain V and V model for testing process.
4. Discuss the life cycle of bug.
5. Explain STLC with diagram.
6. Comment on :-
    - Need for verification and validation
    - Need for software testing
7. Goals of software testing
8. Difference between:- validation and verification
9. Explain verification activity in detail.
10. What are various types of bugs? Explain in detail.
11. List and explain :- States of a bug. (with diagram)

Module 2 || Testing Techniques

Index

2.1 Dynamic Testing

The name itself suggests that it is dynamic in nature, which means altering. This is the kind of testing that we do with altering values and or conditions by executing the code.

This includes testing the request in real-time by giving inputs and grouping the results or the output values of performance.

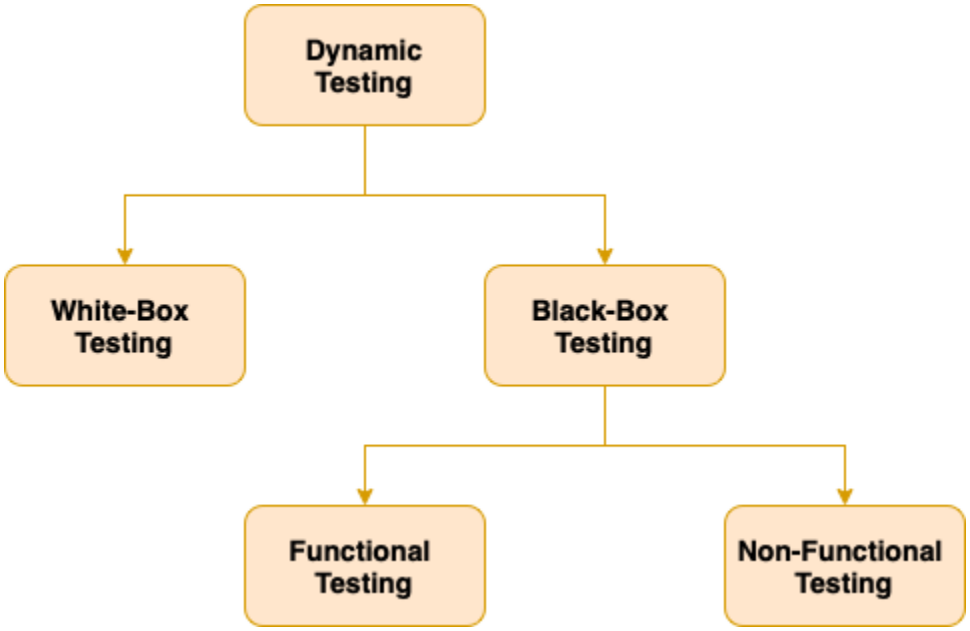


Fig 2.1 Dynamic Testing

2.2 White Box Testing

White Box Testing is performed by testers and developers.

White box testing is a software evaluating method used to examine the internal structure, design code and inner workings of software.

Developers use this testing method to verify the flow of inputs and outputs through the application, improving usability and design and strengthening security.

2.3 Black Box Testing

Black Box Testing is performed by end-users, designers, testers .

It is used to test a system against external factors responsible for software failure.

This testing approach focuses on the input that goes into the software and the output that is produced. The testing team does not cover the inside details such as code server, logic and development method.

2.4 Difference between Black-Box and White-box testing

Black Box Testing	White Box Testing
● It is a testing method which is used to test the software without the knowledge of the internal structure of program or application	● It is a testing method in which internal structure is known to the tester
● It is also known as data driven box testing	● It is also known as structural testing, clear box testing, code based testing, or glass box testing
● Ideal for high-level testing like system, testing, receipt testing … etc	● Ideal for low-level testing like unit testing and integration testing
● Programming knowledge is not needed in this testing	● programming knowledge is required in this testing
● The main objective is to check functionality of system under test	● The main objective is to check the excellence of the code
● Black box testing techniques→ ➢ Boundary value Technique	● White box testing techniques→ ➢ Data Flow testing ➢ Control Flow testing

➢ Decision table Technique ➢ State transition Technique ➢ All pair testing Technique ➢ Error guessing Technique ➢ Use-case Technique	➢ Branch testing ➢ Statement testing

2.5.1 Boundary Value Testing (Formula Table for Numericals)

Techniques	Formula for Number of Test-Cases	Required Inputs
1. Boundary Value Robust Worst Case (BVC)	$4n + 1$	min min^{+1} max max^{-1} $nominal$
2. Boundary value Robust (BVR)	$6n + 1$	min min^{+1} min^{-1} max max^{+1} max^{-1} $nominal$
3. Worst Case	5^n	min min^{+1} max max^{-1} $nominal$

4. Robust Worst Case	7^n	min min^{+1} min^{-1} max max^{+1} max^{-1} $nominal$

(Note:- 'n' is number of inputs**)**

Numericals:-

Question 1: An integer within range 1 to 100; to determine whether it's prime or not-prime. Design test cases using Boundary Value Robust Worst Case method.

Solution:-

Input value → n=1

Range → [1,100]

Total number of test cases → 4n+1=4+1=5

$min = 1$
$min^{+1} = 2$
$max = 100$
$max^{-1} = 99$
$nominal = 55$
Test case table:-

Test Case Id	Input Value (number)	Expected Output
1	1	Not Prime
2	2	Prime

3	100	Prime
4	99	Not Prime
5	55	Prime

Question 2: A program computes a^b where a lies in range 1 to 10 and b lies in range 1 to 5.Design test cases using Boundary Value Robust Worst Case method.

Solution:-

Input value → n=2

Range of a → [1,10]

Range of b → [1,5]

Total number of test cases → 4n+1=8+1=9

	a	b
min	1	1
min^{+1}	2	2
max	10	5
max^{-1}	9	4
$nominal$	5	3

Test case table:-

Test Case Id	Input Value a (number)	Input Value b (number)	Expected Output

1	5	1	5
2	5	2	25
3	5	5	3125
4	5	4	625
5	5	3	125
6	9	1	9
7	9	2	81
8	9	5	59049
9	9	4	6561

Question 3: A program reads 3-numbers within range 1 to 50 and prints the largest number. Design test cases using Boundary Value Robust method.

Solution:-

Input value → n=3

Let a,b,c be the three input numbers,

Range of a,b,c → [1,50]

Total number of test cases → 6n+1

=18+1

=19

	a	b	c
min	1	1	1

min^{+1}	2	2	2
min^{-1}	0	0	0
max	50	50	50
max^{+1}	51	51	51
max^{-1}	49	49	49
$nominal$	24	25	26

Test case table:-

Test Case Id	Input a (number)	Input b (number)	Input c (number)	Expected Output
1	1	2	0	b Is largest
2	2	0	50	c Is largest
3	0	50	51	c Is largest
4	50	51	49	b Is largest
5	51	49	26	a Is largest
6	49	25	1	a Is largest
7	24	1	2	a Is largest
8	1	2	50	c Is largest
9	2	0	51	c Is largest
10	0	50	49	b Is largest
11	50	51	26	b Is largest
12	51	49	1	a Is largest
13	49	25	2	a Is largest

14	24	1	51	c Is largest
15	1	2	49	c Is largest
16	2	0	26	c Is largest
17	0	50	1	b Is largest
18	50	51	2	b Is largest
19	51	49	51	a,c is largest

Question 4: The program determines the date is valid or invalid. Date is entered in the form of dd/mm/yyyy. Design test cases using Boundary Value Robust method. [Year Range → 1900 to 2025]

Solution:-

Input value → n=3

Total number of test cases → 6n+1=18+1=19

	dd	mm	yyyy
min	1	1	1900
min^{+1}	2	2	1901
min^{-1}	0	0	1899
max	31	12	2025
max^{+1}	32	13	2026
max^{-1}	30	11	2024

nominal	15	6	2000

Test case table:-

Test Case Id	dd	mm	yyyy	Expected Output
1	1	1	1900	valid
2	2	2	1901	valid
3	0	0	1899	In valid
4	31	12	2025	valid
5	32	13	2026	In valid
6	30	11	2024	valid
7	15	6	2000	valid
8	1	1	1901	valid
9	2	2	1899	valid
10	0	0	2025	In valid
11	31	12	2026	In valid
12	32	13	2024	In valid
13	30	11	2000	valid
14	15	6	1899	In valid
15	1	1	2025	valid
16	2	2	2026	In valid
17	0	0	2024	In valid
18	31	12	2000	valid

19	32	13	2025	In valid

Question 5: An integer within range 1 to 100; to determine whether it's prime or not-prime. Design test cases using :-

1. **Worst Case**
2. **Robust Worst Case**

Solution:-

Input value → n=1

Range → [1,100]

Worst Case	$5^n = 5$	$min = 1$ $min^{+1} = 2$ $max = 100$ $max^{-1} = 99$ $nominal = 55$
Robust Worst Case	$7^n = 7$	$min = 1$ $min^{+1} = 2$ $min^{-1} = 0$ $max = 100$ $max^{+1} = 101$ $max^{-1} = 99$ $nominal = 55$

Test case table using Worst Case:-

Test Case Id	Input Value (number)	Expected Output
1	1	Not Prime
2	2	Prime
3	100	Prime
4	99	Not Prime
5	55	Prime

Test case table using Robust Worst Case:-

Test Case Id	Input Value (number)	Expected Output
1	1	Not Prime
2	2	Prime
3	0	Not Prime
4	100	Prime
5	101	Prime
6	99	Not Prime
7	55	Prime

2.5.2 Equivalence Class Partitioning

Equivalence class partitioning allows to divide sets of test conditions into a partition which should be measured the same.

It handles two things very efficiently:-

1.Completeness → without executing all the test cases, it tries to touch the completeness of testing domain that gives one to one test case design represented by 1-sample test case from give class

2.Non-redundancy → when the test cases are executed having inputs from the same class, then there's a redundancy in executing test cases. Time and resources are wasted In executing these redundant test cases as they explore the same type of bug.

