Royal Education Society's

COCSIT LATUR

FACULTY OF COMPUTER STUDIES

[NEW CBCS PATTERN]

B Sc. CS TY (Semester VI)

PRE-SEMESTER EXAMINATION NOVEMBER 2022

Answers Sheet

Software Testing (BCS-504A)

(Date:-__/__/2022) ***Time:*** *Three Hours* ***Maximum Marks*** *- 75*

Instructions to the candidates:

1. All questions are Compulsory.

2. Figures to the right indicate full marks.

3. Assume suitable data, if required.

Q.1 Attempt any FIVE of the following (3 Marks each) 15

a) Explain Quality and Software Quality?

Quality:

The degree to which a component, system or process meets specified requirements and/or user/customer needs and expectations.

Software Quality:

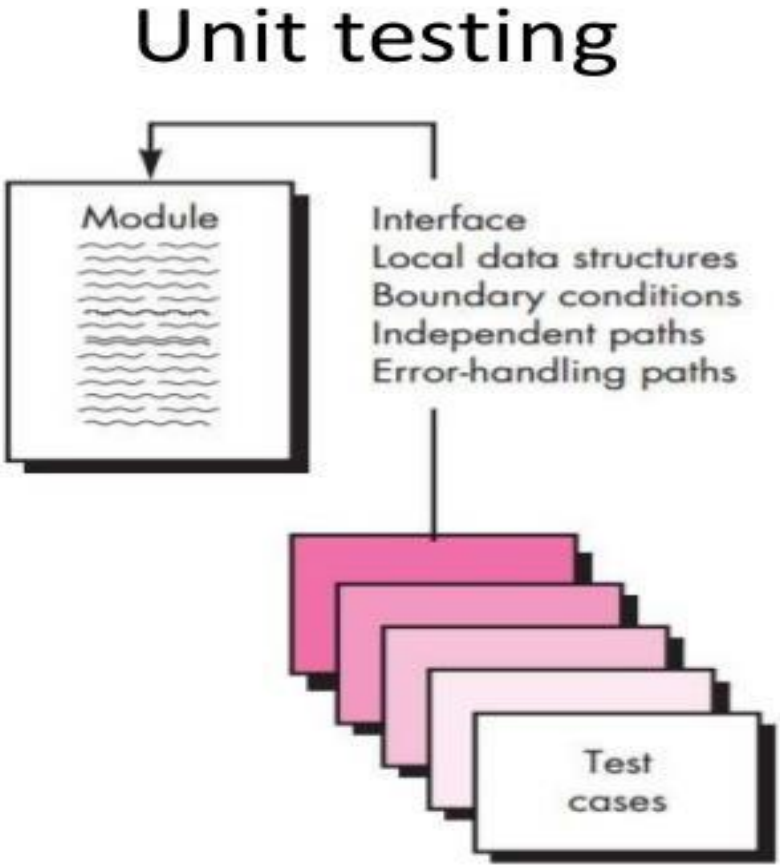
Software quality product is defined in term of its fitness of purpose. That is, a quality product does precisely what the users want it to do. For software products, the fitness of use is generally explained in terms of satisfaction of the requirements laid down in the SRS document. Although "fitness of purpose" is a satisfactory interpretation of quality for many devices such as a car, a table fan, a grinding machine, etc. For software products, "fitness of purpose" is not a wholly satisfactory definition of quality.

Example: Consider a functionally correct software product. That is, it performs all tasks as specified in the SRS document. But, has an almost unusable user interface. Even though it may be functionally right, we cannot consider it to be a quality product.

b) Describe Unit Testing.

Unit testing :

- Focuses testing on the function or software module
- Concentrates on the internal processing logic and data structures
- Is simplified when a module is designed with high cohesion
  - Reduces the number of test cases
  - Allows errors to be more easily predicted and uncovered
- Concentrates on critical modules and those with **high cyclomatic complexity** when testing resources are limited



c) Define Metrics, measure and Measurement.

- Measure
  - Provides a quantitative indication of the extent, amount, dimension, capacity, or size of some attribute of a product or process
- Measurement
  - The act of determining a measure
- Metric
  - (IEEE) A quantitative measure of the degree to which a system, component, or process possesses a given attribute

d) Explain Internal and External Views of testing.

Any engineered product (and most other things) can be tested in one of two ways:
1. Knowing the specified function that a product has been designed to perform, tests can be conducted that demonstrate each function is fully operational while at the same time searching for errors in each function (BLACK-BOX testing)
2. Knowing the internal workings of a product, tests can be conducted to ensure that "all gears mesh," that is, internal operations are performed according to specifications and all internal components have been adequately exercised (WHITE-BOX testing)

e) Describe Content Testing.

Content Testing:

Content testing has three important objectives

- 1. To uncover syntactic errors(for eg., typos, grammar mistakes) in the text-based documents, graphical representations, and other media
- 2. To uncover semantic errors(i.e., focuses on the information presented within each content object)
- 3. To find errors in the organization or structure of the content that is presented to the end user.

f) Explain Software Quality Factors

software quality model identifies 4 main quality characteristics, namely:

Efficiency:

This characteristic is concerned with the system resources used when providing the required functionality. The amount of disk space, memory, network etc. provides a good indication of this characteristic.

Intuivteness:

intuitive software design combines art and science to save us both time and mental stress. Application programs that have a friendly interface and are easy to use.

Richness:

The ability of a software to embed many qualities is called its richness.

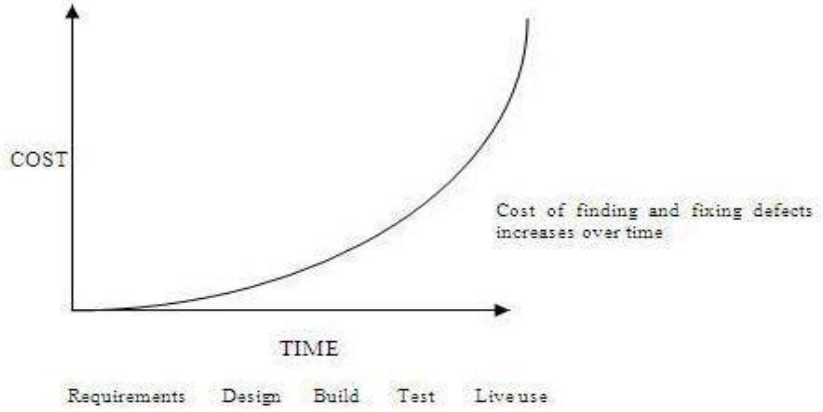
Robustness:

Robustness is the ability of a computer system to cope with errors during execution and cope with erroneous input.

g) Explain the Cost Impact of Software Defects.

