Green = Most Imp
Yellow = 2nd Priority Importance
Blue = Last Option of Importance

Complete Unit-1

1. What is quality? Definitions of quality from different perspectives- customer based, MAnufacturing based, product based, value based. (Topic 1.3)

Ans -

What is Quality? (Is it a fact or perception?)

1. The term `Quality' means different things to different people at different times, different places and for different products.
2. To some users, a quality product may be one, which has no/less defects and works exactly as expected and matches with his/her concept of cost and delivery schedule along with services offered. Such a thought may be a definition of quality - **"Quality is fitness for use".**
3. Quality is defined as,**" Conformance to specifications"** because a very little change to the design of a product and you have to make the product as per the design which will best suit the user's other expectations like less cost, fast delivery and good service support.

<u>Definitions of Quality</u> (CMPV)

For achieving the quality of the product, one must define it in some measurable terms which can be used as a reference to find whether quality is really met or not. The definitions of quality from different perceptions are as follow: -

1) Customer- Based definition of Quality- **Quality product must have "Fitness for use".**

- Customer-Based Definition of Quality states Quality product must have 'Fitness for use' and must meet customer needs, expectations and help in achieving customer satisfaction and possibly customer delight.
- This definition is mainly derived by an approach to quality management through 'Quality is fitness for use'.

2) Manufacturing- Based definition of Quality- This approach gives the definition of **"Conformance to requirements".**

- This definition is mainly derived from engineering product manufacturing where it is not expected that the customer knows all requirements of the product .This approach gives the definition of **"Conformance to requirements".**

3) Product- Based definition of Quality- **The product must have something that other similar products do not have.**

- The product must have something that other similar products do not have which can help the customer satisfy his/her needs in a better way.
- These attributes must add value for the customer/user so that he/she can appreciate the product in comparison to competing products.
- This makes the product distinguishable from similar products in the market. Also, the customers must feel proud of owning it due to its inherent attributes and characteristics.

4) Value- Based definition of Quality- Many times it is claimed that **"People do not buy products, they buy benefits".**

- A product is the best combination of price and features or attributes expected by or required by the customers.
- The customer must get value for his investment by buying the product. The cost of the product has a direct relationship with the value that the customer finds in it. Many times it is claimed that “People do not buy products, they buy benefits”.

2. Different Quality View's- (Topic 1.6 / PDF page30)

a. Customer's view of quality
b. Supplier's view of quality
c. User's gap / Requirements Gap

Or

Define Users gap and producers gap and explain how these gaps can be close effectively .

Ans -

Quality View

1. Customer's view of quality: - Customer's view of quality of product interprets customer's requirements and expectations of getting a better product at defined schedule, cost and with adequate service along with required features and functionalities.

- Delivering Right Product: - The product received by customers must be useful to satisfy their needs and expectations.
- Satisfying Customer's Needs: - Needs may be part of processes for doing requirement analysis and selection of approaches using some techniques such as cost-benefit analysis, etc.
- Meeting Customer Expectations: - customer expectation can be categorized into two parts as expressed expectations and implied expectations.
- Treating Every Customer with Integrity, Courtesy, and Respect: - Customer and the requirements are very important for a developing organization as customer will be paying on the basis of value he finds in the product.

2.Supplier's View of Quality: Supplier is a development organization of software application development. Supplier expectations may range from **profitability, name in the market, repeat orders, customer satisfaction**, etc. These expectations may be fulfilled in the following ways: -

- Doing the Right Things: - Supplier is intended to do right things for the first time so that there is no waste, scrap, rework, etc. Changes in requirements are considered as problems.
- Doing it the Right Way: - A producer may have his own methods, standards, and processes to achieve the desired outputs.
- Doing it Right the First Time: - Doing right thing the first time may avoid frustrations, scrap, rework, etc. and improve profitability, reduce cost and improve customer satisfaction.
- Doing it on Time: - All resources for developing a new product are scarce(lacking in quantity or number) and time factor has a cost associated with it. The value of money changes as time changes. Thus, money received late is as good as less money received.**"Time is money, and late money is less money."**

Fig. 1.1 User's gap and Producer's gap

The above figure explains a gap between actual product, requirements for the product and customer expectations from the product.

User's Gap / requirement Gap: -

It is a gap between requirement specifications for the product and user expectation from it. Developers must convert user needs into product requirement specification and create the product exactly as per these specifications.

Closing User's Gap: -

Methods for closing these gaps may depend on the organization's way of thinking, resource availability, type of customer, customer's thought process, etc. some of these methods are: -

- Customer Survey
- Joint Application Development
- User Involvement in Application Development

1.6.4 PRODUCER'S GAP/DEVELOPMENT GAP

Producer's gap is a gap between product actually delivered and the requirement and design specifications developed for the product. The requirement specifications written by business analyst may not be understood in the same way by the development team. There are communication losses at each stage and business analyst and developers/testers may not be located next to each other to provide explanation for each and every requirement. More stages of communication may lead to more gaps and more distortion of requirements. The product so produced and requirement specifications used may differ significantly creating producer's gap.

Closing Producer's Gap Producer's gap represents a failure on part of development team to convert requirements into product. Producer's gap can be seen as the defects found in in-house testing. Producer's gap is due to process failure at producer's place and there must be process improvement plans to close this gap.

Process Definition Development and testing processes must be sufficiently mature to handle the transfer of information from one person or one stage to another during software development life cycle. There must be continuous 'Do' and 'Check' processes to build better products and get feedback about the process performance. Such product development has life cycle testing activities associated with development.

Work Product Review As the stages of software development life cycle progresses, one may have to keep a close watch on artifacts produced during each stage to find if any inconsistency has been introduced with respect to earlier phase. Generation of requirement traceability matrix is an important factor in this approach.

4. Financial aspects of quality - (Topic 1.7 PDF page 35)

a. Cost of manufacturing

b. Cost of Quality -

i. Cost of prevention - Green Money,

ii. Cost of Appraisal - Blue Money and

iii. Cost of failure - Red Money

Ans -

Financial Aspect of Quality

Earlier, people were of the opinion that more price of a product represents better quality as it involves more inspection, testing, sorting, etc. and ensures that only good products are supplied to the customer. Sales price was defined as: -

Sales Price = Cost of manufacturing + Cost of Quality + Profit

If any product enjoys higher profitability, more number of producers would enter into competition. Number of sellers may exceed the number of buyers. When the products produced match in all aspects, the cost would decide the quantity of sale. The competitor, who reduces the price, may get more volume of sale if all other things remain constant. Reducing the sales price reduces percentage profit. Thus in a competitive environment, the equation changes to:

Profit = Sales price - [Cost of manufacturing + Cost of Quality]

Cost of Manufacturing: - Cost of manufacturing is the cost required for developing the right product by right method at the first time. The money involved in resources like material, people, licenses, etc. forms a cost of manufacturing.The cost of manufacturing remains constant over the time span for the given project and given technology and it has a direct relationship with the efforts.

Cost of Quality: Cost of Quality represents the part of cost of production incurred in improving or maintaining quality of product. Some people keep the cost of manufacturing and the cost of quality as separate while others may include them under the cost of production.