Guidelines :-

1.Two types of classes can always be identified; valid input class and invalid input class

2.Assign unique identification number to each equivalent class

3.Write a new test case covering as many as of uncovered valid equivalence class as possible until oil equivalence class have been identified

4.Write a test case that covers one and only one of the uncovered equivalence classes until all equivalence classes are covered by the test cases.

Numericals:-

Question 1: A program reads 3-numbers within range[1-to-50] and prints the largest number. Design test cases using equivalence class testing.

Solution:-

Let a,b,c be the three input numbers,

Range of a,b,c → [1,50]

Classes →

$I_1 \quad < a: 1 \leq a \leq 50 >$

$I_2 \quad < b: 1 \leq b \leq 50 >$

$I_3 \quad < c: 1 \leq c \leq 50 >$

$I_4 \quad < a: a < 1 >$

$I_5 \quad < a: a > 50 >$

$I_6 \quad < b: b < 1 >$

$I_7 \quad < b: b > 50 >$

$I_8 \quad < c: c < 1 >$

$I_9 \quad < c: c > 50 >$

Test Case Id	Input a (number)	Input b (number)	Input c (number)	Expected Output	Classes Covered
1	13	25	36	36	I_1, I_2, I_3
2	0	13	45	45	I_4
3	51	34	17	51	I_5
4	29	0	47	47	I_6
5	39	53	49	53	I_7
6	23	27	0	27	I_8

7	29	23	57	57	I_9

Question 2: Design set of equivalence classes based on possibilities of 3 integers. Output prints the largest of three numbers entered.

Solution:-
Let a,b,c be the three integers,

Classes →
$I_1 \quad < a: a > b, a > c >$
$I_2 \quad < b: b > a, b > c >$
$I_3 \quad < c: c > a, c > b >$
$I_4 \quad < a: a = b, a \neq c >$
$I_5 \quad < b: b = c, b \neq a >$
$I_6 \quad < c: c = a, c \neq b >$
$I_7 \quad < a = b = c >$

Test Case Id	Input a (integers)	Input b (integers)	Input c (integers)	Expected Output	Classes Covered
1	25	13	13	a Is largest	I_1 ,I_5
2	25	40	25	b Is largest	I_2, I_6
3	24	24	37	c Is largest	I_3, I_4
4	25	25	25	All are same	I_7

Question 3: A program determines the next date in the calendar. Its input is entered in the form of dd/mm/yyyy. [Year within the range

1900-to-2025]. Its output message would be the next date or an error message 'invalid date'. Design test cases using equivalence class testing.

Solution:-

Classes →

$I_1 \quad < mm: 1 \leq mm \leq 12 >$

$I_4 \quad < mm: mm < 1 >$

$I_5 \quad < mm: mm > 12 >$

$I_2 \quad < dd: 1 \leq dd \leq 31 >$

$I_6 \quad < dd: dd < 1 >$

$I_7 \quad < dd: dd > 31 >$

$I_3 \quad < yyyy: 1900 \leq yyyy \leq 2025 >$

$I_8 \quad < yyyy: yyyy < 1900 >$

$I_9 \quad < yyyy: yyyy > 2025 >$

Test Case Id	dd	mm	yyyy	Expected Output	Classes Covered
1	4	10	2001	5/10/2001	I_1, I_4, I_7
2	0	10	2001	invalid	I_5, I_1, I_7
3	4	13	2001	invalid	I_3, I_4, I_7
4	32	10	2001	invalid	I_6, I_1, I_7
5	4	0	2001	invalid	I_2, I_4, I_7
6	4	10	1899	5/10/1899	I_8, I_4, I_1
7	4	10	2026	5/10/2026	I_9, I_4, I_1

Question 4: A program takes an angle as input. Determine in which quadrant that angle lies in. Design test cases using equivalence class testing.

Solution:-

Let a be angle,

classes→

I_1 $< a: 0 \leq a \leq 90 >$ →1st quadrant

I_2 $< a: 91 \leq a \leq 180 >$→2nd quadrant

I_3 $< a: 181 \leq a \leq 270 >$→3rd quadrant

I_4 $< a: 271 \leq a \leq 360 >$→4th quadrant

I_5 $< a: a > 360 >$

I_6 $< a: a < 0 >$

Test Case Id	Expected Input (Angle)	Expected Output (quadrant)	Classes Covered
1	45	1st quadrant	I_1
2	120	2nd quadrant	I_2
3	240	3rd quadrant	I_3
4	300	4th quadrant	I_4
5	377	invalid	I_5
6	-93	invalid	I_6

Question 5: A program takes input 3 angles and determines the type of triangle. Design test cases using equivalence class testing.

Solution:-

Let a,b,c be angles of triangle, such that a+b+c=180

classes→

$I_1 \quad < a: a = 90 >$

$I_2 \quad < b: b = 90 >$

$I_3 \quad < c: c = 90 >$

$I_4 \quad < a: a < 90 >$

$I_5 \quad < a: a > 90 >$

$I_6 \quad < b: b < 90 >$

$I_7 \quad < b: b > 90 >$

$I_8 \quad < c: c < 90 >$

$I_9 \quad < c: c > 90 >$

Test Case Id	Angle a	Angle b	Angle c	Expected Output	Classes Covered
1	60	60	60	Acute angle triangle	I_4, I_6, I_8
2	90	45	45	Right angle triangle	I_1, I_6, I_8
3	30	90	60	Right angle triangle	I_2, I_4, I_8
4	50	40	90	Right angle triangle	I_3, I_4, I_6
5	120	30	30	Obtuse angle triangle	I_5, I_6, I_8
6	20	120	40	Obtuse angle triangle	I_7, I_4, I_8
7	120	50	120	Obtuse angle triangle	I_9, I_4, I_6

Question 6: A mobile phone service provider uses a program that computes the monthly bill of customer as follows:-

Min Rs 300/- → (Up-to free 120 calls) + (1Rs per call for next 70 calls) + (0.80Rs per call for next 50 calls) + (0.40 Rs per call for next calls beyond 240).

Design test cases using equivalence class testing.

Solution:-

Let c: calls

classes→

I_1 $< c: 1 \leq c \leq 120 >$ → 0Rs

I_2 $< c: 121 \leq c \leq 190 >$ → 1Rs

I_3 $< c: 191 \leq c \leq 240 >$ → 0.8Rs

I_5 $< c: c > 240 >$ → 0.4Rs

I_6 $< c: c < 1 >$ → invalid

Test Case Id	Expected Input (calls made)	Expected Output (cost per call)	Classes Covered
1	100	0Rs	I_1
2	145	1Rs	I_2
3	239	0.8Rs	I_3
4	241	0.4Rs	I_4
5	0	invalid	I_5

2.5.3 Decision Table Based Testing

Boundary value analysis and equivalence partition does not consider the combination of input conditions.

These two-methods consider each input separately.

There may be some critical-behaviors to be tested were some combinations of inputs are considered

So Decision Table based Testing supports complex combinations of inputs conditions, with logical expressions depending on input conditions and respective actions decision table test cases will be designed.

- Condition→ List out all the conditions for which complex combinations are made.
- Action→ List out resulting actions which will be performed if a combination of input condition is Satisfied
- Rule→ It's combination of input-conditions which itself becomes a Test case

Numericals:-

Question 1: A program calculates total salary of an employee with the combination that if working hours are less than or equal to 8 hours,then given normal salary.8 hours over working in a normal working day is calculated at a rate of 1.25 of the salary.On holidays/sundays the hours are calculated at rate of 2x of salary.Design test case using Decision table based testing method.

Solution:-

		Rules		
		R1	**R2**	**R3**
Condition Stub	C1: working hours>8	I	F	T
	C2:working on sunday/Holiday	T	F	F
Action Stub	A1: Normal salary		X	
	A2: 1.25x of Normal salary			X
	A3: 2x of Normal salary	X		

Test Case Id	**Working Hours**	**Day**	**Expected output**
1	7	Normal day	Normal salary
2	9	Normal day	1.25x of Normal salary
3	1	sunday/Holiday	2x of Normal salary

Question 2: A wholesaler has 8 commodities to sell and 3 types of customers and a discount is given as per the following procedure.

1.For DGS&D orders 10% discount is given irrespective of the value of the order.
2.For orders more than 50,000 agents get a discount of 15% and retailers get
a discount of 10%.
3.For order of 20,000 or more and upto 50,000 agents gets 12% discount and retailers get 8% discount.
4.For orders less than 20,000 agents get 8% discount and retailers get 5% discount.

Above rules do not apply for furniture items where in a flat range of 10% discount is admissible to all types of customer in irrespective of values of the order.Design test cases for system using Decision-table testing technique.

Solution:-

		Rules							
		R1	**R2**	**R3**	**R4**	**R5**	**R6**	**R7**	**R8**
Condition Stub	DGS/D	T	F	F	F	F	F	F	F
	Agent	F	T	F	T	F	T	F	I
	Retailer	F	F	T	F	T	F	T	I
	Order>50k	I	T	T	F	F	F	F	I
	Order between 20k to 50k	I	F	F	T	T	F	F	I
	order<20k	I	F	F	F	F	T	T	I
	furniture	F	F	F	F	F	F	F	T
Action Stub	Discount of 5%							x	
	Discount of 8%					x	x		
	Discount of 10%	x		x					x
	Discount of 12%				x				
	Discount of 15%		x						

Test Case Id	Type Of Customer	Product Furniture?	Order Value	Expected Output
1	DGS/D	NO	51K	Discount of 10%
2	Agent	NO	52K	Discount of 15%
3	Retailer	NO	53K	Discount of 10%
4	Agent	NO	24K	Discount of 12%
5	Retailer	NO	25K	Discount of 8%
6	Agent	NO	15K	Discount of 8%
7	Retailer	NO	11K	Discount of 5%
8	Agent	YES	2K	Discount of 10%

Question 3: Passengers who travel more than 50,000 Km per calendar year and in addition pay cash for tickets or have been traveling regularly for more than 8 years are to receive a free round trip ticket around India. Passengers who travel less than 50,000 Km per calendar year and have been availing railway services regularly for more than 8 years who get a free round ticket around India.Design test cases for this system using Decision table testing technique.