The cost of defects identified during Software Testing, completely depends on the impact of the defects found. The earlier the defect is found, easier and less costly it is to fix these defects. For instance, if there is a defect found in the project requirement specifications and analysis, then it is relatively cheaper to fix it.

Similarly, if the defects or failures are found in the design of the software, then the product design is corrected and then re-issued. However, if these defects somehow get missed by testers and if they are identified during the user acceptance phase, then it can be way too expensive to fix such type of errors.



Q. 2 Attempt any three of the following (5 Marks each) 15

a) Explain Different software reviews.

A review is a systematic examination of a document by one or more people with the main aim of finding and removing errors early in the software development life cycle. Reviews are used to verify documents such as requirements, system designs, code, test plans and test cases.

Reviews are usually performed manually while static analysis of the tools is performed using tools.

Types of Review:

1) Informal: As suggested by its name, this is an informal type of review, which is extremely popular and is widely used by people all over the world. Informal review does not require any documentation, **"entry criteria"**, or a large group of people. It is a time saving process that is not documented.

2) "Walkthrough": Here, a designer or developer lead a team of software developers to go through a software product, where they ask question and make necessary comments about various defects & errors. This process differs from **"software inspection"** and technical review in various aspects.

3) Formal Technical Review: During the process of technical review a team of qualified personnel's review the software and examine its suitability to define its intended use as well as to identify various discrepancies.

4) Inspection: This is a formal type of peer review, wherein experienced & qualified individuals examine the software product for bugs and defects using a defined process. Inspection helps the author improve the quality of the software.

b) Explain Integration testing in details.

- Defined as a systematic technique for constructing the software architecture
  - At the same time integration is occurring, conduct tests to uncover errors associated with interfaces
- Objective is to take unit tested modules and build a program structure based on the prescribed design
- Two Approaches
  - Non-incremental Integration Testing
  - Incremental Integration Testing

Incremental Integration Testing

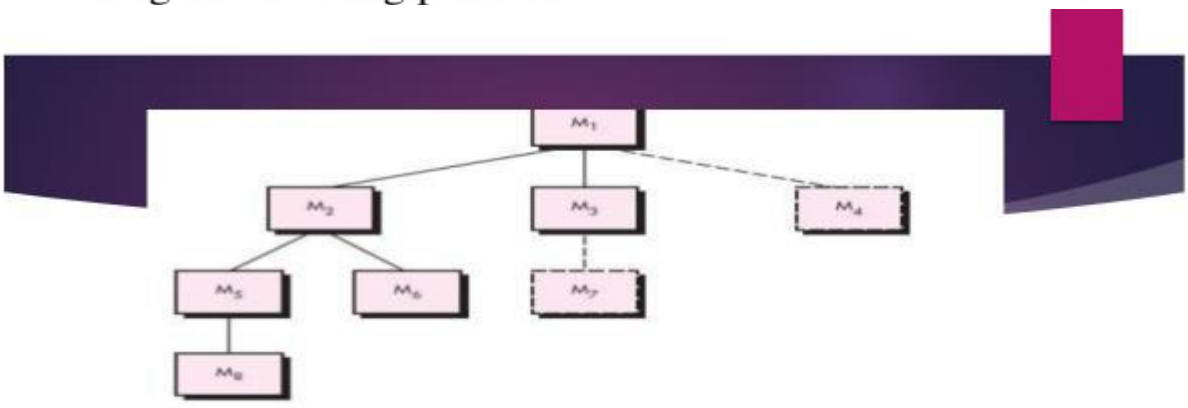
- The program is constructed and tested in small increments
- Errors are easier to isolate and correct
- Interfaces are more likely to be tested completely
- A systematic test approach is applied
- Different incremental integration strategies
  - Top-down integration
  - Bottom-up integration
  - Regression testing
  - Smoke testing

Non-incremental Integration Testing

- Uses "Big Bang" approach
- All components are combined in advance
- The entire program is tested as a whole Chaos results
- Many seemingly-unrelated errors are encountered
- Correction is difficult because isolation of causes is complicated
- Once a set of errors are corrected, more errors occur, and testing appears to enter an endless loop

Top-down Integration

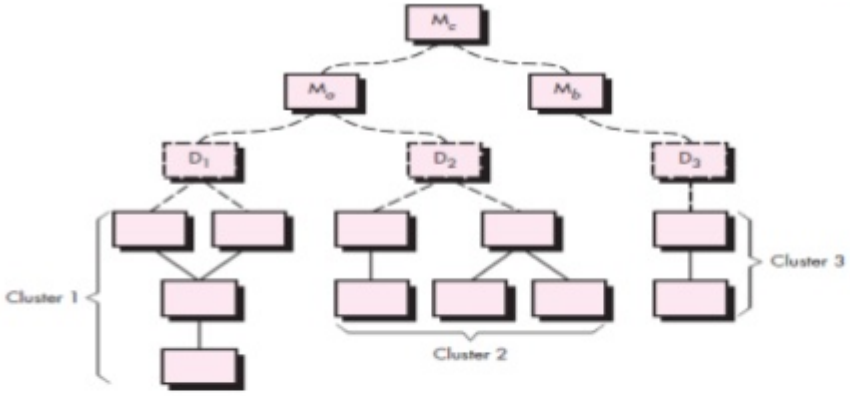
- Modules are integrated by moving downward through the control hierarchy, beginning with the main module
- Subordinate modules are incorporated in two ways :
  - depth-first : All modules on a major control path are integrated
  - breadth-first : All modules directly subordinate at each level are integrated
- Advantages
  - This approach verifies major control or decision points early in the test process
- Disadvantages
  - Stubs need to be created to substitute for modules that have not been built or tested yet; this code is later discarded
  - Because stubs are used to replace lower level modules, no significant data flow can occur until much later in the integration/testing process



Forexample,selecting
the left-hand path, components M1, M2 , M5 would be integrated first. Next,M8 or (if necessary for proper functioning of M2) M6 would be integrated. Then, the central and right-hand control paths are built.