- **Cost of prevention (Green money):** An organization may have defined processes, guidelines, standards of development, testing etc. It may define a program of providing training to all people involved in development and testing. This may represent a cost of prevention. This is also termed as "Green Money". Generally, it is believed that 1 part of cost of prevention can reduce 10 parts of cost of appraisal and 100 parts of cost of failures.
- **Cost of appraisal (Blue money): -** The cost incurred in first time reviews and testing is called as the cost of appraisal. This is termed as "Blue Money" as it can be recovered from the customer. Generally, it is believed that 1 part of the cost of appraisal can reduce 10 parts of the cost of failures.
- **Cost of failure (Red money): -** Cost of failure starts when there is any defect or violation detected at any stage of development including post-delivery efforts spent on defect fixing. Any extent of rework, retesting, sorting, scrapping, regression testing, late

payments, sales, under concession etc. represents the cost of failure. This is termed as "Red Money". This cost affects the profitability of the project/organization badly.

On the basis of quality focus, organizations may be placed in 2 categories viz. organizations which are less quality conscious (termed as 'q') and organizations which are more quality conscious (termed as 'Q'). The distribution of cost of quality for these two types is as below:-

5. Principles of TQM - Total Quality Management (Topic 1.11)page 38

Ans-

Some quality principles of TQM are:-

1) Develop Constancy(the quality of being faithful and dependable) of purpose of definition and deployment of various initiatives:

Management must create constancy of purpose for products and processes, allocating resources adequately to provide for long term as well as short term needs rather than concentrating on short term profitability

2) Adapting to new philosophy of managing people/stakeholders by building confidence and relationships: - Management must adapt to the new philosophies of doing work and getting the work done from its people and suppliers.

3) Declare freedom from mass inspection of incoming/produced output: It was believed that one must check everything to ensure that no defect goes to the customer.

4) Stop awarding of lowest price tag contracts to suppliers: - Organizations must perform measurements of quality of supply along with price and do the source selection on the basis of final cost paid by it in terms of procurement, rework, maintenance, operation etc.

5) Improve every process used for development and testing of products: - Improve every process of planning, production and service to the customer and other support processes constantly.

6) Institutionalize training across the organization for all people: - An organization must institute modern methods of training which may include on-the-job training, classroom training, self-study etc. for all people, including management, to make better use of their abilities.

7) Institutionalize leadership throughout organization at each level: - An organization must adopt and institute leadership at all levels with the aim of helping people to do their job in a better way.
8) Drive out fear of failure from employees: - An organization must encourage effective two-way communication and other means to drive out fear of failure from the minds of all employees.
9) Break down barriers between functions/departments:- Physical as well as psychological breaking down of barriers between departments and staff areas may create a force of cohesion.
10) Eliminate exhortations by numbers, goals, targets: - Eliminate use of slogans, posters and exhortations of the work force, demanding 'Zero defects' and new level of productivity, without providing methods and guidance about how to achieve it.
11) Eliminate arbitrary numerical targets which are not supported by processes: Eliminate work standards that prescribe quotas for the work force and numerical goals for managers to be achieved.
12) Permit pride of workmanship for employees: - Remove the barriers that take away the pride of workmanship for workers and management. People must feel proud of the work they are doing and know how they are contributing to organizational vision.
13) Encourage education of new skills and techniques: - Institute a rigorous program of education and training for people working in different areas and encourage self- improvement programs for everyone.

6. PDCA Cycle (Topic Topic 1.14) OR Write a short note on continual improvement cycle.

Ans -

Continual (continuous) improvement cycle is based on systematic sequence of Plan-Do-Check-Act activities representing a never ending cycle of improvements.

Stages of Continual (Continuous) improvement through PDCA are: -

1) Plan: - An organization must plan for improvements on the basis of its vision and mission definition. Planning includes answering all questions like who, when, where, why, what, how etc.(w5-h)

2) Do: - Plan is not everything but a roadmap. Actual execution of a plan can determine whether the results as expected are achieved or not. Do process need inputs like resources, hardware, software, training, etc. for execution of a plan.

3) Check: - An organization must compare actual outcome of Do stage with reference or expected results which are planned outcomes. It must be done periodically to assess whether the progress is in proper direction or not and whether the plan is right or not.

4) Act: - If any deviations (positive or negative) are observed in actual outcome with respect to planned results, the organization may need to decide actions to correct the situations.

7. Problematic areas of software development life cycle. (Topics 2.11)

8. Pillars of Quality Management System (topic - 2.16)

Generic view of quality management includes three tiers and three pillars.
Following are the tiers: -

1) 1st Tier (Quality Policy)

Quality policy sets the wish, intent and directions by the management about how activities will be conducted by the organization.

2) 2nd Tier (Quality Objective)

Quality objectives are the measurements established by the management to define progress and achievements in a numerical way.

3) 3rd Tier (Quality Manual)

Quality manual, also termed as a policy manual, is established and published by the management of the organization.

Following are the pillars: -

1) Quality processes/ Quality procedures/ Work Instructions

At the organization level, they're defined by functional experts, while at the project and function levels, experts in those specific areas determine them separately.
“Quality processes are like well-crafted blueprints—defined by experts at different levels—to ensure the final product stands strong and true.”

2) Guidelines and Standards

They are used by an organization’s project team for achieving quality goals for the products and the services delivered to customers.

3) Format and templates

They are used for tracking a project, function,and department information within an organization.

9. Quality culture in 'Q' and 'q' organizations. (page 56)

Organization Culture

'Q' organizations are more quality conscious organizations while 'q' organizations are less quality conscious organizations.

Quality culture is `Q'	Quality culture is not `q'
These organizations believe in listening to customers and determining their requirements	These organizations assume that they know customer requirements
These organizations concentrate on identifying cost of quality and focusing on it to reduce cost of failure which would reduce overall cost and price	These organizations overlook cost of poor quality, hidden factory effect. They believe in more testing to improve product quality.
Doing things right for the first time and every time is the motto of success	Doing things again and again to make them right is the way of working. Inspection, rework, scrap etc. are essential.
They concentrate on continuous / continual process improvement to eliminate waste and get better output	They work in a way of finding and fixing the problem as and when they are found. Onetime fix for each problem after it occurs.
These organizations believe in taking ownership of processes and defects at all levels	These organizations tries to assign responsibility of defects to someone else
They demonstrate leadership and commitment to quality and customer satisfaction	They believe in assigning responsibility for quality to others

Complete Unit-2

1. Approaches of testing- (topic 3.4)
    a. Big Bang approach of testing
    b. TQM as against Big Bang

Test methodologies /Approaches (topic- 3.33)
    a. Black Box
    b. White Box

Ans -
2. Testing during development life cycle.(3.6)
    a. Requirement testing
    b. Design testing
    c. Code testing

2. Testing during development life cycle.(3.6)
Ans -

a. Requirement testing
b. Design testing
c. Code testing

a. Requirement testing

Requirement testing involves mock running of future applications using the requirement statements to ensure that requirements meet their acceptance criteria. This type of testing is used to evaluate whether all requirements are covered in the requirement statement or not.

Characteristics of Requirement verification:

1. **Completeness** of requirement statements as per organization standards and formats is crucial.
2. **Clarity** about what is expected by the users at each step of working while using an application is essential.
3. **Measurability** of expected results, possibly in numerals, is important so that these results can be tested.
4. **Testability** of the scenarios defined in the requirement statement is a must. Some requirements, such as "the application must work 24 x 7 for 10 years," may not be directly testable, but efforts should be made to identify measurable aspects or components that can be tested.
5. **Traceability** of requirements further down the development life cycle must be ensured.

b. Design testing

Design verification focuses on reviewing the design, typically by subject-matter experts, utilizing standards, templates, and guidelines defined for creating these designs. Design verification ensures that designs meet their exit criteria. This includes:

Completeness of Design: Ensuring that the design covers all possible outcomes of processing and handles various controls as defined by requirements.