Solution:-

Rules

		R1	R2	R3	R4	R5
Condition Stub	Distance>50,000 km	T	I	T	F	F
	Cash payment	T	I	F	T	F
	Travel years>8	F	T	F	F	F
Action Stub	Free ticket	X	X			
	No Free ticket			X	X	X

Test Case Id	Distance	Cash Payment	Travel Years	Expected Output
1	51000	yes	5	Free ticket
2	22000	yes	11	Free ticket
3	52000	no	6	No Free ticket
4	21000	no	4	No Free ticket
5	6000	no	2	No Free ticket

2.6 Integration Testing

Integration testing is defined as a type of testing where software components are combined logically and tested as a group.

A typical software project contains multiple software models and code by dis-similar programmers .

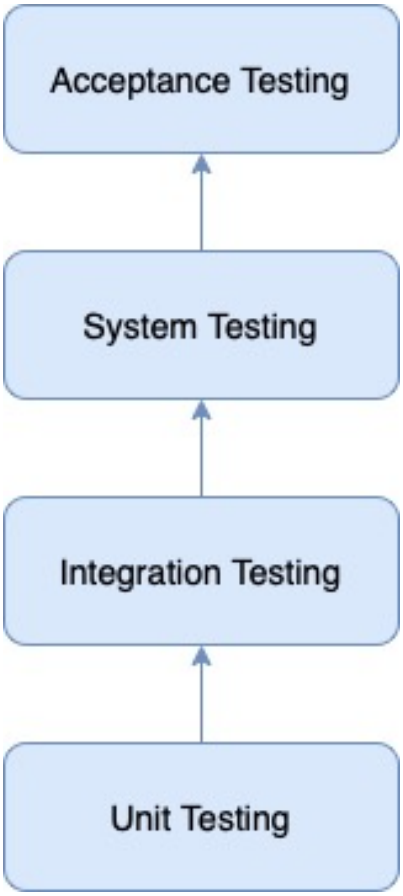


Fig 2.2 Integration Testing

Unit testing

It tests small models or a piece of code of an application or a product.

It's a quickly written and run test typically performed by software developers.

Can be performed anytime and has very low maintenance.

2.7 Design Stub And Driver Module

Stub and driver are considered as elements that play a vital role in testing; equivalent to modules that could be replaced, If models are in the developing stage, missing or not developed yet.

- Stub → mainly for top down integration testing

- Driver → yeah, mainly for bottom of integration testing

Top down approach

(Big to small mechanism)

- Also known as incremental approach/ combination testing.
- The higher level mechanisms are first completed after which low level models are tested.
- Once it's done, they are combined
- It uses stubs to simulate the stub components

Bottom up approach

(Small to big mechanism)

- Lower level model Sir tested first and then higher level models are expressed.
- It uses test drivers to initiate and pass required data to the stub module.
- It’s highly complex and data intensive .

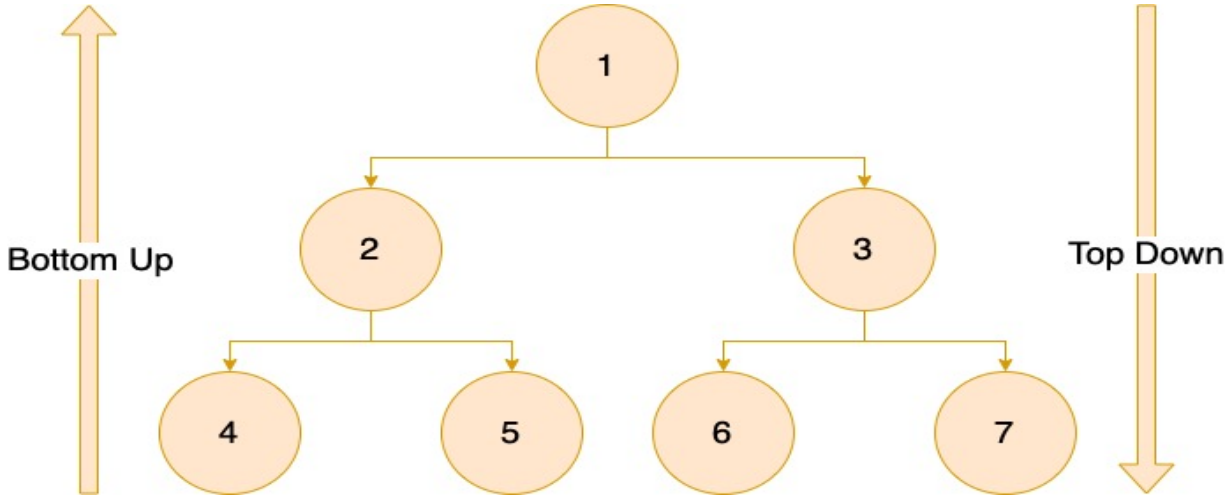
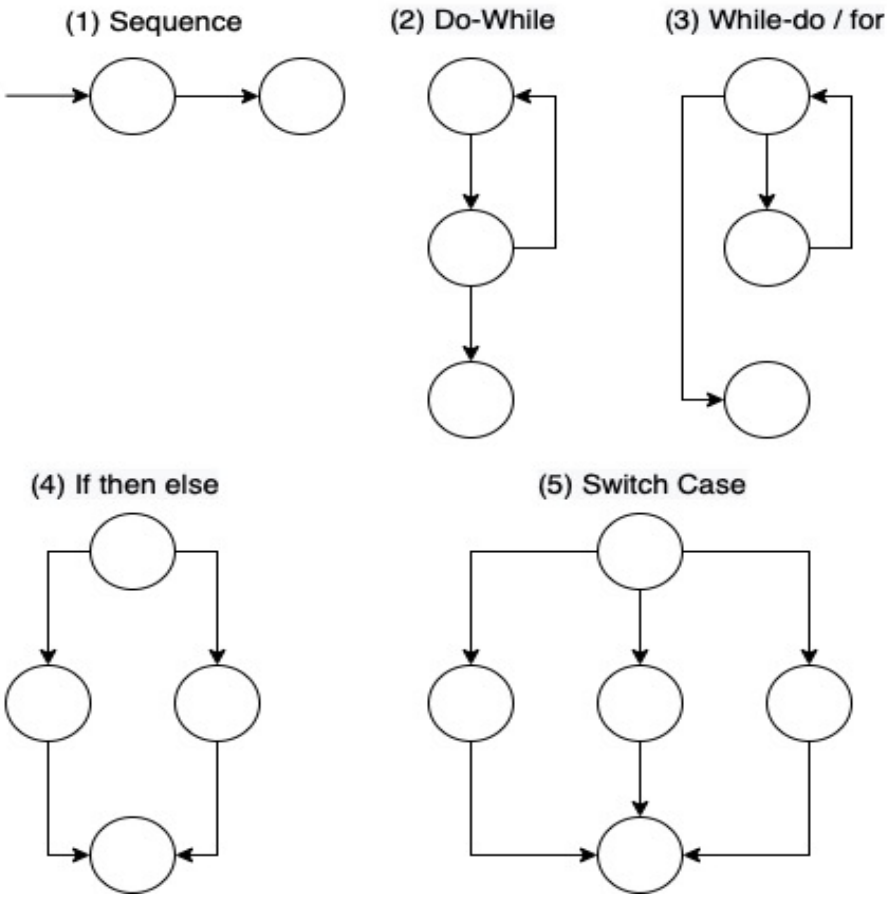


Fig 2.3 Top Down/Bottom Up approach

2.7 White Box Numericals

1. Node	Its representation of one or more procedural statements. Denoted by circle
2. Edge or Link	It represents the flow of control in a program. Denoted by arrow on the edge [an edge must terminate on node]
3. Decision Node	A node with more than one arrow leaving is called division node.
4. Junction Node	A node with more than one arrow entering is called a junction node.
5. Regions	Area bounded by edge and nodes are called regions [when counting regions always remember to add 1 considering the region outside the graph]

Flow graph notations for different programming constructs:-



Cyclomatic Complexity V(G) —(formula)→

V(G)=e-n+2P

V(G)=d+P

V(G)= Number Of Independent Paths

Where;

e→ edges

n→ nodes

P→ procedures

d→ decision nodes

Question 1:

Code →

```
    main()
    {
    int number, index;
1.  printf (“enter a number”);
2.  scanf(“%d”,&number);
3.  index=2;
4.  While (index <=number-1)
5.    {
6.      If (number% index ==0)
7.      {
8.      printf(“not a prime number”);
9.      break;
10.         }
11.        Index++;
12.       }
13.    if (index==number)
14.    printf(“prime number”);
15.    } //end main
```

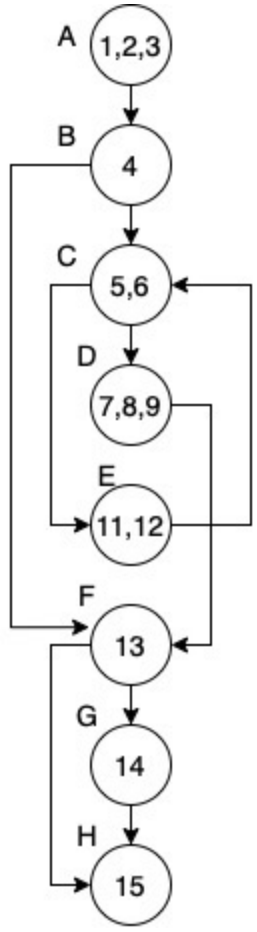
Find →

- **Draw DD-Graph for the program**
- **Calculate cyclomatic complexity**
- **List all Independent paths**
- **Design test case for all Independent paths**

Solution:-

p=1
n=8
e=10
d=3

V(G)=e-n+2p=10-8+2(1)=4
V(G)=d+p=3+1=4



V(G)= Number Of Independent Paths =4 →

A→ B→ F→ H

A→ B→ C→ D→ F→ H

A→ B→ F→ G→H

A→ B→ C→ E→ B→ F→ G→ H

Test Case Id	Input (Number)	Expected Output	Path Covered
1	**1**	**Not prime**	A→ B→ F→ H
2	**2**	**Prime**	A→ B→ F→ G→H
3	**3**	**prime**	A→ B→ C→ D→ F→ H
4	**4**	**Not prime**	A→ B→ C→ E→ B→ F→ G→ H