Bottom-up Integration

- Integration and testing starts with the most atomic modules in the control hierarchy
- Advantages
  - This approach verifies low-level data processing early in the testing process
  - Need for stubs is eliminated
- Disadvantages
  - Driver modules need to be built to test the lower-level modules; this code is later discarded or expanded into a full-featured version
  - Drivers inherently do not contain the complete algorithms that will eventually use the services of the lower-level modules; consequently, testing may be incomplete or more testing may be needed later when the upper level modules are available



Integration follows the pattern illustrated in Figure Components are combined to form clusters 1, 2, and 3. Each of the clusters is tested using a driver (shown as a dashed block). Components in clusters 1 and 2 are subordinate to M*a*. Drivers D1and D2 are removed and the clusters are interfaced directly to M*a*. Similarly, driver D3 for cluster 3 is removed prior to integration with module M*b*. Both M*a* and M*b* will ultimately be integrated with component M*c*, and so forth.

21

c) Describe Security and Quality.

Security testing is a type of Software Testing that uncovers vulnerabilities, threats, risks in a software application and prevents malicious attacks from intruders. The purpose of Security Tests is to identify all possible loopholes and weaknesses of the software system which might result in a loss of information, revenue, repute at the hands of the employees or outsiders of the Organization.

 Types of Security Testing:

There are seven main types of security testing as per Open Source Security Testing methodology manual. They are explained as follows:

1.Vulnerability Scanning: This is done through automated software to scan a system against known vulnerability signatures.

2.Security Scanning: It involves identifying network and system weaknesses, and later provides solutions for reducing these risks. This scanning can be performed for both Manual and Automated scanning.

3.Penetration testing: This kind of testing simulates an attack from a malicious hacker. This testing involves analysis of a particular system to check for potential vulnerabilities to an external hacking attempt.

4.Risk Assessment: This testing involves analysis of security risks observed in the organization. Risks are classified as Low, Medium and High. This testing recommends controls and measures to reduce the risk.

5.Security Auditing: This is an internal inspection of Applications and Operating systems for security flaws. An audit can also be done via line by line inspection of code

6.Ethical hacking: It's hacking an Organization Software systems. Unlike malicious hackers, who steal for their own gains, the intent is to expose security flaws in the system.

7.Posture Assessment: This combines Security scanning, Ethical Hacking and Risk Assessments to show an overall security posture of an organization.

Quality Control:

 SQC is a set of activities for ensuring quality in software products. Software Quality Control is limited to the Review/Testing phases of the Software Development Life Cycle and the goal is to ensure that the products meet specifications/requirements.

SQC Activities

It includes the following activities:

Reviews

Requirement Review
Design Review
Code Review
Deployment Plan Review
Test Plan Review
Test Cases Review

Testing

Unit Testing
Integration Testing
System Testing
Acceptance Testing

d) Explain Metrics for the requirements model.

Metrics for the Analysis Model

- Functionality delivered
  - Provides an indirect measure of the functionality that is packaged within the software
- System size
  - Measures the overall size of the system defined in terms of information available as part of the analysis model
- Specification quality
  - Provides an indication of the specificity and completeness of a requirements specification

Introduction to Function Points

- First proposed by Albrecht in 1979; hundreds of books and papers have been written on functions points since then
- Can be used effectively as a means for measuring the functionality delivered by a system
- Using historical data, function points can be used to
  - Estimate the cost or effort required to design, code, and test the software
  - Predict the number of errors that will be encountered during testing
  - Forecast the number of components and/or the number of projected source code lines in the implemented system
- Derived using an empirical relationship based on
  1) Countable (direct) measures of the software's information domain
  2) Assessments of the software's complexity

Information Domain Values

- Number of external inputs
  - Each external input originates from a user or is transmitted from another application
  - They provide distinct application-oriented data or control information
  - They are often used to update internal logical files
  - They are not inquiries (those are counted under another category)
- Number of external outputs
  - Each external output is derived within the application and provides information to the user
  - This refers to reports, screens, error messages, etc.
  - Individual data items within a report or screen are not counted separately
- Number of external inquiries
  - An external inquiry is defined as an online input that results in the generation of some immediate software response
  - The response is in the form of an on-line output
- Number of internal logical files
  - Each internal logical file is a logical grouping of data that resides within the application's boundary and is maintained via external inputs
- Number of external interface files
  - Each external interface file is a logical grouping of data that resides external to the application but provides data that may be of use to the application

e) Describe Software Quality Assurance Activities in details.

Assurance (SQA) is simply a way to assure quality in the software. It is the set of activities which ensure processes, procedures as well as standards suitable for the project and implemented correctly. Software Quality Assurance is a process which works parallel to development of a software. It focuses on improving the process of development of software so that problems can be prevented before they become a major issue. Software Quality Assurance is a kind of an Umbrella activity that is applied throughout the software process

SQA Activities :

1) Creating an SQA Management Plan:

The foremost activity includes laying down a proper plan regarding how the SQA will be carried out in your project.

Along with what SQA approach you are going to follow, what engineering activities will be carried out, and it also includes ensuring that you have a right talent mix in your team.

2) Setting the Checkpoints:

The SQA team sets up different checkpoints according to which it evaluates the quality of the project activities at each checkpoint/project stage. This ensures regular quality inspection and working as per the schedule.

3) Apply software Engineering Techniques:

Applying some software engineering techniques aids a software designer in achieving high-quality specification. For gathering information, a designer may use techniques such as interviews and FAST (Functional Analysis System Technique).

Later, based on the information gathered, the software designer can prepare the project estimation using techniques like WBS (work breakdown structure), SLOC (source line of codes), and FP(functional point) estimation.

4) Executing Formal Technical Reviews:

An FTR is done to evaluate the quality and design of the prototype.

In this process, a meeting is conducted with the technical staff to discuss regarding the actual quality requirements of the software and the design quality of the prototype. This activity helps in detecting errors in the early phase of SDLC and reduces rework effort in the later phases.

5) Having a Multi-Testing Strategy:

By multi-testing strategy, we mean that one should not rely on any single testing approach, instead, multiple types of testing should be performed so that the software product can be tested well from all angles to ensure better quality.

6) Enforcing Process Adherence:

This activity insists the need for process adherence during the software development process. The development process should also stick to the defined procedures.

This activity is a blend of two sub-activities which are explained below in detail:

(i) Product Evaluation:

This activity confirms that the software product is meeting the requirements that were discovered in the project management plan. It ensures that the set standards for the project are followed correctly.

(ii) Process Monitoring:

This activity verifies if the correct steps were taken during software development. This is done by matching the actually taken steps against the documented steps.

7) Controlling Change:

In this activity, we use a mix of manual procedures and automated tools to have a mechanism for change control.

By validating the change requests, evaluating the nature of change and controlling the change effect, it is ensured that the software quality is maintained during the development and maintenance phases.