Clarity of Data Flow: Verifying the clarity of the flow of data within an application and between different applications supposed to work together in the production environment.

Testability of Design: Evaluating the software structure and structural testing to ensure that the design is testable.

Traceability with Requirements: Ensuring that the design covers all requirements and maintains traceability back to the original requirements.

c. Code testing

1. Code files, tables, stored procedures, etc., are written by developers as per guidelines, standards, and detailed design specifications.
2. In reality, developers do not implement requirements directly but implement detailed design as defined by the thesis.
3. Debugging and testing are done using the guidelines, and code review is conducted to ensure that the files are readable and maintainable in the future.
4. There are adequate comments available for ease of understanding, making it feasible in unit testing.
5. Traceability with requirements and designs is crucial.
6. Anything extra, as well as anything missing, can be considered as a defect.
7. The code should be testable in integration and system testing, optimized to ensure better working of the software. Reusability creates a lighter system.

3. Requirement traceability matrix (Advantage and disadvantage)(topic- 3.7)

a. Horizontal traceability ,
b. Bidirectional traceability
c. Vertical traceability

Ans -

REQUIREMENT TRACEABILITY MATRIX

- Some quality management models and standards prescribe complete traceability of a software application from requirements through designs and code files upto test scenario, test data, test cases and test results.

- Requirement traceability matrix is one way of doing the complete mapping for the software.
- One can expect a blueprint of an entire application using the requirement traceability matrix. Typical requirement traceability matrix is as shown below.

Table 3.2 Requirement traceability matrix

Require-ments	High-level Design	Low-level design	Code files/ Stored Proce-dures/TBLs	Test scenario	Test cases	Test results

Advantages :

- Entire software development can be tracked completely through the requirement traceability matrix.
- Any test case failure can be tracked through requirements, designs, coding, etc.
- Any changes in requirements can be affected through the entire work product upto test cases and vis-à-vis(in relation to) any test case failure can be traced back to requirements.
- The application becomes more maintainable as there is a clear and complete relationship mapped from requirements to test results.

Disadvantages :

- Number of requirements is huge. It is very difficult to create a requirement traceability matrix manually. For using some tools, one needs to invest money. Also, people may need to be trained for using tools.
- There may be one-to-many, many-to-one and many-to-many relationships between various elements of the traceability matrix, when we are trying to connect columns and rows of a traceability matrix, and maintaining these relationships need huge efforts.
- Developing teams may not understand the importance of requirement traceability matrix, if development follows a waterfall model, and during maintenance, it may be too late to create it. Incremental and iterative developments are the major challenges for maintaining traceability.
- A customer may not find value in it and may not pay for it.

<u>HORIZONTAL TRACEABILITY</u>

1. When an application can be traced from requirement through design and coding till test scenario and test cases upto test results, it is termed as horizontal traceability.
2. On failure of any test case, we must be able to find which requirements have not been met. Any design which does not have requirement, introduces an extra feature which may be considered as defect. Similarly, when any requirement is not traceable to design, that requirement is not implemented at all.
3. Same thing can happen in the relationship between design and coding, coding and test scenario, and also, test scenario and test case. When any entity can't be traced in forward direction horizontal traceability is lost.

BIDIRECTIONAL TRACEABILITY

1. One must be able to go from requirements, designs, coding, and testing to reach the test result. Reverse must also be possible, where one may start from the result and go to requirements. One must be able to go in any direction from any point in the traceability matrix. This is referred to as 'bidirectional traceability'.
2. CMMi model mandates bidirectional traceability for all products.

VERTICAL TRACEABILITY

1. Traceability may exist in individual columns as the requirements may have some interdependencies between them, or these may be child and parent relationships. For achieving a requirement, the other child requirement must be achieved.
2. One requirement may have several child requirements, while some child requirements may have several parent requirements. If these requirements are traced completely, it ensures vertical traceability.
3. Similarly design, coding, and testing may have a vertical traceability where there may be parent-child relationship and interdependence on different parts. Designs may have parent-child relationships, and coding may have 'called functions' and 'calling functions' traceability relationships.

4. Work bench (topic- 3.9)

a. Tester's workbench (topic- 3.9.1) with Diagram

Ans -

1. **Workbench Role:** Each stage in the software development lifecycle acts like a distinct workbench, receiving inputs from the previous stage and providing outputs to the next stage, akin to a conveyor belt system.

2. **Interrelated Activities:** The development and testing process consists of a series of interconnected activities where the output of one activity serves as the input to the next, with each activity representing a workbench.

3. **Workbench Procedures:** A workbench comprises defined procedures for performing tasks (e.g., requirements collection, design, coding, testing). Additionally, there are procedures to verify the outcomes of the work done.

Fig. 3.2 Typical workbench

TESTER'S WORKBENCH

Tester's workbench is made of testing processes, standards, guidelines and tools used for conducting tests, and checking whether the test processes applied are effective or not. For every workbench, there should be a definition of entry criteria, process of doing/checking the work, and exit criteria. For testers, everything that comes to their workbench should be clearly listed. These may be defined in a test plan.

Examples of Tester's Workbench Considering a typical system testing life cycle for a product/ project, the different workbenches for a tester may be defined as follows.

- Workbench for creating test strategy
- Workbench for creating a test plan
- Work bench for writing test scenario
- Workbench for writing test cases
- Workbench for test execution
- Workbench for defect management
- Workbench for retesting
- Workbench for regression testing

5. Principles and Challenges of S/W (software) testing (3.12)

Ans -

Principles

1. Define the expected output or result for each test case executed, to understand if expected and actual output matches or not.
2. Developers should not test their own programs. Development teams must not test their own products.
3. Inspect the results of each test completely and carefully. It would help in root cause analysis and can be used to find weak processes.
4. Include test cases for invalid or unexpected conditions which are feasible during production. Testers needs to protect the users from any unreasonable failure so that one can ensure that the system works properly.
5. Test the program to see if it does what it is not supposed to do as well as what it is supposed to do.
6. Avoid disposable test cases unless the program itself is disposable. Reusability of test cases is important for regression.
7. Don't create test plans thinking you won't find any errors. There must be a targeted number of defects for testing.
8. The probability of locating more errors in any one module is directly proportional to the number of errors already found in that module.
9. Initiate actions for correction, corrective action and preventive actions.

Challenges in testing

1. Requirements are not clear, complete, consistent, measurable and testable. These may create some problems in defining test cases.
2. Requirements are wrongly documented and interpreted by business analysts and system analysts. These knowledgeable people are supposed to gather requirements of customers by understanding their business workflow. But sometimes they are prejudiced based on their experience.
3. Code logic may be difficult to capture. Often testers are not able to understand the code due to lack of technical language knowledge. Sometimes they do not have access to files.
4• Error handling may be difficult to capture. There may be combinations of errors and various error messages and controls are required such as detective controls , corrective controls, suggestive controls and preventive controls.