Question 2:

Code →

```
    main()
    {
    char string [80];
    int index;
1.  printf("Enter the string for checking its characters");
2.  scanf("%s", string);
3.  for(index = 0; string[index] != '\0'; ++index) {
4.  if(string[index] >= ‘0’ && string[ index] <='9')
5.  printf("%c is a digit", string[index]);
6.  else if (string[index] >= 'A' && string[index] <'Z') || (string[index]
    >= 'a' && string[index] <'z')
7.  printf("%c is an alphabet", string[index]);
8.  else
9.  printf("% is a special character", string[ index]);
10.   }
11.   }
```

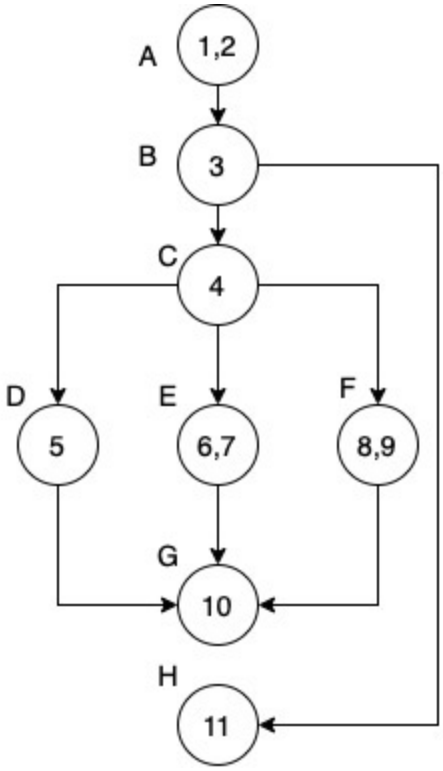
Find →

- **Draw DD-Graph for the program**
- **Calculate cyclomatic complexity**
- **List all Independent paths**
- **Design test case for all Independent paths**

Solution:-

p=1
n=8
e=10
d=3

V(G)=e-n+2p=10-8+2(1)=4
V(G)=d+p=3+1=4
V(G)=V(G)= Number Of Independent Paths =4 →
A→B→C→D→G→B→H
A→B→C→E→G→B→H
A→B→C→F→G→B→H
A→B→H



Test Case Id	Input	Expected Output	Path Covered
1	1	Digit	A→B→C→D→G→B→H
2	A	Alphabet	A→B→C→E→G→B→H
3	@	Special Character	A→B→C→F→G→B→H

Question 3:

Code →

```
main()
{
char chr;
1. printf ("Enter the special character\n");
2. scanf ("%c", &chr);
3. if (chr != 48) && (chr != 49) && (chr != 50) && (chr != 51) && (chr ! =
   52) && (chr !=53) && (chr != 54) && (chr != 55) && (chr != 56) && (chr
   != 57)
```

```
4. {
5. switch(chr)
6. {
7. Case ‘*’: printf("It is a special character");
8. break;
9. Case '#': printf("It is a special character");
10.  break;
11.  Case '@': printf("It is a special character");
12.  break;
13.  Case '!': printf("It is a special character");
14.  break;
15.  Case '%': printf("It is a special character");
16.  break;
17.  default : printf("You have not entered a special character");
18.  break;
19.  }// end of switch
20.  } / end of If
21.  else
22.  printf("You have not entered a character");
23.  }// end of main
```

Find →

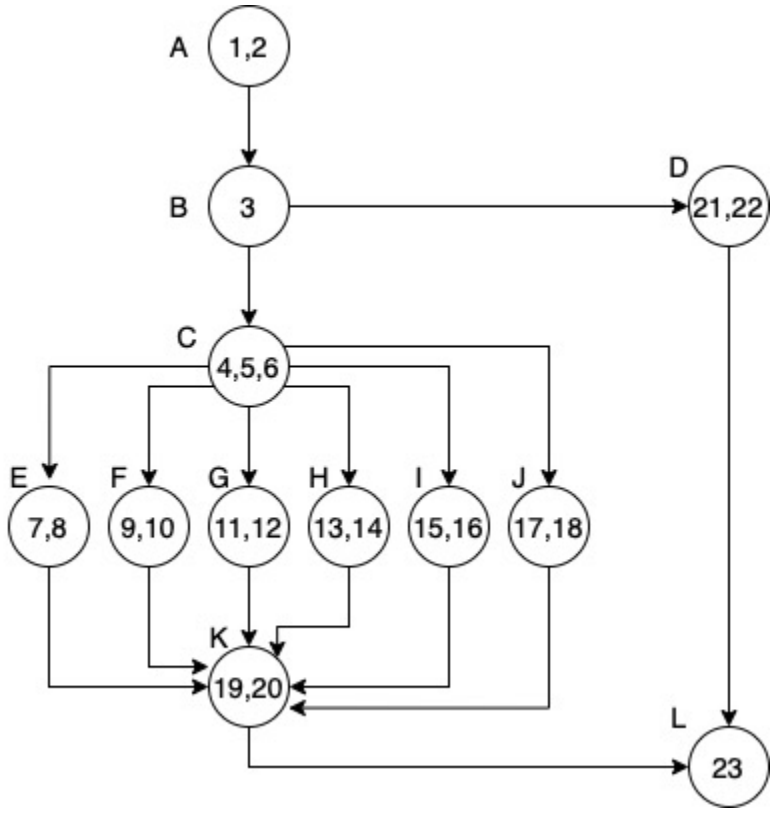
- **Draw DD-Graph for the program**
- **Calculate cyclomatic complexity**
- **List all Independent paths**
- **Design test case for all Independent paths**

Solution:-

p=1
n=12
e=17
d=6

V(G)=e-n+2p=7
V(G)=d+p=6+1=7
V(G)=V(G)= Number Of Independent Paths =7
A→B→D→L
A→B→C→E→K→L
A→B→C→F→K→L

A→B→C→G→K→L
A→B→C→H→K→L
A→B→C→I→K→L
A→B→C→J→K→L



Test Case Id	Input	Expected Output	Path Covered
1	{	You have not entered a character	A→B→D→L
2	*	It is a special character	A→B→C→E→K→L
3	#	It is a special character	A→B→C→F→K→L
4	@	It is a special character	A→B→C→G→K→L
5	!	It is a special character	A→B→C→H→K→L
6	%	It is a special character	A→B→C→I→K→L
7	$	You have not entered a special character	A→B→C→J→K→L

Question 4:

Code →

```
1   main()
    {
    int num, small;
    int i,j,sizelist,list [10],pos, temp;
2   clrscr();
    printf("\nEnter the size of list :\n ");
    scanf("%d" ,&sizelist);
    for(i=0;1<sizelist;i++)
    {
    printf("\nEnter the number");
3   scanf ("%d",&list[i]);
    }
4   for(i=0;i<sizelist;i++)
    {
5   small=list[i];
    pos=i;
    for(j=i+1;j‹sizelist;i++)
6   {
7   if(small>list[j])
    {
    small=list[j];
8   pos=j
    }
9   }
    temp=list[i];
10  list[i]=list[pos];
    list[pos]=temp;
11  }
12  printf("\nList of the numbers in ascending order : ");
13  (for I=0; i<sizelist; i++)
14  printf (“\n %d”, list[i]);
    getch();
15  }
```

Find →

- **Draw DD-Graph for the program**
- **Calculate cyclomatic complexity**
- **List all Independent paths**

Solution:-

p=1
n=15
e=19
d=5

V(G)=e-n+2p=6

V(G)=d+p=5+1=6

V(G)=V(G)= Number Of Independent Paths =6 →

1→2→3→2→5→6→7→8→9→6→10→11→4→12→13→14→13→15

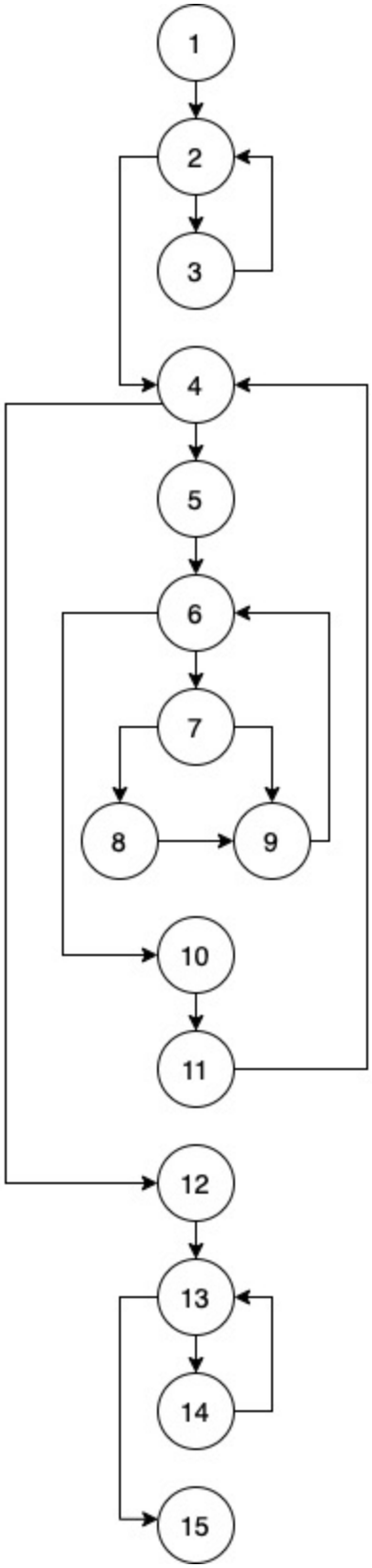
1→2→3→2→5→6→7→9→6→10→11→4→12→13→14→13→15

1→2→3→2→4→5→6→10→11→4→12→13→14→13→15

1→2→3→2→4→12→13→14→13→15

1→2→3→2→4→12→13→15

1→2→4→12→13→15


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Question 5:

Code →

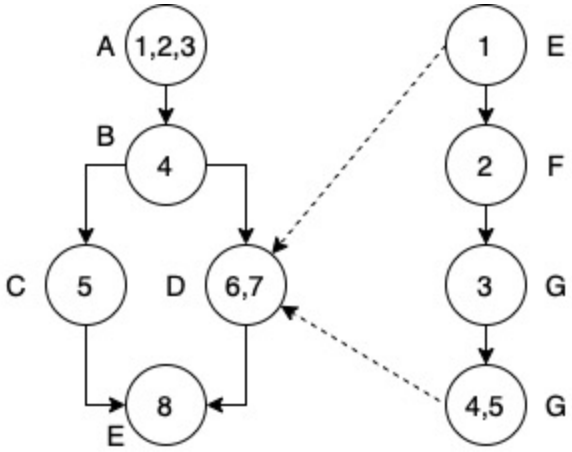
```
    int fact (int);
    main()
    {
    int number;
1. clrscr();
2. printf("Enter the number whose factorial is to be found out");
3. scanf("%d", &number);
4. if(number <0)
5. printf("Factorial cannot be defined for this number);
6. else
7. printf("Factorial is %d", fact (number));
8. }
    int fact(int number)
    {
    int index;
1. int product =1;
2. for (index=1, index<=number, index++)
3. product = product * index;
4. return(product);
5. }
```

Find →

- **Draw DD-Graph for the program**
- **Calculate cyclomatic complexity**

Solution:-

p=2
n=9
e=9
d=2
V(G)=e-n+2p= 9-9+2(2)=4
V(G)=d+p=2+2=4



Question Bank

1. What is White Box Testing?
2. What is Black Box Testing?
3. List out black box and white box testing techniques
4. Difference between:- black box and white box testing.
5. What is integration testing? What do you mean by UNIT Testing?
6. What is the importance of designing Stub and Driver module in any project? Explain with suitable Example.
7. Explain Top down and Bottom up approach with example in Testing.
8. A certain university is admitting students in a following course subject to following conditions:
   (1)

Subject	Marks
Java	>=70
C++	>=60
OOAD	>+60

   AND
   (2)Total in all 3 subjects>=220 or total in Java and C++ >=150.

   Now,if the total marks of an eligible candidate is more than 240 he/she eligible for scholarship course or it=f note then for normal course.The program reads input(marks) in subject and generate 3

types of output which are Not eligible,eligible for scholarship,eligible for normal course.Design test cases using Decision table based testing technique.

(Black-Box and White-Box testing Numericals)

Module 3 || Managing The Test Process

Index

3.1 Test Management

It's the process of managing the test activities in order to ensure high-quality and high-end testing of software.

By investing in a test-organization a company has access to a group of specialists who have the responsibility and motivation to:-

1. Maintain Testing Policy Statement
2. Plan the testing efforts
3. Monitor and track testing so as to be on time and within budget
4. Provide management with product and process quality information
5. Design and execute tests with no duplication of efforts
6. Automate testing
7. Participate in reviews to ensure quality; are meet
8.

3.2 Test Management Phases

Planning and Execution are two main parts of Test-Management process

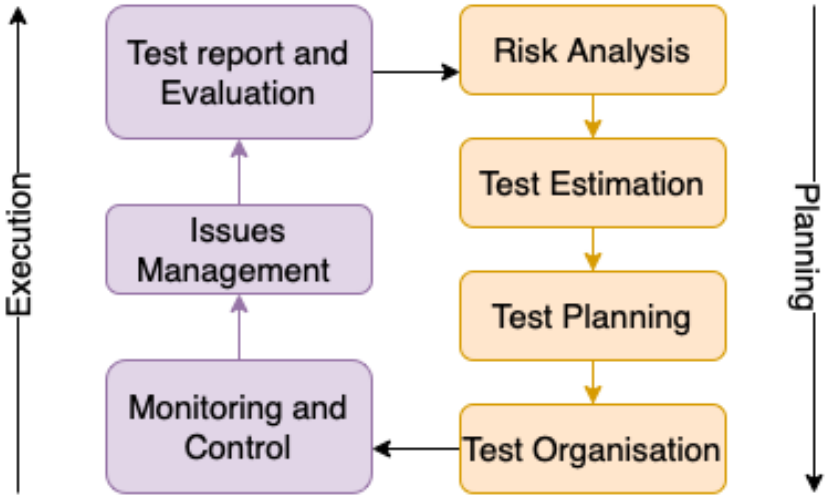


Fig 3.1 Test management phases

1. Planning Risk Analysis and Solution

All projects contain some amount of risk, And early risk detection & identification of its solution will help test-managers to avoid potential losses.

2. Test Estimation

Accurate test estimation leads to better planning, execution and monitoring of tasks under test managers attention.

3. Test Planning

Test planning gives a detailed testing information regarding the upcoming testing efforts including:- Test strategy, Test objective, exit/suspension criteria, resource-planning, test-deliverables

4. Test Organization in Software Testing

Defines who is responsible for which activities in the testing process.

Now you have a plan, but how will you stick to the plan and execute it?

→ the answer to this question is the Organization phase.

5.Execution

What will you do when your project runs out of time or resources ?

You need to control and monitor test activities to bring it back on schedule.

Monitoring	Control
● Define project goal or performance standard. ● Observe the performance and compare it with expected performance. ● Record and report generation	● Taking actions to correct the deviation from the plan. ●

6. Issue Management

In the life-cycle of any project, there will always be unexpected problems.

Risk to be avoided while testing:-

- Missing the dead lines
- Exceed the project budget
- Lose the customer trust

When issues arise you have to be ready to deal with them, or they can potentially affect the project's outcome.

3.3 6-key elements of test management software

1. Attractive UI and easy to use design
2. Allow for multiple projects and user permission
3. Traceability
4. Facilitate scheduling and organization
5. Monitoring and Metrics
6. Flexibility

1. Attractive UI and easy to use design

Simple is Beautiful.

2. Allow for multiple projects and user permission

There are range of individuals :- Project owner, project management professionals, product manager, tester & coder, … etc.

Some stakeholders will need read-only access for report and analysis, while others may need full admin access to everything in the project, depending on their status.

3. Traceability

Helps with useful data that you can refer-back to later.

4. Facilitate scheduling and organization

One of the most important tasks in software test management is figuring out how long the process will take, and you can't do that without effective scheduling, planning and organization.

5. Monitoring and Metrics

Important part of testing ensures that the project-testing is running properly on time, as it should and on budget.

6. Flexibility

Every project is different and should be treated as such.

3.4 Structure of Testing Group

It's important for a software organization to have an independent testing group.

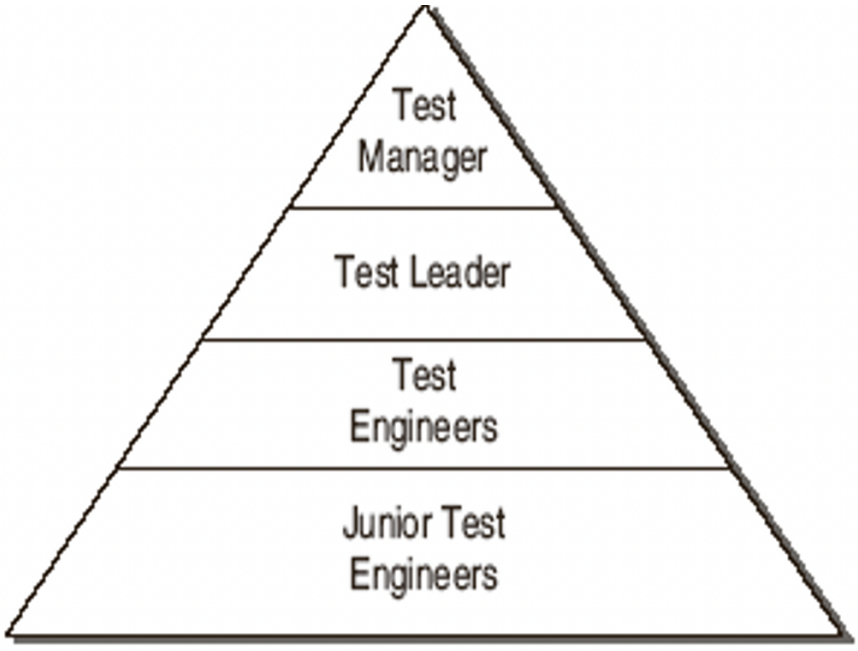


Fig 3.2 Structure of Testing Group

1. Test Manager

The test manager is usually responsible for test policy making, customer interaction, test planning, test documentation, controlling and monitoring of tests, training, test tool acquisition, participation in inspections and walkthroughs, reviewing test work, the test repository, and staffing issues such as hiring, firing, and evaluation of the test team members.

2. Test Leader

Assists the test manager and works with a team of test engineers on individual projects.
Responsible for duties such as test planning, staff supervision, and status reporting.
The test lead also participates in test design, test execution and reporting, technical reviews, customer interaction, and tool training.

3. Test Engineers

The test engineers design, develop, and execute tests, develop test harnesses, and set up test laboratories and environments.

4. Junior Test Engineers

Usually new hires. They gain experience by participating in test design, test execution, and test harness development.
They may also be asked to review user manuals and user help facilities defect and maintain the test and defect repositories.

3.5 Test Plan Hierarchy

- Test plans can be organized in several ways depending on the organizational testing policy.

- At the top of the plan hierarchy is a master plan, which gives an overview of all verification and validation activities for the project, as well as details related to other quality issues such as audits, standards, and configuration control. Below the master test plan, there is individual planning for each activity in verification and validation, as shown in figure.