8) Measure Change Impact:

If any defect is reported by the QA team, then the concerned team fixes the defect.

After this, the QA team should determine the impact of the change which is brought by this defect fix. They need to test not only if the change has fixed the defect, but also if the change is compatible with the whole project.

For this purpose, we use software quality metrics which allows managers and developers to observe the activities and proposed changes from the beginning till the end of SDLC(Software Development Life Cycle) and initiate corrective action wherever required.

9) Performing SQA Audits:

The SQA audit inspects the entire actual SDLC process followed by comparing it against the established process.

It also checks whatever reported by the team in the status reports were actually performed or not. This activity also exposes any non-compliance issues.

10) Maintaining Records and Reports:

It is crucial to keep the necessary documentation related to SQA and share the required SQA information with the stakeholders. The test results, audit results, review reports, change requests documentation, etc. should be kept for future reference.

11) Manage Good Relations:

In fact, it is very important to maintain harmony between the QA and the development team.

1) We often hear that testers and developers often feel superior to each other. This should be avoided as it can affect the overall project quality.

Q. 3 Attempt any three of the following (5 Marks each) **15**

a) Explain Validation testing in details

Validation Testing

- Validation testing follows integration testing
- The distinction between conventional and object-oriented software disappears and Focuses on user-visible actions and user-recognizable output from the system

Validation test criteria :

- Demonstrates conformity with requirements
- Designed to ensure that All functional requirements are satisfied,All behavioral characteristics are achieved,All performance requirements are attained
- Documentation is correct
- Usability and other requirements are met (e.g., transportability, compatibility, error recovery, maintainability)
- After each validation test
  - The function or performance characteristic conforms to specification and is accepted
  - A deviation from specification is uncovered and a deficiency list is created

Configuration review:

- The intent of this review is to ensure that all elements of th software configuration have been properly developed, are cataloged, and have the necessary detail to bolster the support activities

Alpha and beta testing :

- Alpha testing conducted at the developer's site by end use
  - Software is used in a natural setting with developers watching intently
  - Testing is conducted in a controlled environment
- Beta testing conducted at end-user sites
  - Developer is generally not present
  - It serves as a live application of the software in an environment that cannot be controlled by the develope
  - The end-user records all problems that are encounterec and reports these to the developers at regular intervals
- After beta testing is complete, software engineers make software modifications and prepare for release of the software product to the entire customer base

b) Describe the Cost of Quality in details.

Cost of Quality (COQ) is a measure that quantifies the cost of control/conformance and the cost of failure of control/non-conformance. In other words, it sums up the costs related to prevention and detection of defects and the costs due to occurrences of defects.

Prevention Cost

The cost arises from efforts to prevent defects.

Example: Quality Assurance costs

Appraisal Cost

The cost arises from efforts to detect defects.

Example: Quality Control costs

Cost of Failure of Control (Also known as Cost of Non-Conformance)

Internal Failure Cost

The cost arises from defects identified internally and efforts to correct them.

Example: Cost of Rework (Fixing of internal defects and re-testing)

External Failure Cost

The cost arises from defects identified by the client or end-users and efforts to correct them.

Example: Cost of Rework (Fixing of external defects and re-testing) and any other costs due to external defects (Product service/liability/recall, etc)

FORMULA / CALCULATION

Cost of Quality (COQ) = Cost of Control + Cost of Failure of Control

where

Cost of Control = Prevention Cost + Appraisal Cost

and

Cost of Failure of Control = Internal Failure Cost + External Failure Cost

c) Explain the White-Box Testing in details.

White box testing

- White box testing is also called as glass-box testing
- Using white-box testing methods can derive test cases that
  - guarantee that all independent paths within a module have been exercised at least once
  - exercise all logical decisions on their true and false sides
  - execute all loops at their boundaries and within their operational bounds
  - exercise internal data structures to ensure their validity

White box testing involves the testing of the software code for the following:
Internal security holes

Broken or poorly structured paths in the coding processes

The flow of specific inputs through the code

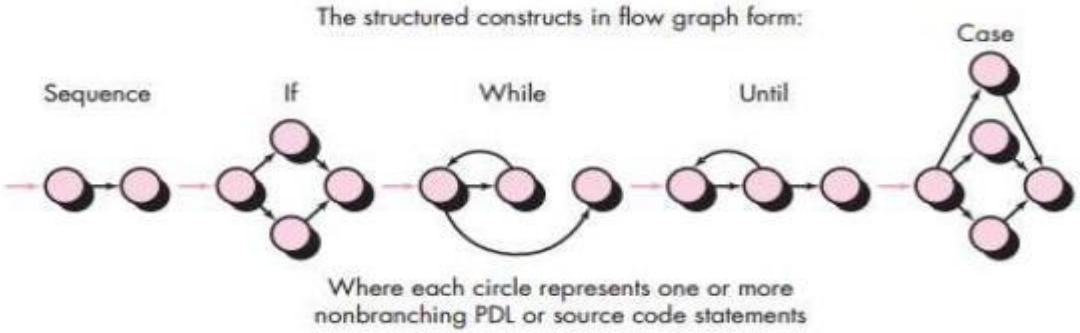
Expected output

The functionality of conditional loops

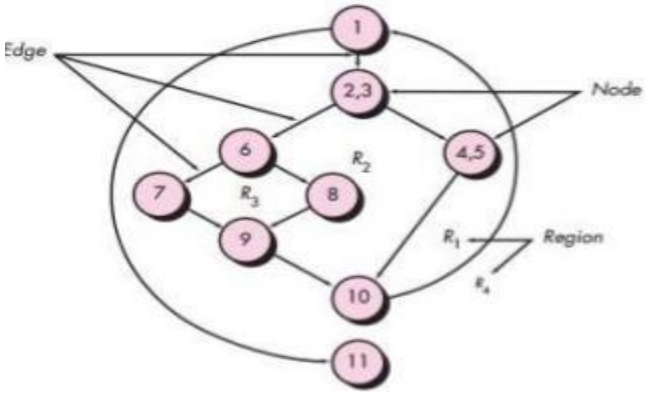
Testing of each statement, object, and function on an individual basis

Basis path testing

- Basis path testing is a white-box testing technique
- The basis path method enables the test-case designer to derive a logical complexity measure of a procedural design and use this measure as a guide for defining a basis set of execution paths
- Flow Graph Notation:
  - a simple notation for the representation of control flow, called a flow graph
  - It also know as program graph



- Arrows called edges or links represent flow of control
- Circles called floe graph nodes represent one or more actions
- Areas bounded by edges and nodes called regions
- A predicate node is a node containing a condition
- Independent program paths:
  - An independent path is any path through the program that introduces at least one new set of processing statements or a new condition
  - independent path must move along at least one edge that has not been traversed before the path is defined
  - Example:



Path 1: 1-11

Path 2: 1-2-3-4-5-10-1-11

Path 3: 1-2-3-6-8-9-10-1-11

Path 4: 1-2-3-6-7-9-10-1-11

d) Explain Guidelines for Formal Technical review in details.