Other challenges in testing

• Badly written code introduces many defects
• Bad architecture of software cannot implement good requirement statement
• Testing is considered as a negative activity
• Testers find themselves in Lose - Lose situation(no-win situation)

6. Salient features of good testing (3.13)

Ans -

Salient Features of Good Testing :

Defects indicate the quality of software under testing, development, and test processes used for making testing a life-cycle activity where testers participate from the proposal stage until the application is finally accepted by the customer/user.

Good software testing involves the testing of the following.

Capturing User Requirements:

The requirements defined by the users or customers, as well as some implied requirements (which are intended by the users but not expressed), represent the foundation on which software is built. Testers need to analyze and document intended requirements to write test scenarios and test cases.

Capturing User Needs:

User needs may differ from user requirements specified in software requirement specifications. User needs may include present and future requirements, process requirements (including the definition of deliverables), and implied requirements.

Design Objectives:

Design objectives state why a particular approach has been selected for building software. The selection process indicates the reasons and criteria framework used for development and testing. interface requirements, performance requirements, etc., should be defined in an approach document.

User Interfaces:

User interfaces are the ways in which the user interacts with the system. This includes the system's user interfaces, such as how users input information, how they interact with the system, and how they interpret the information generated by the system.

Internal Structures:

Internal structures are mainly influenced by software designs and guidelines, standards, and best practices in designing and development. Internal structures may be defined by development organizations to ensure reusability, testability, and maintainability.

7. Mutation Testing (3.19)

3.19 MUTATION TESTING

Mutation testing is used to check the capability of test program and test cases to find defects. Test cases are designed and executed to find defects. If test cases are not capable of finding defects, it is a loss for an organisation (as it requires time to write and execute test cases). This is also termed 'test case efficiency'.

Solved Example 3.2

A program is written, and set of test cases are designed and executed on the program. The test team may find out few defects. The original program is changed and some defects are added deliberately. This is called 'mutant of the first program' and process is termed 'mutation'. It is subjected to the same test case execution again. The test cases must be able to find the defects introduced deliberately in the mutant.

Suppose,

Defects deliberately introduced by development = X
Defects found by test cases in original program = Y
Defects found by test cases in mutant = Z

The ratio of $(Z - Y)/X$ will give the test case efficiency. Theoretically, it must be 100%. But it may not be exactly 100% due to the following reasons.

3.19.1 REASONS FOR DEVIATION OF TEST TEAM EFFICIENCY FROM 100% FOR TEST TEAM AS WELL AS MUTATION ANALYSIS

Though desirable, it is very difficult to get a test team with 100% efficiency of finding defects and test cases with 100% efficiency of finding defects. Some of the reasons for deviation are listed below.

- *Camouflage Effect* It may be possible that one defect may camouflage another defect, and the tester may not be able to see that defect, or test case may not be able to locate the hidden defect. It is called 'camouflage effect' or 'compensating defects' as two defects compensate each other. Thus, defect introduced by developer may not be seen by the tester while executing a test case.

- *Cascading Effect* It may be possible that due to existence of a certain defect, few more defects are introduced or seen by the tester. Though there is no problem in the modification, defects are seen due to cascading effect of one defect. Thus, defects not introduced by developer may be seen by tester while executing a test case.
- *Coverage Effect* It is understood that nobody can test 100%, and there may be few lines of code or few combinations which are not tested at all due to some reasons. If defect is introduced in such a part which is not executed by given set of test cases, then tester may not be able to find the defect.
- *Redundant Code* There may be parts of code, which may not get executed under any condition, as the conditions may be impossible to occur, or some other conditions may take precedence over it. If developer introduces a defect in such parts, testers will not be able to find the defect as that part of code will never get executed.

Complete Unit-3

NOTE : These Unit Is More Focused on practical Solving Skills — Be Ready , Stay Happy 😀
All the problems given in assignments and Journals

1. Boundary Value Testing(Input domain testing) for a function of 2 variables

Ans -

- Input domain testing (also called “boundary value testing”) is the best-known **specification-based testing** technique.

Definition

Boundary value analysis is a test case design technique to test boundary value between partitions (valid and invalid). A boundary value is an input or output value on the border of an equivalence partition, including minimum and maximum values at inside and outside boundaries.

- Consider a **function F**, of two **variables x1 and x2**. When the function F is implemented as a program, the input variables x1 and x2 will have some boundaries

$$a \le x1 \le b$$
$$c \le x2 \le d$$

- The intervals **[a, b]** and **[c, d]** are referred to as the **ranges of x1 and x2**
- The input space (domain) of our function F is shown below : -

Input domain of a function of two variables.

4 types of BVA

1. Normal Boundary Value
2. Robust Boundary Value
3. Worst-case Boundary value (more than one variable has an extreme value)
4. Robust worst-case boundary value

1. Normal Boundary Value

- Normal Boundary value testing focuses on testing extreme values of input variables because errors are more likely to happen at these points.
- The basic idea of BVA is to use input variable values at their

Min	→	Minimal
Min+	→	Just above Minimal
Nom	→	Average
Max-	→	Just below maximin
Max	→	Maximum

Boundary value analysis test cases for a function of two variables.

- The normal boundary value analysis test cases for our function F of two variables (illustrated in Figure 5.2) are
  {<x1nom, x2min>, <x1nom, x2min+>, <x1nom, x2nom>, <x1nom, x2max–>, <x1nom, x2max>, <x1min, x2nom>, <x1min+, x2nom>, <x1max–, x2nom>, <x1max, x2nom>}
  Total number of test cases = **4n+1** (n=no. Of input variables)
  Total test cases = 9

2. Robust Boundary Value

- Robust boundary value testing is a simple extension of normal boundary value testing: in addition to the five boundary value analysis values of a variable, we see what happens when the extrema are exceeded with a value slightly greater than the maximum (max+) and a value slightly less than the minimum (min–).
- The main advantage of robustness testing is that it forces attention on exception handling.

Robustness test cases for a function of two variables.

Total number of test cases = **6n + 1** (n = number of input variables)
Total test cases = 13

3. **Worst-case Boundary value (more than one variable has an extreme value)**

- Both forms of boundary value testing, operate on the single fault assumption and this operates on multi fault assumption
- understanding the outcomes when more than one variable has an extreme value. In electronic circuit analysis, this concept is known as "worst-case analysis," and we apply it here to generate worst-case test cases.
- For each variable, we initiate with a five-element set that contains the minimum, minimum plus a small increment, the nominal value, the maximum minus a small decrement, and the maximum value. Subsequently, we take the Cartesian product of these sets to generate comprehensive test cases.The outcome of the two-variable version of this process is depicted below

Worst-case test cases for a function of two variables.

- Total test cases = 5^n where n is the number of input variable

4. **Robust worst-case boundary value**

- In Robust worst-case boundary value we consider all the previous three types of BVA and The outcome of the two-variable version of this process is depicted below

Robust worst-case test cases for a function of two variables.

- Total test cases = 7^n where n is the number of input variable

2.Examples of BVA-given in manual testing and PPT. For example - triangle problem,
next date problem etc..