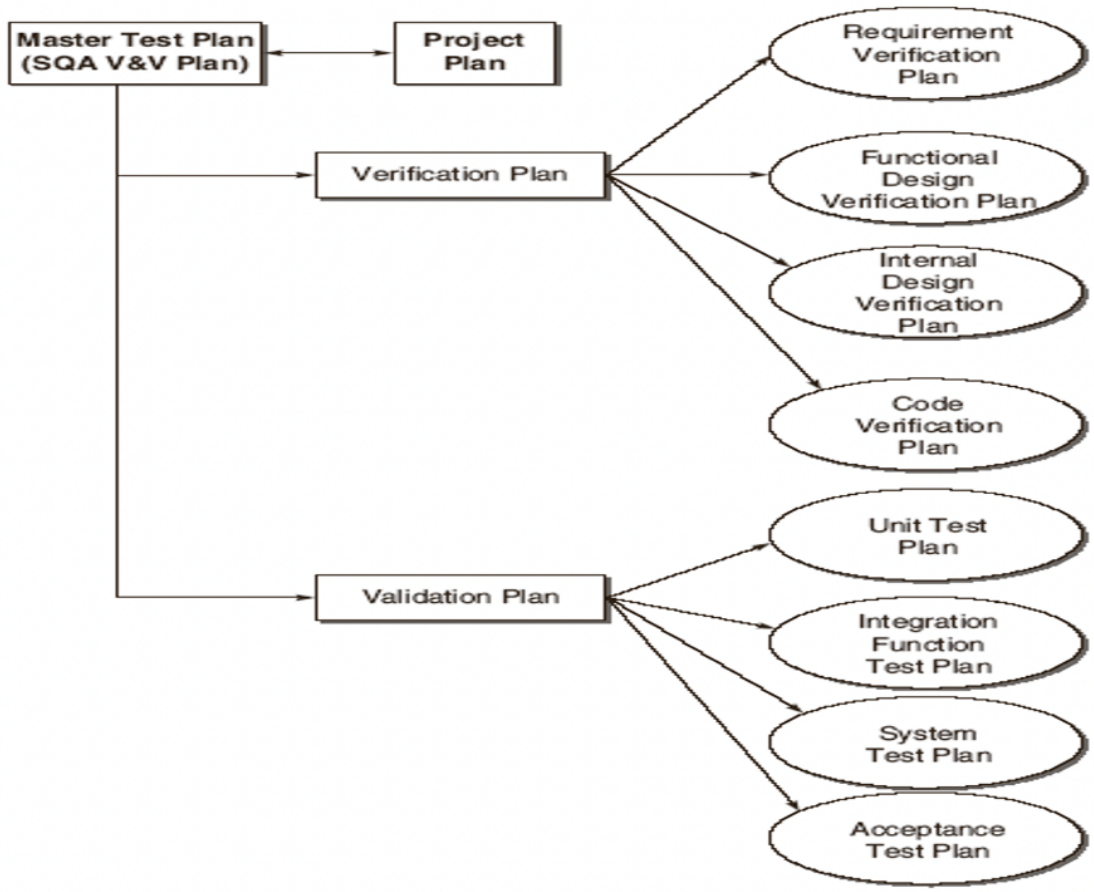


Fig 3.3 Test Plan Hierarchy

3.6 Detailed Test Design And Test Specification

Detailed test designing for each validation activity maps the requirements or features to the actual test cases to be executed. One way to map the features to their Test cases is to analyze the following.

- ➢ Requirement traceability
- ➢ Design traceability
- ➢ Code traceability

The analysis can be maintained in the form of Traceability Matrix

Requirement/Feature	Functional Design	Internal Design/Code	Test cases
R1	F1, F4,F5	abc.cpp, abc.h	T5, T8,T12,T14

Fig 3.4 Traceability Matrix

3.6.1 Test Design Specification

A test design specification should have the following components according to IEEE recommendation:

1. Identifier A unique identifier is assigned to each test design specification with a reference to its associated test plan.
2. Features to be tested The features or requirements to be tested are listed with reference to the items mentioned in SRS/SDD(software-requirement-specification and software-design-document).
3. Approach refinements In the test plan, an approach to overall testing was discussed. Here, further details are added to the approach.
4. Test case identification The test cases to be executed for a particular feature/function are identified here, as shown in Traceability Matrix
5. Feature pass/fail criteria The criteria for determining whether the test for a feature has passed or failed, is described.

3.6.2 Test Cases Specifications

Since the test design specifications have recognized the test cases to be executed, there is a need to define the test cases with complete specifications.

A test case specification should have the following components according to IEEE recommendation :

1.Test case specification identifier A unique identifier is assigned to each test case specification with. reference to its associated test plan.
2.Purpose The purpose of designing and executing the test case should be mentioned here.
3.Test items needed List the references to related documents that describe the items and features, for example, SRS, SDD, and user manual…etc
4.Special environmental needs In this component, any special requirement in the form of hardware or software is recognized. Any requirement of the

tool may also be specified.
5.Special procedural requirements Describe any special condition or constraint to run the test case, if any.
6.InterCase dependencies There may be a situation that some test cases are dependent on each other. Therefore, previous test cases that are run prior to the current test case must be specified.
7.Input specifications This component specifies the actual inputs to be given while executing a test case.
8.Test procedure The step-wise procedure for executing the test case is described here.
9.Output specifications Whether a test case is successful or not is decided after comparing the output specifications with the actual outputs achieved.

3.7 Test Result Specification

There is a need to record and report the testing events during or after the test execution.

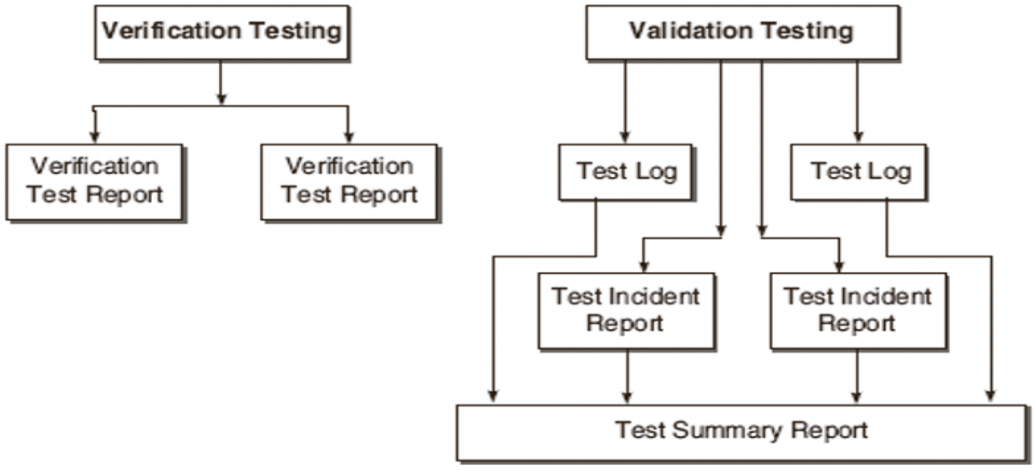


Fig 3.4 Test Result Specification

3.8 Test Incident Report

This is a form of bug report. It is the report about any unexpected event during testing, which needs further investigation to resolve the bug.

Example:-

- **Test Incident Report Identifier**
  TI2
- **Summary**
  Function 3.5 in SRS v 2.1. Test Case T2 and Test Log TL2.
- **Incident Description**
  It describes the following:
  (i) Date and time: 23/03/2009, 2.00pm
  (ii) Testing personnel names: ABC, XYZ
  (iii) Environment: Online environment with X database
  (iv) Testing inputs
  (v) Expected outputs
  (vi) Actual outputs
  (vii) Anomalies detected during the test
  (viii) Attempts to repeat the same test

Testing Inputs	Expected outputs	Actual outputs	Anomalies detected	Attempts to repeat the same test
4102645876	S12	S12	nil	–
21A2345672	Enter correct 10 digit PNR number	S14	Alphabet is being accepted in the input.	3
234	Enter correct 10 digit PNR number	Enter correct 10 digit PNR number	nil	–
asdgggggggg	Enter correct 10 digit PNR number	RAC12	Alphabet is being accepted in the input.	5

- **Impact**
  The severity value is 1 (Highest).

3.9 Test Summary Report

It is basically an evaluation report prepared when the testing is over. It is the summary of all the tests executed for a specific test design specification.

Example:-

- **Test Summary Report Identifier**
  TS2
- **Description**
  SRS v 2.1

S. No.	Functionality	Function ID in SRS	Test cases
1	Login the system	F3.4	T1
2	View reservation status	F3.5	T2
3	View train schedule	F3.6	T3
4	Reserve seat	F3.7	T4
5	Cancel seat	F3.8	T5
6	Exit the system	F3.9	T6

- **Variances**
  Nil
- **Comprehensive Statement**
  All the test cases were tested except F3.7, F3.8, F3.9 according to the test plan.
- **Summary of Results**

S. No.	Functionality	Function ID in SRS	Test cases	Status
1	Login the system	F3.4	T1	Pass
2	View reservation status	F3.5	T2	Bug Found. Could not be resolved.
3	View train schedule	F3.6	T3	Pass

- **Evaluation**
  The functions F3.4, F3.6 have been tested successfully. Function F3.5 couldn't be tested as the bug has been found. The bug is that the PNR number entry has also accepted alphabetical entries as wrong

---

Question Bank

1. What is Test Management?
2. Explain test management phases.

3. What are the 6 key elements of test management software?
4. Explain Structure of testing group.
5. What is Test-Plan-Hierarchy?
6. Explain: Detailed Test Design And Test Specification (in detail).
7. What are components of :- Test Design Specification?
8. What are components of :- Test Case Specification?
9. What is a Test-Incident-Report and Test-Summary-Report?

Module 4 || Test Automation

Index

4.1 Automation Testing/ Test Automation

4.2 Common Types of Testing in the field of Automation-Testing

4.3 JIRA

4.4 Bugzilla

4.5 Test Director

4.6 IBM Rational

4.7 Selenium IDE

4.8 Selenium's RC

4.9 Selenium Web Driver

4.10 Selenium Grid

4.1 Automation Testing/ Test Automation

Automation Testing or Test Automation is a software testing technique that performs using special automated testing tools to execute a test case suite.