Guidelines for the conduct of formal technical reviews must be established in advance, distributed to all reviewers, agreed upon, and then followed. A review that is uncontrolled can often be worse that no review at all. The following represents a minimum set of guidelines for formal technical reviews:

1. Review the product, not the producer. An FTR involves people and egos. Conducted properly, the FTR should leave all participants with a warm feeling of accomplishment. Conducted improperly, the FTR can take on the aura of an inquisition. Errors should be pointed out gently; the tone of the meeting should be loose and constructive; the intent should not be to embarrass or belittle. The review leader should conduct the review meeting to ensure that the proper tone and attitude are maintained and should immediately halt a review that has gotten out of control.

2. Set an agenda and maintain it. One of the key maladies of meetings of all types is drift. An FTR must be kept on track and on schedule. The review leader is chartered with the responsibility for maintaining the meeting schedule and should not be afraid to nudge people when drift sets in.

3. Limit debate and rebuttal. When an issue is raised by a reviewer, there may not be universal agreement on its impact. Rather than spending time debating the question, the issue should be recorded for further discussion off-line.

4. Enunciate problem areas, but don't attempt to solve every problem noted. A review is not a problem-solving session. The solution of a problem can often be accomplished by the producer alone or with the help of only one other individual. Problem solving should be postponed until after the review meeting.

5. Take written notes. It is sometimes a good idea for the recorder to make notes on a wall board, so that wording and priorities can be assessed by other reviewers as information is recorded.

6. Limit the number of participants and insist upon advance preparation. Two heads are better than one, but 14 are not necessarily better than 4. Keep the number of people involved to the necessary minimum. However, all review team members must prepare in advance. Written comments should be solicited by the review leader (providing an indication that the reviewer has reviewed the material).

7. Develop a checklist for each product that is likely to be reviewed. A checklist helps the review leader to structure the FTR meeting and helps each reviewer to focus on important issues. Checklists should be developed for analysis, design, code, and even test documents.

8. Allocate resources and schedule time for FTRs. For reviews to be effective, they should be scheduled as a task during the software engineering process. In addition, time should be scheduled for the inevitable modifications that will occur as the result of an FTR.

9. Conduct meaningful training for all reviewers. To be effective all review participants should receive some formal training. The training should stress both process-related issues and the human psychological side of reviews. Freedman and Weinberg estimate a one-month learning curve for every 20 people who are to participate effectively in reviews.

10. Review your early reviews. Debriefing can be beneficial in uncovering problems with the review process itself. The very first product to be reviewed should be the review guidelines themselves. Because many variables (e.g., number of participants, type of work products, timing and length, specific review approach) have an impact on a successful review, a software organization should experiment to determine what approach works best in a local context. Porter and his colleagues provide excellent guidance for this type of experimentation.

e) Explain control structural testing in details.

1. Condition Testing

Condition testing is a test construction method that focuses on exercising the logical conditions in a program module.

Errors in conditions can be due to:

- Boolean operator error
- Boolean variable error
- Boolean parenthesis error
- Relational operator error
- Arithmetic expression error

definition: "For a compound condition C, the true and false branches of C and every simple condition in C need to be executed at least once."

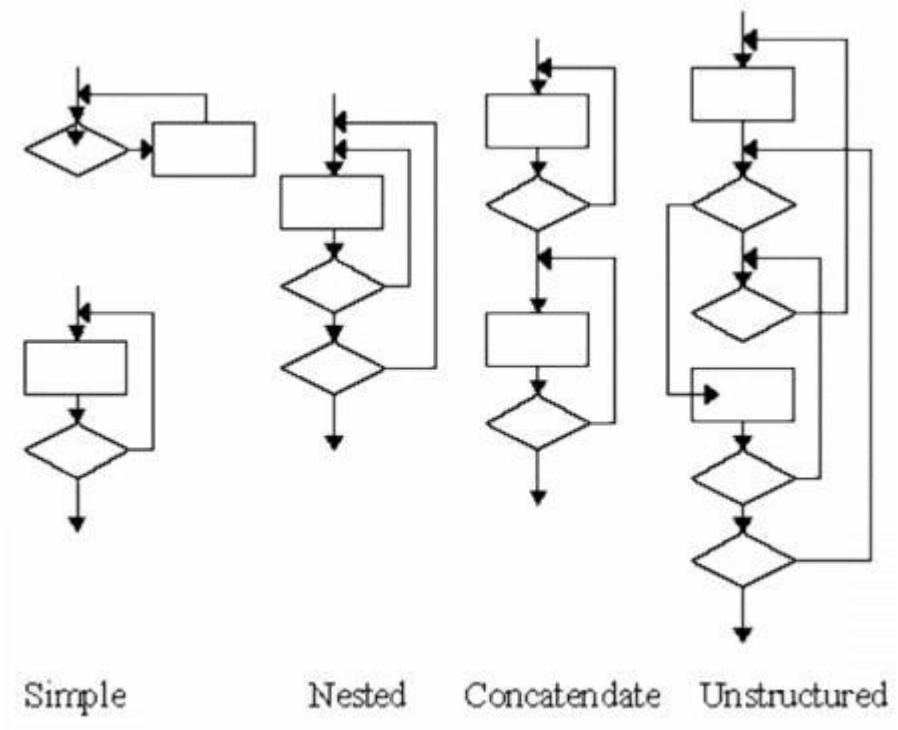
Multiple-condition testing requires that all true-false combinations of simple conditions be exercised at least once.
Therefore, all statements, branches, and conditions are necessarily covered.

2. Data Flow Testing

Selects test paths according to the location of definitions and use of variables. This is a somewhat advanced technique and is not practical for extensive use. Its use should be targeted to modules with nested if and loop statements.

3. Loop Testing

Loops are fundamental to many algorithms and need thorough testing.
There are four different classes of loops: simple, concatenated, nested, and unstructured.
Create a set of tests that force the following situations:



Simple Loops

1. Where n is the maximum number of allowable passes through the loop.

2. Skip loop entirely

3. Only one pass through loop Two passes through loop

4. m passes through loop where m<n.< p="">

5. (n-1), n, and (n+1) passes through the loop.