Ans -

Conditions :

D: 1<Day<31

M: 1<Month<12

Y: 1800 <Year <2048

Boundary Value Analysis:

No. of test Cases (n = no. of variables) = 4n+1 = 4*3 +1 =13

Test Cases:

Test case Id	Day	Month	Year	Expected Output
1	1	6	2000	2-6-2000
2	2	6	2000	3-6-2000
3	15	6	2000	16-6-2000
4	30	6	2000	1-7-2000
5	31	6	2000	Invalid Date
6	15	1	2000	16-1-2000
7	15	2	2000	16-2-2000
8	15	11	2000	16-11-2000
9	15	12	2000	16-12-2000
10	15	6	1800	16-6-1800
11	15	6	1801	16-6-1801
12	15	6	2047	16-6-2047
13	15	6	2048	16-6-2048

<u>Implement Triangle problem :</u>

Write test cases (boundary value and equivalence class) to check all the combinations The triangle problem accepts three integers (a, b and c)as its input, each of which are taken to be sides of a triangle. The values of these inputs are used to determine the type of the triangle (Equilateral, Isosceles, Scalene or not a triangle).

Conditions to be implemented –

C1. $1 \leq a \leq 100$
C2. $1 \leq b \leq 100$
C3. $1 \leq c \leq 100$.
C4 : $a < b + c$
C5 : $b < a + c$
C6 : $c < a + b$

The type of the triangle, provided the conditions are met, is determined as follows:
1. If all three sides are equal, the output is Equilateral.
2. If exactly one pair of sides is equal, the output is Isosceles.
3. If no pair of sides is equal, the output is Scalene.

BVA

Test Case ID	a	b	c	Expected Output
T1	1	50	50	Isosceles
T2	2	50	50	Isosceles
T3	99	50	50	Isosceles
T4	100	50	50	Not a Triangle
T5	50	50	50	Equilateral
T6	50	1	50	Isosceles
T7	50	2	50	Isosceles
T8	50	99	50	Isosceles
T9	50	100	50	Not a Triangle
T10	50	50	1	Isosceles
T11	50	50	2	Isosceles
T12	50	50	99	Isosceles
T13	50	50	100	Not a Triangle

3. Decision table concept and examples

(Little Theory of Decision Table and Few Examples)

Ans -

Stub	Rule 1	Rule 2	Rules 3,4	Rule 5	Rule 6	Rules 7,8
c1	T	T	T	F	F	F
c2	T	T	F	T	T	F
c3	T	F	-	T	F	-
a1	X	X		X		
a2	X				X	
a3		X		X		
a4			X			X

condition stubs	condition entries
action stubs	action entries

1. A decision table has four portions: the part to the left of the bold vertical line is the stub portion; to the right is the entry portion.The part above the bold horizontal line is the condition portion, and below is the action portion. Thus, we can refer to the condition stub, the condition entries, the action stub, and the action entries.
2. The 1st column in the entry portion is a rule. Rules indicate which actions, if any, are taken for the circumstances indicated in the condition portion of the rule.
3. Decision tables in which all the conditions are binary are called **Limited Entry Decision Tables (LEDTs)**. If conditions are allowed to have several values, the resulting tables are called **Extended Entry Decision Tables (EEDTs)**

Example - Transferring money online to an account which is already added and approved.

Here the **Conditions** to transfer money are
Condition 1 : ACCOUNT ALREADY APPROVED
Condition 2 : OTP (One Time Password) MATCHED
Condition 3 : SUFFICIENT MONEY IN THE ACCOUNT

And the **Actions** performed are
Action 1 : TRANSFER MONEY
Action 2 : SHOW A MESSAGE AS INSUFFICIENT AMOUNT
Action 3 : BLOCK THE TRANSACTION INCASE OF SUSPICIOUS TRANSACTION

DECISION TABLE

ID	CONDITIONS/ACTIONS	TEST CASE 1	TEST CASE 2	TEST CASE 3	TEST CASE 4	TEST CASE 5
Condition 1	Account Already Approved	T	T	T	T	F
Condition 2	OTP (One Time Password) Matched	T	T	F	F	X
Condition 3	Sufficient Money in the Account	T	F	T	F	X
Action 1	Transfer Money	Execute				
Action 2	Show a Message as 'Insufficient Amount'		Execute			
Action 3	Block The Transaction Incase of Suspicious Transaction			Execute	Execute	X

For login page :

Conditions	Rule 1	Rule 2	Rule 3	Rule 4
Username (T/F)	F	T	F	T
Password (T/F)	F	F	T	T
Output (E/H)	E	E	E	H

Legend

- T - Make sure your login and password are correct.
- F - Incorrect login or password
- E - An error message appears.
- H - The home screen appears.

4. Cyclomatic complexity, and flow graph

Ans -
Cyclomatic complexity **quantifies the number of linearly independent paths** through a program's source code. It's calculated based on the control flow graph of the program, where nodes represent individual statements or blocks of code, and edges represent the flow of control between them.

The formula to calculate cyclomatic complexity is:
M = E-N+2

Where,

M is the **cyclomatic complexity**.

E is the **number of edges** in the control flow graph.

N is the **number of nodes** in the control flow graph.

Example

```
1  Program triangle2
2  Dim a,b,c As Integer
3  Dim IsATrinagle As Boolean
4  Output("Enter 3 integers which are sides of a triangle")
5  Input(a,b,c)
6  Output("Side A is", a)
7  Output("Side B is", b)
8  Output("Side C is", c)
9  If (a < b + c) AND (b < a + c) AND (c < a + b)
10    Then IsATriangle = True
11    Else IsATriangle = False
12 EndIf
13 If IsATriangle
14    Then  If (a = b) AND (b = c)
15             Then Output ("Equilateral")
16             Else  If (a≠b) AND (a≠c) AND (b≠c)
17                      Then Output ("Scalene")
18                      Else Output ("Isosceles")
19                   EndIf
20             EndIf
21   Else  Output("Nota a Triangle")
22 EndIf
23 End triangle2
```

```
1  NextDate Fragment
2  Dim day, month, year As Integer
3  Dim dayOK, monthOK, yearOK As Boolean
4  Do
5      Input(day, month, year)
6      If 0 < day < 32
7          Then dayOK = True
8          Else dayOK = False
9      EndIf
10     If 0 < month < 13
11         Then monthOK = True
12         Else   monthOK = False
13     EndIf
14     If 1811 < year < 2013
15         Then yearOK = True
16         Else   yearOK = False
17     EndIf
18 Until (dayOK AND monthOK AND yearOK)
19 End Fragment
```

Below question can be ask in unit 5 portion of exam also

Complexity testing :

Complexity testing is a verification technique where complexity of system design and coding is verified through **reviews, walkthroughs or inspections** as per planned arrangement. It is used in the following.

.

- Customer has some specific requirement for complexity measurements.
- Complex programs are difficult to maintain in future.
- More complex code is difficult to maintain as well as it may introduce few defects when complex decisions are executed. Black box testing also becomes complicated when code is very complicated.
- Applications have major algorithms and decision loops, and programmers may become victims of complexity while handling these decisions.
- Optimisation, simplicity and complexity must be balanced by designers and implemented by developers.
- Complexity leads to complex testing, and permutations and combinations of system increase infinitely. It may be possible that complex decisions may give wrong outputs.

CONTROL FLOW GRAPHS :

1. Control flow graphs describe the logic structure of software unit, where decisions can go while executing instructions.
2. Each flow graph consists of nodes and edges. The nodes represent statements or expressions, and the edges represent transfer of control between different nodes depending upon the condition it faces. When decisions are encountered, edges increase to more extent than the nodes, and represent complexity of the code.