On the other hand, Manual-testing is performed by a human sitting in front of computers carefully executing the test-steps.
Software-test-automation demands a considerable amount of investment of money & resources.
The gaol of Automation-testing is to reduce the number of test-cases to be run manually and to eliminate manual-testing altogether.
The test-automation can also enter data into the system under test and compare expected & actual results & generate detailed test reports.

4.2 Common Types of Testing in the field of Automation-Testing

1. Smoke Testing
2. Integration Testing
3. Regression Testing
4. Security Testing
5. Performance Testing
6. Acceptance Testing

- **Smoke Testing :**
  It's a type of functional testing that only covers the most crucial functional features of software-solution, to ensure it can be further tested without “Catching Fire”, hence the name smoke testing.
- **Integration Testing :**
  It takes all the individual pieces & functionalities of a software-solution & test then all together as a whole to guarantee smooth operation between all of them.
- **Regression Testing :**
  It runs a combination of functional & non-functional tests to check if the solution has “refused” after a given change.
- **Security Testing :**
  It covers functional & non-functional testing that screen the surface for any vulnerability.
- **Performance Tests :**
  These are non-functional tests that evaluate criteria are responsiveness & stability as the software handles loads and stress.

- **Acceptance Test :**
  Acceptance tests are functional tests that determine how acceptable software is to the end-users. This is the final test a solution must pass before it could be released.

4.3 JIRA

It is a software testing tool; Developed by the Australian's Company "Atlassian".
It's a bug tracking tool that reports all the issues related to your software or mobile Apps.
The word JIRA comes from a Japanese word, i.e., "Gojira" which means Godzilla.
It's one of the most widely used open-source testing tool used in manual-testing.
Aspects provided by JIRA :

1. Project (manage efficiently)
2. Issues (Track & manage defects/issues)
3. WordFlow (Processes the defects/issues life-cycle)
4. Search (Find with ease)
5. DasBoard (It's a display which you see when you login to JIRA).

You can create multiple dashboards for multiple-projects.

JIRA tools used because of the following reasons:

1. Plan, Track and Work faster.
2. It's a bug-tracking tool used to track, organize & prioritize bugs, newly added features, and improvements for certain software releases.
3. Projects are sub-divided into issues & issues can be of multiple types such as bugs, new-features & document tasks.
4. It manages projects as well as technical documentation.

4.4 Bugzilla

Bugzilla is a bug-tracking tool that helps to track issues related to their product.

Bugzilla tool is written in perl language and uses MySQL database.
It's a bug-tracking tool, but can also be used as a testing-management tool because it can be linked with other test cases management tools such as Quality-Center, Testling, etc.

Features Of Bugzilla :

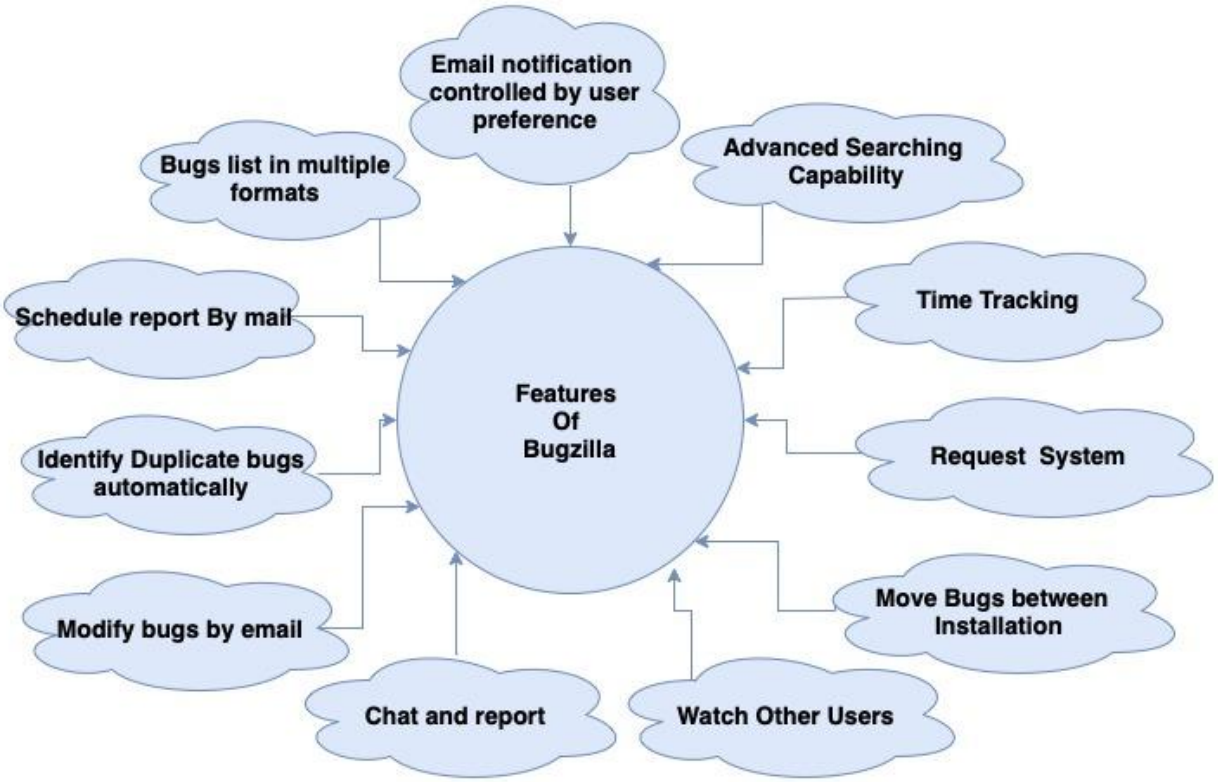


Fig 4.1 Features of BugZilla

4.5 Test Director

It’s a Global Test Management tool, the industry's 1st Global Test Management Solution.

It helps organizations deploy high-quality applications more quickly & efficiently.
Is has four modules :

1. Requirements
2. Test Plan
3. Test Lab
4. Defects

These models are seamlessly integrated allowing for a smooth information flow between various testing stages.
The complete Web-enabled Test Director supports high levels of communication & collaboration among distributed testing teams, driving a more efficient & effective global application testing process.

4.6 IBM Rational

IBM Rational Developer for i provides an integrated development environment (IDE) to create, maintain and modernize.
It integrates developer tools such as search, edit, build and analysis, refactoring capabilities and debuggers with the widely used Eclipse Framework for faster, easier application development & modernization.
Rational Developer for i integrates with IBM Rational Team Concert for better application life-cycle planning & management.
IBM Rational Developer for z system is an integrated development tool that you can use to design, develop, deploy & test mainframe software applications, Web & composite applications.
IBM Rational Developer for z systems offers integration for a variety of Source-Code-Management (SCM) tools in addition to providing a framework for creating your own SCM integration.

4.7 Selenium IDE

Selenium IDE is an open-source automation-testing-tool under selenium suit
Unlike selenium Web-Driver & RC, it does not require any programming logic to write its rest-scripts, rather you can simply just record your interaction with the browser to create test cases.

Subsequently you can use the play-bck option to re-run the test case.
Selenium IDE is the simplest framework in the Selenium suite.
It is a browser plugin to record and playback the operations performed on the browser.
Selenium IDE plugins are available for chrome & firefox browsers.
It does not support the programming features.
Selenium is the language which is used to write the test scripts in Selenium IdE.
Selenium IDE allows software tester to export record scripts in many languages like HTML, Ruby, PHP, Python. C#...etc.
Pros :

- Simple and easy to install and use.
- Built-in test result reporting and help modules.
- Test results can be easily exported.
- No prior knowledge of programming needed.

Cons :

- Only available for firefox.
- Data-driver testing is not supported.
- Test-case-execution is slower as compared to Web Driver and RC.
- Not able to test dynamic web applications.

4.8 Selenium's RC

Selenium RC is the main feature in the selenium. A tester can use it to simulate user actions such as input data, submit-a-form, and click a button in a web browser.
Selenium RC was the 1st tool used on the selenium project. It was the core application written in Java as the programming language.
This tool receives commands from the browser via 'HTTP" request.

It consist of 2 components :

1. Selenium RC server.

2. SeleniumRC client.

RC server will communicate with HTTP/GET/POST requests.
RC client will include programming codes.
You will be able to write & test-automate your code in most programming languages such as Java, JS, PHP, Python, C#..etc.
Pros :

- Supports cross-browser testing.
- Supports Data-Driver testing.
- Execution speed or more as compared to IDE.
- Supports conditional operations & interactions.

Cons :

- Slower execution speed as compared to web driver.
- Browser interaction is less realistic.
- Programming knowledge required.

4.9 Selenium Web Driver

Web Driver is the new feature added in Selenium 2.
It aimed to drive an easy and helpful programming interface to resolve the limitations of Selenium RC programming API.
Different from RC, Web Driver uses the Browser-native support to interact with web-pages.
So different web-browsers have different web-driver libraries & different features too.
Example :

1. HTTP Unit Driver : runs on Linux, Windows & Mac because of its pure Java implementation (one of the fastest and reliable)
2. Firefox Driver : Easy to configure and use (does not need any extra configuration to use other than that of Firefox Browser.)
3. Chrome Driver : (on chrome browser)
4. Internet-Explorer-Driver : Only runs in window OS, slower than the chrome and firefox browser-driver.

Selenium Web Driver (Selenium 2) is the successor of Selenium RC & is by for most important component of Selenium suite.

Pros :

1. No separate component such as RC servers are needed.
2. Execution time is faster than RC.
3. Supports testing on different platforms such as Android, iOS, Windows, Mac and Linux.

Cons :

1. Image testing is not available.
2. Prior knowledge of programming required.

4.10 Selenium Grid

With the Selenium Grid feature, test-script can run on multiple machines at the same time, which reduces the total test script run time.

This helps to find the bugs more quickly because the test cases runs more quickly.

This is suitable for large applications with too many test scripts to run.

You can also choose to run test-scripts, on different web browsers & on different test machines.

Selenium Grid is the part of Selenium Vision 1 that combined with Selenium RC scale for large test suites and able to run tests on remote machines.