Nested Loops:-

1. Start with inner loop. Set all other loops to minimum values.
2. Conduct simple loop testing on inner loop.
3. Work outwards
4. Continue until all loops tested.

Concatenated Loops:-

1. If independent loops, use simple loop testing.

2. If dependent, treat as nested loops.

Unstructured loops

1. Don't test - redesign.

Q. 4 Attempt any three of the following (5 Marks each) **15**

a) Explain the black-Box testing in details.

Black box testing

- Black-box testing, also called behavioral testing

- Black-box testing attempts to find errors in the following categories
  - incorrect or missing functions
  - interface errors
  - errors in data structures or external database access
  - behavior or performance errors
  - initialization and termination errors.

Equivalence Partitioning

- Equivalence partitioning is a black-box testing method that divides the input domain of a program into classes of data from which test cases can be derived
- equivalence classes for an input condition. Using concepts introduced in the preceding section, if a set of objects can be linked by relationships that are symmetric, transitive, and reflexive, an equivalence class is present

- Equivalence classes may be defined according to the following guidelines:

1. If an input condition specifies a range, one valid and two invalid equivalence classes are defined.
2. If an input condition requires a specific value, one valid and two invalid equivalence classes are defined.
3. If an input condition specifies a member of a set, one valid and one invalid equivalence class are defined.
4. If an input condition is Boolean, one valid and one invalid class are defined.

Boundary Value Analysis

- A greater number of errors occurs at the boundaries of the input domain rather than in the "center." It is for this reason that boundary value analysis (BVA) has been developed as a testing technique
- Boundary value analysis leads to a selection of test cases that exercise bounding values
- Guidelines for BVA are similar in many respects to those provided for equivalence partitioning:
  - If an input condition specifies a range bounded by values a and b, test cases should be designed with values a and b and just above and just below a and b.
  - If an input condition specifies a number of values, test cases should be developed that exercise the minimum and maximum numbers. Values just above and below minimum and maximum are also tested.
  - Apply guidelines 1 and 2 to output conditions
  - If internal program data structures have prescribed boundaries (e.g., a table has a defined limit of 100 entries), be certain to design a test case to exercise the data structure at its boundary.

b) Describe Metrics for design model in details.

Metrics for the Design Model

- Architectural metrics
  - Provide an indication of the quality of the architectural design
- Component-level metrics
  - Measure the complexity of software components and other characteristics that have a bearing on quality
- Interface design metrics
  - Focus primarily on usability
- Specialized object-oriented design metrics
  - Measure characteristics of classes and their communication and collaboration characteristics

c) Explain System testing in details.

System Testing:

Testing which is done on the entire system known as system testing. By this test, we uncover the errors. It ensures that all the system works as expected. We check System performance and functionality to get a quality product. System testing is nothing but testing the system as a whole. This testing checks complete end-to-end scenario as per the customer's point of view.

Functional and Non-Functional tests also done by System testing. All things are done to maintain trust within the development that system is defect free and bug-free. System testing is also intended to test hardware/software requirements specifications

Types of System Testing

A. Recoverability Testing

This testing determines whether operations can be continued after a disaster or after the integrity of the system has been lost.

The best example of this supposes we are downloading one file. And suddenly connection goes off. After resuming connection our downloading starts at where we left. It does not start from starting again.

This used where continuity of the operations is essential

B. Security Testing

Testing which confirms that the program can access to authorized personnel and that authorized personnel can access the functions available to their security level.

This testing makes sure that the system does not allow unauthorized access to data and resources.

The purpose of security testing is to determine, how well a system protects against unauthorized internal or external access or willful damage.

There is the following area where we generally can check for security:

- Authentication
- Authorization
- Data validation
- Transport security
- Data protection
- Session management

C. Stress Testing

1. This testing generally checks the system is going to continue to function when subjected to the large volume of data than expected.
2. Stress testing may contain input transactions, internal tables, communication channels, disk space, etc.
3. Stress testing checks that the system should run as it would in a production environment.
4. It checks the system under extreme conditions.
5. Stress Testing is also known as Endurance Testing.

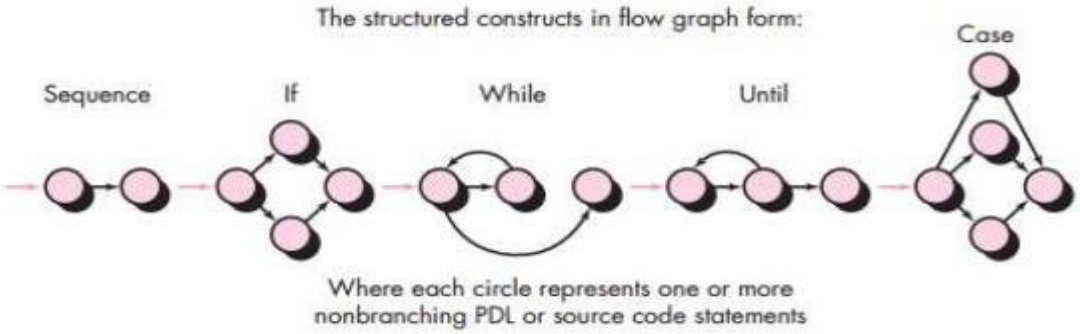
D. Performance Testing

1. This testing makes sure the system's performance under the various condition, in terms of performance characteristics.
2. This testing is also called as compliance testing with respect to performance.
3. This testing ensures that meets the system requirements
4. It checks when multiple users use the same app at a time, then how it responds back

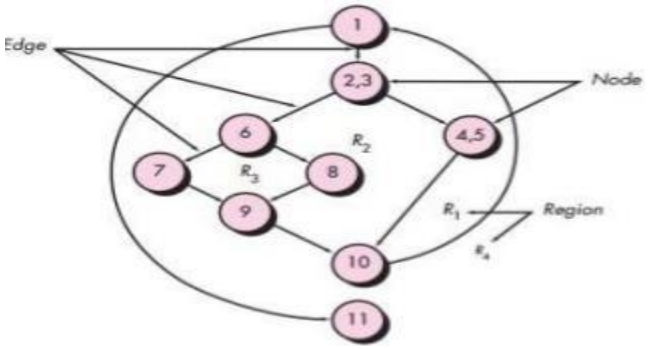
d) Explain Basic Path Testing in details

Basis path testing

- Basis path testing is a white-box testing technique
- The basis path method enables the test-case designer to derive a logical complexity measure of a procedural design and use this measure as a guide for defining a basis set of execution paths
- Flow Graph Notation:
  - a simple notation for the representation of control flow, called a flow graph
  - It also know as program graph



- Arrows called edges or links represent flow of control
- Circles called floe graph nodes represent one or more actions
- Areas bounded by edges and nodes called regions
- A predicate node is a node containing a condition
- Independent program paths:
  - An independent path is any path through the program that introduces at least one new set of processing statements or a new condition
  - independent path must move along at least one edge that has not been traversed before the path is defined
  - Example:



Path 1: 1-11

Path 2: 1-2-3-4-5-10-1-11

Path 3: 1-2-3-6-8-9-10-1-11

Path 4: 1-2-3-6-7-9-10-1-11

e) Explain McCall's Quality factors in details.