Each possible execution path of a software module has a corresponding path from entry to exit node of the unit's control flow graph. This correspondence is the foundation for the structured testing methodology.

DEFINITION OF CYCLOMATIC COMPLEXITY :

Cyclomatic complexity is defined for each unit to be [e - n + 2], where 'e' represents number of edges and 'n' represents number of nodes in the control flow graph, respectively. More decisions add more edges than the nodes, and increase complexity of software.

5. Path testing concepts and examples

Complete Unit-4

Note : Any one question from V-model and VV model is confirm in exam

Verification- advantages and disadvantages

1. Verification is a disciplined approach to evaluate whether a software product fulfills the requirements or conditions imposed on them by the standards or processes.
2. It is done to ensure that the processes and procedures defined by the customer and/or organization for development and testing are followed correctly.
3. Verification is **also called 'static technique'** as it does not involve execution of any code, program or work product.
4. It is done by systematically reading and assessing the contents of work products with an intention of detecting defects with respect to standards or processes.
5. It helps in identification of defects and its location which then assists in correction.
6. Some people follow a methodology of finding and fixing verification defects then and there only.
7. One must do a root cause analysis and take corrective as well as preventive actions on the defects found.

6.2.1 ADVANTAGES OF VERIFICATION

- Verification can confirm that the work product has followed the processes correctly as defined by an organisation or customer (as the case may be). There is higher probability of success due to following processes religiously.
- It can find defects—in terms of deviations from standards—easily and also, the location of the defects. This helps in fixing the defects easily as well as conducting root cause analysis so that the same defects are not repeated.
- It can reduce the cost of finding and fixing defects as each work product is reviewed and corrected faster. Sometimes, defects are fixed as and when they are found.
- It can locate the defect easily as the work product under review is yet to be integrated with other work products. Cost of fixing defect is less as there is no/minimum impact on other parts of software.
- It can be used effectively for training people about processes and standards. Typically peer review, superior reviews, and walkthrough can be excellent tools for training.

6.2.2 DISADVANTAGES OF VERIFICATION

- It cannot show whether the developed software is correct or not. Rather, it shows whether the processes have been followed or not. If processes are not capable, then the outcome may not be good.

Scanned by CamScanner

Actual working software may not be assessed by verification as it does not cover any kind of execution of a work product. 'Fit for use' cannot be assessed in verification.

2. Verification - work bench with diagram (topic- 6.3)

Ans -

6.3 VERIFICATION WORK BENCH

A verification work bench is where verification activities are conducted, and may be a physical or virtual entity. For every work bench, the following basic things are required.

Fig. 6.1 Verification workbench

Inputs There must be some entry criteria definition when inputs are entering the work bench. This definition should match with the output criteria of the earlier work bench. For example, if we are verifying a code file, then we must have a written and compilable code as an input to the work bench along with verification plan, verification artifacts, etc.

Outputs Similarly to the entry criteria, there must be some exit criteria from work bench which should match with input criteria for the next work bench. Outputs may include review comments and the work product with defects if any.

Verification Process Verification process must describe step-by-step activities to be conducted in a work bench. It must also describe the activities done while verifying the work product under review.

Check Process Check process must describe how the verification process has been checked. It is not a verification of the work product but a verification of the process used for verification. Quality plan must define the objectives to be achieved, and check processes must verify that the objectives have been really achieved.

Standards, Tools, and Guidelines These may be termed 'the tools' available for conducting verification activities. There may be coding guidelines or testing standards available for verifications. Sometimes checklists are used for doing verification.

3. Methods of verification (topic - 6.4)

a. Self Review (Advantages and disadvantages)
b. Peer Review (Advantages and disadvantages) - online / offline
c. Walkthrough (Advantages and disadvantages)
d. Inspection (Advantages and disadvantages)
e. Audits (Advantages and disadvantages)

Ans-

- **Self Review** Self review may not be referred as an official way of review in most of the software verification descriptions, as it is assumed that everybody does a self check before giving work product for further verification. One must capture the self review records and defects found in self review to improve the process. It is basically a self-learning and retrospection process.

- **Peer Review** Peer review is the most informal type of review where an author and a peer are involved. It is a review done by a peer and review records are maintained. A peer may be a fellow developer or tester as the case may be. There is also a possibility of superior review where peer is a supervisor with better knowledge and experience.

- **Walkthrough** Walkthrough is a semi formal type of review as it involves larger teams along with the author reviewing a work product. It may involve a project team or part of a project team doing a review jointly as the case may be.

- **Inspection (Formal Review)** Inspection is a formal review where people external to the team may be involved as inspectors. They are 'subject matter experts' who review the work product. It is also termed 'Fagan's inspection'.

- **Audits** Audit is a formal review based on samples. Audits are conducted by auditors who may or may not be experts in the given work product.

SELF REVIEW

1. Self review is not considered as an official review in software development as it is expected to be done by the author or creator of an artifact by default.
2. It is a primary expectation that no one must say that his work is done before doing a self review and being satisfied that there are no problems in the output.
3. Self review is excellent in defect prevention through self learning or retrospection.
4. Defects found in self review can help in self education and self improvement.

Advantages of Self Review

1. Self reviews are highly flexible with respect to time and defect finding, as one need not take an appointment or allocate specific time while doing it. It may be an online activity.
2. Self review is an excellent tool for self learning, as learning from mistakes is a good way of Improvement. There is no ego involved in self review as the findings need not be shared with anybody.

Disadvantages of Self Review
1) Approach- or understanding-related defects may not be found in self review as it is difficult to think in some other way.
2) People involved in self review may not conduct a review in reality due to time and focus issues. For example, developers may concentrate more on creating code rather than reviewing the code developed by them.
3) Something which has been implemented correctly in initial development may get changed due to personal issues. Some people with less self confidence may change the correct implementation.

PEER REVIEW

Peer reviews are conducted frequently in software development life cycle at various stages of development. Example of peer review could be code review done by a peer or fellow developer, test case review done by fellow tester and so on.

There are 2 kinds of peer review :

Online Peer Review (Peer-to-Peer Review) In such kind of review, the author and reviewer meet together and review the work product jointly. Any explanation required by the reviewer may be provided by the author. The extreme programming way of development recommends online review as defects are found and corrected jointly by the peers.

Offline Peer Review (Peer Review) In such kind of review, the author informs the reviewer that the work product is ready for review. The reviewer may review the work product as per his time availability. The review report is send to the author along with the defects, if any. The author may decide to accept/reject the review comments or defects identified. This review is prone to mistakes as there is no joint review, and the author and the reviewer do not discuss the defects. It is more on an opinion basis.
Advantages:
1. It is highly informal & unplanned
2. Defects are discussed & decision are taken informally

Disadvantages:
1. Reviewer might not be an expertise
2. Peer might fix defects but won't show mistake of his friends

4.Draw & Explain V- Model Of Software testing

Ans -

Validation model describes the **validation activities** associated with different phases of software development.

Fig. 7.1 V model for testing (Validation model)

1. At the requirement phase, there is system testing and acceptance testing, where system testers and users confirm that requirements have been really met or not.
2. Design phase is associated with interface testing which covers design specification testing as well as structural testing.
3. Program-level designs are associated with integration testing.
4. At code level, unit testing is done to validate individual units.