It allows to write test-scripts in different programming languages such as Java, C#, Python, Perl, etc.

Although it overcomes all limitations of Selenium RC, generating a detailed test report is not possible with Selenium-Web-Deriver yet.

Pros :

1. Offers tools needed to diagnose the failure and rebuild a similar environment for the new test execution.
2. Save Time.
3. Supports simultaneous execution on multiple browsers, machines as well as remotely if needed.

Cons :

1. Remote machines receive only the browser control commands.

2. Considerable efforts & time are must for the initial operation of parallel testing.

Question Bank

1. What is Automation testing? / What is Test Automation?
2. What are the common types of testing in the field of Automation testing?
3. What is JIRA ?
4. What is Bugzilla ?
5. What is Test-Director ?
6. What is IBM Rational ?
7. What is Selenium's IDE ? Explain in detail.
8. What is Selenium's RC ? Explain in detail.
9. What is Selenium's Web-Driver ? Explain in detail.
10. What is Selenium's Grid ? Explain in detail.

Module 5 || TESTING FOR SPECIALIZED ENVIRONMENT

Index

5.1 Agile Testing

5.2 Process involved in agile testing (Agile Testing Life Cycle)

5.3 Agile Testing Principles

5.4 Merits & Demerits of Agile Testing

5.1 Agile Testing

Agile is an interactive development methodology where both development & testing activities are concurrent.

Testing is not a separate phase, coding & testing are done interactively & incrementally resulting in quality of end product.

Continuous integration results in early defects removed and hence time efforts and cost saving.
So, Agile Testing is a software testing practice that follows the principles of Agile Software Development.
In Agile Testing all team members of the project are involved, with special expertise contributed by testers.
Testing is not a separate phase and is interwoven with all the development phases such as requirements, design, coding & test code generation.
Testing takes place simultaneously through the Development Life Cycle.
Furthermore, with testers participating in the entire development lifecycle in conjunction with cross-functional team members, the contribution of testers towards building the software as per the customer requirements with better code and design would become possible.
Agile testing covers all the levels of testing and all types of testing.

5.2 Process involved in agile testing (Agile Testing Life Cycle)

- Agile Testing Life Cycle includes the following 5 phases:

Fig 5.1 Agile Testing Life Cycle

1. **Impact Assessment :** Gather input from stakeholders & users, this will act as feedback for the next development cycle.
2. **Agile Test Planning :** All stakeholders come together to plan the schedule for testing processes, meeting frequency and deliverables.
3. **Daily Scrums :** This includes everyday standup morning meetings to catch up on the status of testing and set the goals for the whole day.
4. **Agility Review meeting :** Weekly review meetings with stakeholders meet to review and access the process against milestones.
5. **Approval Meeting for Development :** At this stage we review the features that have been developed/implemented are ready to go live or not.

5.3 Agile Testing Principles

The principle of Agile Testing are :

1. **Testing is Continuous :** Agile team tests continuously because it's the only way to ensure continuous progress of product.

2. **Continuous Feedback :** Agile Testing provides feedback on an ongoing basis and this is how your product meets the business needs.
3. **Test Performed by the Whole Team :** Traditionally in SDLC, only testors perform testing, but in agile testing the developers and business analysts also test the application.
4. **Simplified and Clear :** All the defects which are raised by the agile team are fixed within the same interaction & helps in keeping the code clear and simplified.
5. **Less Documentation :** Use of reusable checklist, team focuses on tests instead of the incidental details.
6. **Test Driver :** In Agile testing is performed at the same time of implementation, whereas in the traditional process, the testing is performed after implementation.

5.4 Merits & Demerits of Agile Testing

- **Merits :**
    1. It saves time and money.
    2. Agile testing reduces documentation.
    3. It is flexible and highly adaptable to changes.
    4. Provides a way to receive regular feedback from the end user.
    5. Better determination of issues through daily meetings.

- **Demerits :**
    1. Not useful for small development projects.
    2. Cost of Agile development/testing methodology is slightly more as compared to other development methodology.

Question Bank

1. What do you mean by agile testing? Explain it in detail.
2. Explain process involved in agile testing / Agile Testing Life Cycle.

3. List agile testing Principles.
4. Discuss merits and demerits of Agile testing

Module 6 || Quality Management

Index

6.1 Quality in Software Testing

6.2 ISO-9000:2000 Quality Factors

6.2 Mackall's Quality Factors

6.4 Views of Software Quality

6.1 Quality in Software Testing

Quality refers to confirmation of implicit and explicit requirements, expectations and standards.

Need of quality in software testing:-

1. Requirement Quality (correctness, completeness and consistency of requirements).
2. Design Quality (every element of design model assured by software team)
3. Code Quality
4. Quality Control effectiveness
5. It saves time and money: Software quality assurance ensures that the developers find bugs and errors at the early stages of software development. Therefore, they spend a lot less time and money fixing them.
6. Stable and competitive software product. Software architects specifically vet each block in the software development process against industry standards. Granular testing for different requirements like reliability, functionality, usability, portability, etc., helps ensure that their product is high-quality.
7. Protect your company's reputation. Businesses need to ensure that their product works as intended before releasing into the market. If the customers notice the product's errors before you do, it will significantly impact your brand image and reputation.
8. Ensures security: Software quality assurance helps organizations ensure that their application is efficient, secure, and trustworthy.
9. Customer satisfaction. Your software application has to fulfill all the needs to satisfy the customers. It has to work smoothly without any malfunctions. With software quality assurance processes in place, you can ensure that your product delivers everything that your audience expects.

6.2 ISO-9000:2000 Quality Factors

1) **Customer Focus:** Success of an organization is highly dependent on satisfying the customers. An organization must understand its customers and their needs on a continued basis. Understanding the customers helps in understanding and meeting their requirements. It is not enough to just meet customer requirements. Rather, organizations must make an effort to exceed customer expectations.

2) **Leadership:** Leaders set the direction their organization should take, and they must effectively communicate this to all the people involved in the process. All the people in an organization must have a coherent view of the organizational direction. Without a good understanding of the organizational direction, employees will find it difficult to know where they are heading.

3) **Involvement of People:** In general, organizations rely on people. People are informed of the organizational direction, and they are involved at all levels of decision making. People are given an opportunity to develop their strength and use their abilities.

4) **Process Approach:** There are several advantages to performing major tasks by using the concept of process. A process is a sequence of activities that transform inputs to outputs. Organizations can prepare a plan in the form of allocating resources and scheduling the activities by making the process defined, repeatable, and measurable

5) **System Approach to Management:** A system is an interacting set of processes. A whole organization can be viewed as a system of interacting processes. In the context of software development, we can identify a number of processes. For example, gathering customer requirements for a project is a distinct process involving specialized skills.

6) **Continual Improvement:** Continual improvement means that the processes involved in developing, say, software products are reviewed on a periodic basis to identify where and how further improvements in the processes can be affected.

7) **Factual Approach to Decision Making:** Decisions may be made based on facts, experience, and intuition. Facts can be gathered by using a sound measurement process. Identification and quantification of parameters are central to measurement.Once elements are quantified, it becomes easier to establish methods to measure those elements.

8) **Mutually Beneficial Supplier Relationships:** Organizations rarely make all the components they use in their products. It is a common practice for organizations to procure components and subsystems from third parties.An organization must carefully choose the suppliers and make them aware of the organization's needs and expectations.

6.2 Mackall's Quality Factors

1) **Correctness**

Extent to which a program satisfies its specifications and fulfills the user's mission objectives

2) **Reliability**

Extent to which a program can be expected to perform its intended function with required precision.

3) **Efficiency**

Amount of computing resources and code required by a program to perform a function

4) **Integrity**

Extent to which access to software or data by unauthorized persons can be controlled

5) **Usability**

Effort required to learn, operate, prepare input, and interpret output of a program

6) **Maintainability**

Effort required to locate and fix a defect in an operational program

7) **Testability**

Effort required to test a program to ensure that it performs its intended functions

8) **Flexibility**

Effort required to modify an operational program

9) **Portability**

Effort required to transfer a program from one hardware
and/or software environment to another

10) **Reusability**

Extent to which parts of a software system can be
reused in other applications

11) **Interoperability**

Effort required to couple one system with another.

6.4 Views of Software Quality

1) **Transcendental View:**

In the transcendental view quality is something that can be recognized through experience but is not defined in some tractable form. Quality is viewed to be something ideal, which is too complex to lend itself to be precisely defined.

2) **User View:**

The user view concerns the extent to which a product meets user needs and expectations. Quality is not just viewed in terms of what a product can deliver, but it is also influenced by the service provisions in the sales contract. In this view, a user is concerned with whether or not a product is fit for use. This view is highly personalized in nature. The idea of operational profile, discussed in Chapter 15, plays an important role in this view. Examples of subjective elements are usability, reliability,testability, and efficiency.

3) **Manufacturing View:**

The manufacturing view has its genesis in the manufacturing sectors, such as the automobile and electronics sectors. In this view, quality is seen as conforming to requirements. Any deviation from the stated requirements is seen as reducing the quality of the product. The concept of process plays a key role in the manufacturing view. Products are to be manufactured “right the first time” so that development cost and maintenance cost are reduced.

4) **Product View:**

The central hypothesis in the product view is this: If a product is manufactured with good internal properties, then it will have good external qualities. The product view is attractive because it gives rise to an opportunity to explore causal relationships between internal properties and external qualities of a product. In this view, the current quality level of a product indicates the presence or absence of measurable product properties.

5) **Value-Based View:**

The value-based view represents a merger of two independent concepts: excellence and worth. Quality is a measure of excellence, and value is a measure of worth. The central idea in the value-based view is how much a

customer is willing to pay for a certain level of quality. The reality is that quality is meaningless if a product does not make economic sense

---

Question Bank

1. Explain need of Quality in Software Testing
2. What do you mean by Quality in Software Testing?
3. State ISO-9000:2000 Quality Factors.
4. State Mackall's Quality Factors.
5. List out five views of Software Quality.