McCall software quality model was introduced in 1977. This model is incorporated with many attributes,termed as software factors, which influence a software.

Many organizations around the globe develop and implement different standards to improve the quality needs of their software. This chapter briefly describes some of the widely used standards related to Quality Assurance and Testing.

This model classifies all software requirements into 11 software quality factors

1. Correctness

These requirements deal with the correctness of the output of the software system. They include −

- Output mission
- The required accuracy of output that can be negatively affected by inaccurate data or inaccurate calculations.
- The completeness of the output information, which can be affected by incomplete data.
- The up-to-dateness of the information defined as the time between the event and the response by the software system.
- The availability of the information.
- The standards for coding and documenting the software system.

2. Reliability

Reliability requirements deal with service failure. They determine the maximum allowed failure rate of the software system, and can refer to the entire system or to one or more of its separate functions.

3. Efficiency

It deals with the hardware resources needed to perform the different functions of the software system. It includes processing capabilities (given in MHz), its storage capacity (given in MB or GB) and the data communication capability (given in MBPS or GBPS).

It also deals with the time between recharging of the system's portable units, such as, information system units located in portable computers, or meteorological units placed outdoors.

4. Integrity

This factor deals with the software system security, that is, to prevent access to unauthorized persons, also to distinguish between the group of people to be given read as well as write permit.

5. Usability

Usability requirements deal with the staff resources needed to train a new employee and to operate the software system.

6. Maintainability

This factor considers the efforts that will be needed by users and maintenance personnel to identify the reasons for software failures, to correct the failures, and to verify the success of the corrections

7. Flexibility

This factor deals with the capabilities and efforts required to support adaptive maintenance activities of the software. These include adapting the current software to additional circumstances and customers without changing the software. This factor's requirements also support perfective maintenance activities, such as changes and additions to the software in order to improve its service and to adapt it to changes in the firm's technical or commercial environment.

8. Testability

Testability requirements deal with the testing of the software system as well as with its operation. It includes predefined intermediate results, log files, and also the automatic diagnostics performed by the software system prior to starting the system, to find out whether all components of the system are in working order and to obtain a report about the detected faults. Another type of these requirements deals with automatic diagnostic checks applied by the maintenance technicians to detect the causes of software failures.

9. Portability

Portability requirements tend to the adaptation of a software system to other environments consisting of different hardware, different operating systems, and so forth. The software should be possible to continue using the same basic software in diverse situations.

10. Reusability

This factor deals with the use of software modules originally designed for one project in a new software project currently being developed. They may also enable future projects to make use of a given module or a group of modules of the currently developed software. The reuse of software is expected to save development resources, shorten the development period, and provide higher quality modules.

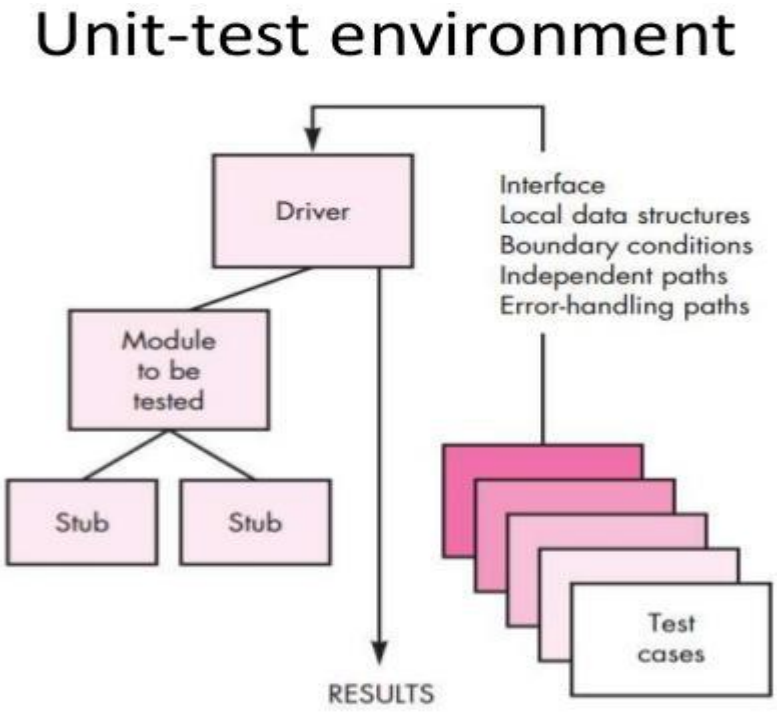
11. Interoperability

Interoperability requirements focus on creating interfaces with other software systems or with other equipment firmware. For example, the firmware of the production machinery and testing equipment interfaces with the production control software.

Q .5 Short notes on any three of the following (5 Marks each) 15

a) Stubs and Drivers

- Driver
  - A simple main program that accepts test case data, passes such data to the component being tested, and prints the returned results
- Stubs
  - Serve to replace modules that are subordinate to (called by) the component to be tested
  - It uses the module's exact interface, may do minimal data manipulation, provides verification of entry, and returns control to the module undergoing testing
- Drivers and stubs both represent testing overhead.
  - Both must be written but don't constitute part of the installed software product



b) Metrics for testing

Metrics for Testing

- Statement and branch coverage metrics
  - Lead to the design of test cases that provide program coverage
- Defect-related metrics
  - Focus on defects (i.e., bugs) found, rather than on the tests themselves
- Testing effectiveness metrics
  - Provide a real-time indication of the effectiveness of tests that have been conducted
- In-process metrics
  - Process related metrics that can be determined as testing is conducted

c) Explain the Art of Debugging.