5.Draw & Explain VV- Model Of Software testing

Ans -

'VV model' considers all the activities related to verification as well as validation. It is **also termed 'Verification and Validation Model' or 'Quality model'.** 'VV model' talks about verification and validation activities associated with software development during the entire life cycle.

Fig. 7.2 VV model (Verification and Validation model

Requirement : The intention would be to find if there is any gap existing between user requirements and requirement definition, and also to check whether all the requirements have been captured correctly and completely or not.

Requirement Verification : Generally requirement verification is done through **inspection** of requirement specification document using checklist, standards or guidelines. Experts in domain, customer representative, and other stakeholders may be involved in conducting such an inspection. Outcome of the inspection must be recorded and defects must be corrected before going further to next stages of development life cycle.

Requirement Validation : Requirement validation takes place at **two or more stages** during software development. The first stage of validation involves writing complete **use cases by** referring to requirement statements. The second stage of validation is through **system testing**. System test scenarios and test cases are defined using requirement specifications.

Design : Design may include high-level design or architectural design, and low-level design or detail design. Designs are created by architects (high-level designs)/designers (low-level designs) as the case may be.

Design Verification : Verification of design may be a **walkthrough** of a design document by design experts, team members and stakeholders of the project. Project team along with the architect / designer may walkthrough the design to find the completeness and give comments, if any. Traceability of design with requirement must be established in the requirement traceability matrix.

Design Validation : Validation of design can happen at **two or more stages** during software development life cycle. The first stage of validation happens when **data flow diagrams can be created** by referring to the design document. If the flow of data is complete, design is considered to be complete. Any interruption in flow of data indicates lacunae in design. This is

also termed 'flow anomaly'. Sometimes, there is a circular reference in data flow which will lead to a loop which will never stop.the second stage of validation happens at **integration testing and interface testing.**

Coding : Coding is an activity of writing individual units as defined in low level design. It is done by developers where low-level designs are implemented. At some places, database (including tables and stored procedures) is also handled by different teams.

Code Verification : As coding is done, it undergoes a code review (generally peer review). Peer review helps in identification of errors with respect to coding standards, indenting standards, commenting standards, and variable declaration issues. Checklist approach is used in code review.

Code Validation : Validation of coding happens through unit testing where individual units are tested separately. **The developer may have to write special programs (such as stubs and drivers) so that individual units can be tested.** The executable is created by combining stubs, drivers and the units under testing. This executable is tested with the test cases mainly derived from low-level design.

5. Levels of Validation (topic- 6.12)
    a. Unit Testing
    b. Integration testing

6. Coverage offered during validation - Requirement coverage, Functionality coverage, Feature coverage

7. Coverage in verification testing (Topic- 6.8)
a. Statement Coverage
b. Path Coverage
c. Decision Covearge

Complete Unit-5

Unit -5

1. Proposal Testing (topic- 9.2)
2. Requirement Testing (topic- 9.3)
3. Design Testing (topic- 9.4)
4. Code Review (topic- 9.5)
5. Unit testing (topic- 9.6))
6. Difference between debugging and testing (9.6.1)
7. Integration testing (9.8)
    a. Bottom-up testing
    b. Top-Down Testing
    c. Stubs/ Drivers
8. BIG-Bang Testing
9. Sandwich testing
10. Complexity testing (topic -11.2)
11. Graphical User interface testing (Topic- 11.3)
12. Compatibility testing (Topic- 11.4)
13. Types of compatibility (Topic- 11.4.4)
14. Internationalization (topic- 11.4.5) - Scope
15. Security Testing- (11.5)
16. Performance testing , Volume testing and stress testing (11.6)
17. Recovery Testing (11.7)
18. Installation testing(11.8)
19. Regression testing, Smoke testing, Sanity testing
20. Usability testing (11.22)
21. COT's Testing(12.7)

Q5 **Attempt any Two from the following.** [1

a) What is Big-Bang testing? What are the advantages and disadvantages of Big-Bang testing?

b) Describe requirement verification and validation process.

c) Write a note on the importance of 'Graphical User Interface' testing.

d) What is Load and Performance testing?

********** END **********

Q5 **Attempt any two of the following**

A How does customer prepare a proposal? What is the need of requirement testing?

B State the importance of compatibility testing. What are the different types of compatibility?

C Discuss the use of any two of the following testing techniques –

i) Installation Testing

ii) Regression Testing

iii) Control Testing

iv) Compliance Testing

D What are testing approaches of web application testing?

****

Q.5. Answer any one of the following:

(a) What is bottom- up integration testing? What is the role of drivers in bottom-up testing?

(b) What is recovery testing? Discuss some ways of disaster recovery of a system.

Q.1] What is Big-Bang testing? What are the advantages and disadvantages of Big-Bang testing?

Ans:

Big-bang approach is the most commonly seen approach at many places, where the system is tested completely after development is over. There is no testing of individual units/modules and integration sequence. System testing tries to compensate for any other kind of testing, reviews, etc. Sometimes, it includes a huge amount of random testing which may not be repeatable.

Advantages of Big-Bang Approach

1. It gives a feeling that cost can be saved by limiting testing to the last phase of development.
2. Testing is done as a last-phase of the development lifecycle in the form of system testing.
3. Time for writing test cases and defining test data at unit level, integration level, etc may be saved.
4. Big-bang approach is very fast, It may not give adequate confidence to users as all permutations and combinations cannot be tested in this testing. Even the test log may not be maintained. Only defects are logged in defect-tracking tool

Disadvantages of Big-Bang Approach

1. Problems found in this approach are hard to debug. It is difficult to say that system interfaces are working correctly and will work in all cases.
2. Big-bang approach executes the test cases without any understanding of how the system is built.
3. Location of defects may not be found easily. In big bang testing, even if we can reproduce the defects, it can be very difficult to locate the problematic areas for correcting them.
4. Interface faults may not be distinguishable from other defects.

Q.2] What is Load and Performance testing ?
Ans -

1. Performance testing is intended to find whether the system meets its performance requirements under normal load or normal level of activities. Normal load must be defined by the requirement statement.
2. Performance criteria must be expressed in numerical terms. Design verification can help in determining whether required measures have been taken to meet performance requirements or not.
3. This is one area where verification does not work to that much extent, and one needs to test it by actually performing the operations on the system.
4. Performance criteria must be measurable in quantitative terms such as time in 'milliseconds'. Some examples of performance testing can be as given below.

- Adding a new record in database must take maximum five milliseconds. It means that when the record is added in database, it may take time lesser or equal to five milliseconds.
- Searching of a record in a database containing one million records must not take more than one second. One must add one million records in the system, and then use the search criteria to test this.
- Sending information of say one MB size across the system with a network of 512 KBPS must not take more that one minute. User may have to try various combinations in this testing.

Volume (Load) Testing :

1. Volume testing talks about the maximum volume or load a system can accept before it collapses due to load or volume.
2. It can be described in terms of concurrent users, number of connections, maximum size of an attachment to be transferred over a network, etc.
3. When the volume is above normal, it will definitely have poor performance. One must not check the performance under volume testing. Only factor to be assessed is whether the system is living or dead due to increased volume or load on the system.

Usage of Volume Testing in Design : There are different options available in designing with respect to volume testing. It is governed by a '**factor of safety**' expected by the user and the consequences of a load level that is more than the expected load.

1. One way is not to allow more number of users than the stipulated number, so that the system does not undergo heavy load at all.
2. Another way can be to allow users to join the system and give them warnings that they are extra joiners and may experience slow responses from the system due to load.
3. This approach can be used to create session timeouts if a user is not using a system for some stipulated time, so that new users can join the system without overloading it. This can be used along with security

Q.3] regression , smoke,sanity,sandwich testing?
Ans -

11.10 REGRESSION TESTING

Regression testing is intended to determine whether the changed components have introduced any error in unchanged components of the system. Regression testing may not be considered as special testing in development projects. But often, maintenance projects may consider it as special testing as regression testing of entire application may be very costly. Regression testing can be done at,

- Unit level to identify that changes in the units have not affected its intended purpose, and other parts of the unit are working properly even after the changes are made in some parts.
- Module level to identify that the module behaves in a correct way after the individual units are changed. Whatever was a correct behavior before making a change must be retained, but where it was failing and change was necessary must also behave correctly now (second part is called retesting).
- System level to identify that the system is performing all the correct actions that it was doing previously as well as actions intended by requirements after change is made in some parts of the system.

Regression testing is performed where there is high risk that changes in one part of software may affect unchanged components or system adversely. It is done by rerunning previously conducted successful tests

Smoke Testing :

1. Smoke testing involves testing basic functionality of software application developed to ensure that application is living and one can work with it.
2. Generally, these tests are performed without any user input. Steps involved in smoke testing can be installation, navigating through the application, invoking or accessing some major functionalities. Smoke testing is not for approving or certifying the application. It only tells the tester whether the application is alive or not.
3. Smoke testing is not applied for testing the application but to ensure that normal user will be able to work with it. If smoke testing fails, user will not be able to work with the application. Such failure may result into rejection of an application without going further.
4. Success of smoke testing is one of the **entry criteria for system testing.** If smoke testing fails, system cannot be taken for further testing. Generally, smoke testing is done by a test manager or senior tester, as the case may be. Smoke testing is also termed 'Smell test' as the test manager may have to make a judgment about the system in very less time.

SANITY TESTING :

1. Sanity testing is performed to test the major behavior or functionality of the application.
2. Depth of sanity testing is more than smoke testing. It normally includes a set of core tests of basic GUI and functionality to demonstrate connectivity to database, application servers, printers, etc.

3. Some people consider smoke testing and sanity testing as the same while some differentiate between them. Sanity testing is also known as **'Build Verification Testing (BVT)'** where the testers test if consistent results can be obtained when some test cases are executed.
4. Sanity testing helps in identifying whether the test case results are consistent or not, and also application behaviors are reproducible or not. If sanity testing fails, application may be rejected without doing any further testing. Success of sanity testing is another entry criterion before system testing. Generally, sanity testing is done by a test manager or senior tester, as the case may be.

9.10 SANDWICH TESTING

Sandwich testing defines testing into two parts, and follows both parts starting from both ends i.e., top-down approach and bottom-up approach either simultaneously or one after another. It combines the advantages of bottom-up testing and top-down testing at a time. Figure 9.5 shows a sandwich testing approach.

Fig. 9.5 Sandwich testing approach

9.10.1 PROCESS OF SANDWICH TESTING

- Bottom-up testing starts from middle layer and goes upward to the top layer. Generally for very big systems, bottom-up approach starts at subsystem level and goes upwards.
- Top-down testing starts from middle layer and goes downward. Generally for very big systems, top-down approach starts at subsystem level and goes downwards.
- Big-bang approach is followed for the middle layer. From this layer, bottom-up approach goes upwards and top-down approach goes downwards.

Q.4] Integration testing :

Top-down

Bottom-up

stubs/drivers

TOP-DOWN TESTING

1. In the top-down testing approach, the top level of the application is tested first and then it goes downward till it reaches the final component of the system.
2. All top-level components called by tested components are combined one by one and tested in the process. Here, integration goes downward. **Top-down approaches need design and implementation of stubs.** Drivers may not be required as we go downward driver for latter phase while one may have to design stubs to take care of lower-level components which are not available at that time.
3. Top-level components are the user interfaces which are created first to elicit user requirements or creation of prototype, Agile approaches **like prototyping, formal proof of concept, test-driven development use this approach for testing**. Figure shows a top down integration and testing.

Fig. 9-3 Top-down integration approach

Bottom-up testing

Bottom-up testing approach focuses on testing the bottom part/individual units and modules, and then goes upward by integrating tested and working units and modules for system testing and intersystem testing. The process goes as mentioned below. Figure 9.2 shows a bottom-up integration and testing.

1. Units at the lowest level are tested using stubs / drivers. stubs/drivers are designed for a special purpose .they must be tested before using them for unit testing
2. Once the units are tested and found to be working, they are combined to form modules. Modules may need only drivers, as the low-level units which are already tested and found to be working modules may act as stubs.
3. Bottom-up approach is also termed **'classical approach'** as it may indicate a normal way of doing things. Theoretically, it is an excellent approach but practically, it is very difficult to implement.

Fig. 9.2 Bottom-up integration approach

Stubs and drivers

Stubs and drivers are special-purpose arrangements, generally code, required to test the units individually which can act as an input to the unit/module and can take output from the unit/module

Stub is a piece of code emulating a called function. In absence of a called function, stub may take care of that part for testing purposes.

Driver is a piece of code emulating a calling function. In absence of actual function calling the code under testing, the driver tries to work as a calling function.

- Stubs are mainly created for integration testing, such as in the top-down approach.
- Drivers are primarily created for integration testing, like in the bottom-up approach.
- Stubs and drivers must be very simple to develop and use. They must not introduce any defects in the application.It is not a rule that stubs and drivers must be taken into consideration when delivering code to the customer or user.
- Reusability of stubs and drivers can improve productivity, but designing and coding multipurpose stubs and drivers can be a big challenge.
- Estimation must consider the time required for creating stubs and drivers.

Q..5] COTS testing ?
Ans -

'COTS' stands for **'Commercially Of The Shelf 'software**. These softwares are readily available in the market and users can buy and use them directly. These may be integrated in a new development, or used for development or testing activities as a tool.

WHY SOFTWARE ORGANIZATIONS USE COTS:
There are many limitations for a development organization that prevent it from making its own required software, and one has to make a decision to buy 'COTS'. Some of the reasons are given below.

1. **Line of Business**: An organization may not be in the line of business of developing certain types of software that it requires.
2. **Cost-Benefit Analysis**: Sometimes, it is more cost-effective to acquire software from the market rather than developing it in-house.
3. **Delivery Schedules**: COTS products are readily available for purchase, typically with predictable delivery schedules, by paying a price.

Features of COTs Testing:

COTS' are developed with general requirements collected from the market. It may not exactly match the organisation's needs and expectations. One must find the percentage fit of 'COTS' to the organisation business and then decide whether it is successful or not.

Some 'COTS' may need changing business processes to suit the 'COTS' implementation in the organisation. This is another way of Business Process Reengineering (BPR) for an organization where internationally proven practices can be implemented by using 'COTS'.

'COTS' may have some of their best processes accepted at national/international level, and organizations may benefit by using such processes along with the product.

Sometimes, 'COTS' may need configuration of software or system to suit business needs.

Generally, when 'COTS' are implemented, many business rules must be defined to customize the software to the organisation.