Debugging happens as a result of testing. When a test case uncovers an error, debugging is the process that causes the removal of that error.

The Debugging Process

Debugging is not testing, but always happens as a response of testing. The debugging process will have one of two outcomes:

(1) The cause will be found, corrected and removed, or

(2)The cause will not be found. Why is debugging difficult?

▶ The symptom and the cause are geographically remote.

▶ The symptom may disappear when another error is corrected.

▶ The symptom may actually be the result of nonerrors (eg round off in accuracies).

▶ The symptom may be caused by a human error that is not easy to find.

▶ The symptom may be intermittent.

▶ The symptom might be due to the causes that are distributed across various tasks on diverse processes.

1) Psychological Considerations

There is evidence that debugging is an innate human trait. Some people are good at it and others not. Although experimental evidence on debugging can be considered in many ways large variations in debugging ability has been identified in software engineering of the same experience.

2) Debugging Approaches

Regardless of the approach that is used, debugging has one main aim: to determine and correct errors. The aim is achieved by using systematic evaluation, intuition, and good fortune. In general three kinds of debugging approaches have been put forward: Brute force, Back tracking and Cause elimination.

Brute force is probably the most popular despite being the least successful. We apply brute force debugging methods when all else fails. Using a “let the computer find the error” technique, memory dumps are taken, run-time traces are invoked, and the program is loaded with WRITE statements. Backtracking is a common debugging method that can be used successfully in small programs. Beginning at the site where a symptom has been uncovered, the source code is traced backwards till the error is found. In cause elimination a list of possible causes of an error are identified and tests are conducted until each one is eliminated.

d) Software Reliability

Software Reliability is the probability of failure-free software operation for a specified period of time in a specified environment. Software Reliability is also an important factor affecting system reliability. It differs from hardware reliability in that it reflects the design perfection, rather than manufacturing perfection. The high complexity of software is the major contributing factor of Software Reliability problems

Measures of reliability & availability

Reliability metrics are used to quantitatively express the reliability of the software product. The option of which metric is to be used depends upon the type of system to which it applies & the requirements of the application domain. Some reliability metrics which can be used to quantify the reliability of the software product are as follows:

1. Mean Time to Failure (MTTF)

MTTF is described as the time interval between the two successive failures. An **MTTF** of 200 mean that one failure can be expected each 200-time units. The time units are entirely dependent on the system & it can even be stated in the number of transactions. **MTTF** is consistent for systems with large transactions.

For example, It is suitable for computer-aided design systems where a designer will work on a design for several hours as well as for Word-processor systems.

To measure **MTTF**, we can evidence the failure data for n failures. Let the failures appear at the time instants $t_1, t_2 ..... t_n$.

2. Mean Time to Repair (MTTR)

Once failure occurs, some-time is required to fix the error. **MTTR** measures the average time it takes to track the errors causing the failure and to fix them.

3. Mean Time Between Failure (MTBR)

We can merge **MTTF** & **MTTR** metrics to get the MTBF metric.

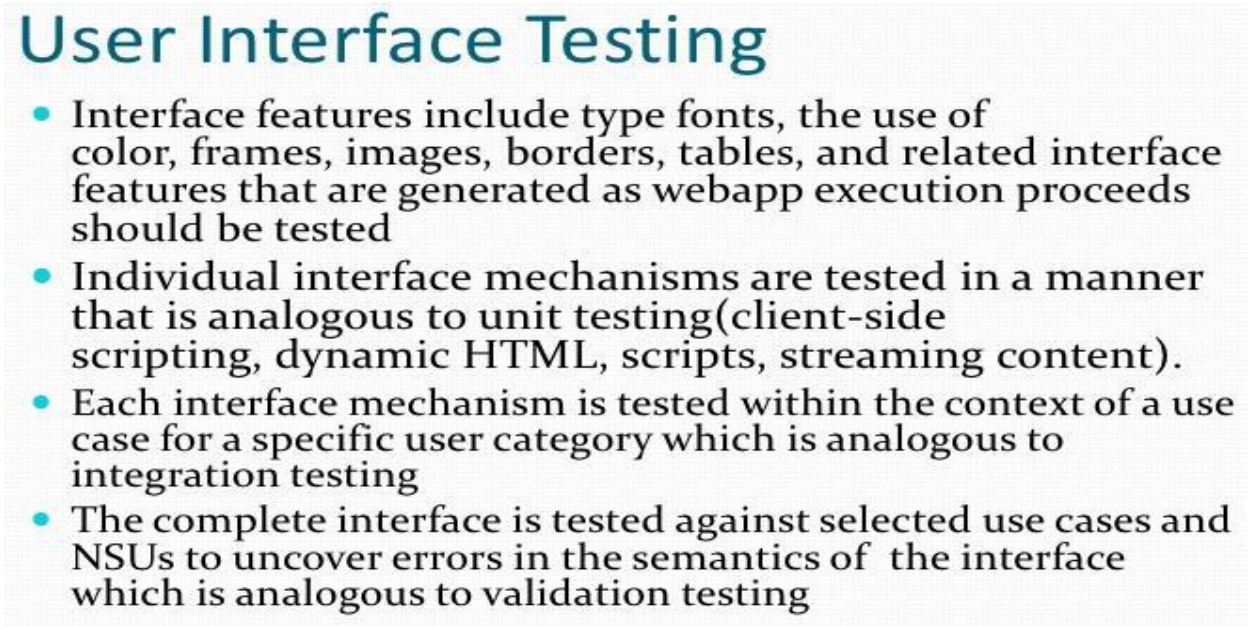
MTBF = MTTF + MTTR

Thus, an **MTBF** of 300 denoted that once the failure appears, the next failure is expected to appear only after 300 hours. In this method, the time measurements are real-time & not the execution time as in **MTTF**.

Software Availability: Software availability is the probability that a program is operating according to requirements at a given point of time and is defined as:

Availability = [MTTF / (MTTF + MTTR)] * 100%

e) User interface Testing for Web Applications



Interface mechanisms

When a user interacts with a webapp, the interaction occurs through one or more interface mechanisms.

Links:

Each navigation link is tested to ensure that the proper content object or function is reached.

Forms:

- At a macroscopic level, tests are performed to ensure that Labels correctly identify fields within the form and that mandatory fields are identified visually for the user
- The server receives all information contained within the form and that no data are lost in the transmission between client and server
- Appropriate defaults are used when the user does not select from a pull-down menu or set of buttons
- Browser functions(e.g., back arrow) do not corrupt data entered in a form

******