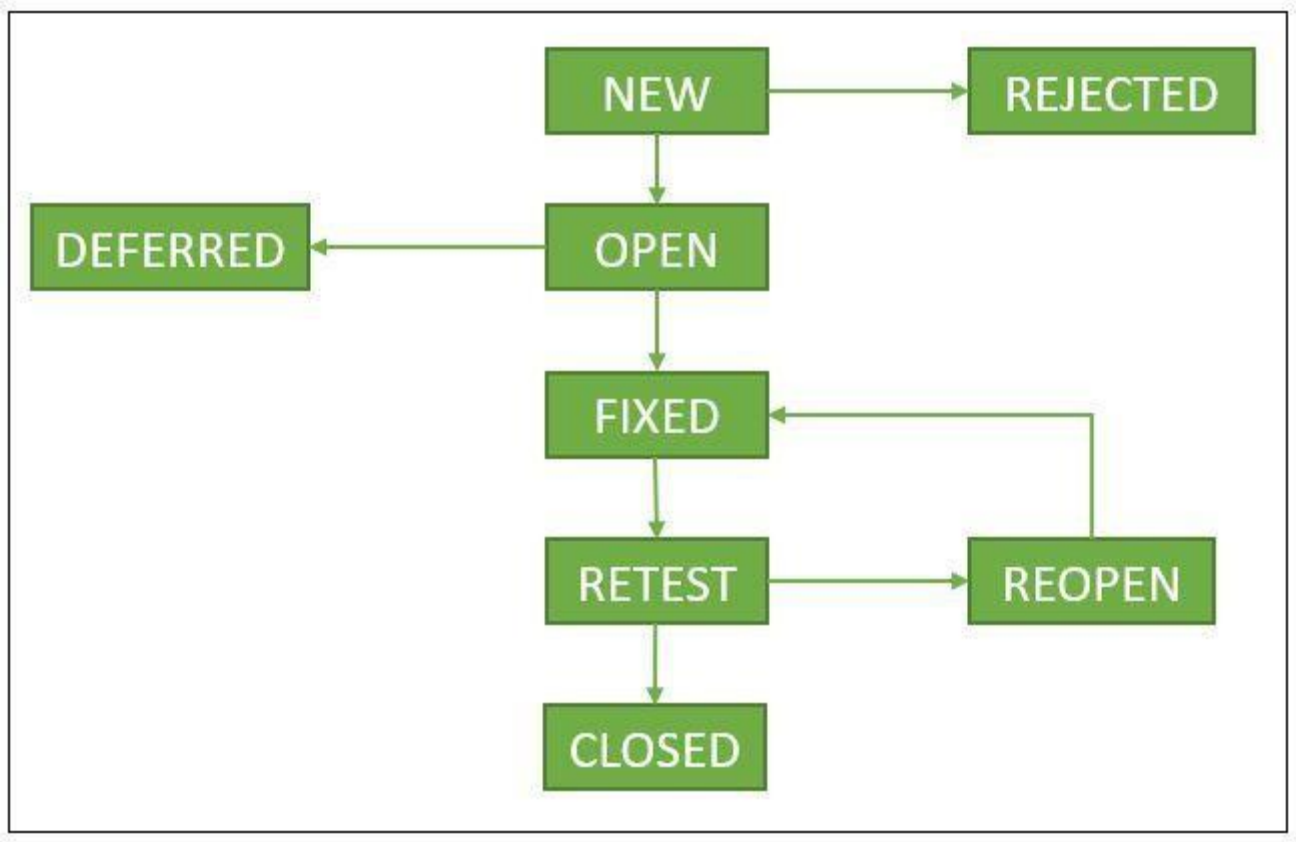
Software Testing – 2 Marks

SOFTWARE TESTING - 2 MARKS

1. What is bug prevention?

Preventative methods are used to maintain software free of flaws. Detection techniques are used to detect software bugs. Because software defects are detected after they exist, detection is more costly than prevention. Let's look at the many types of software flaws, their causes, and how to prevent them.



For example, the process of determining Why a problem has occurred in software is known as Root Cause Analysis (RCA). Root cause analysis is used in many industries so as in the software development world and it ensures bugs prevention in software.

2. What is the goal of software testing?

The main goal of software testing is to find bugs as early as possible and fix bugs and make sure that the software is bug-free. It is a systematic and disciplined approach that aims to identify and rectify defects, errors, and potential issues within the software during its development lifecycle. The primary goals of software testing are to validate the software's functionality, enhance its performance, and improve the overall user experience.

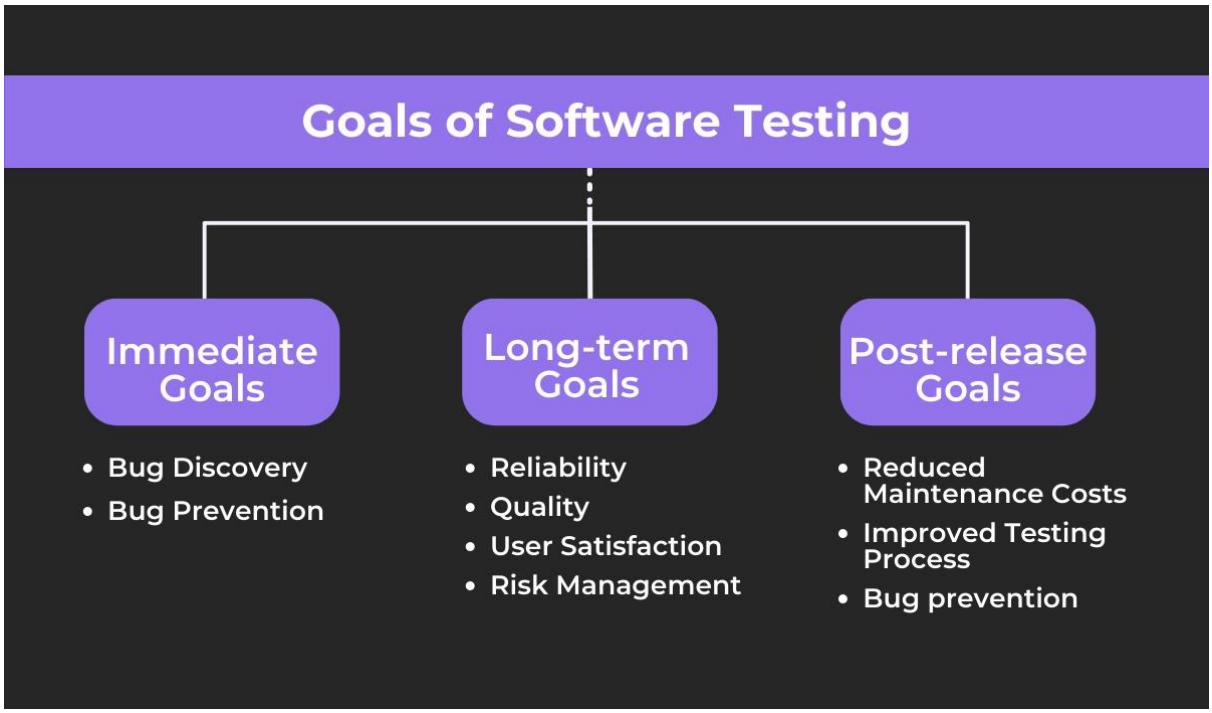
Important Goals of Software Testing:

Software Testing – 2 Marks

- Detecting bugs as soon as feasible in any situation.
- Avoiding errors in a project's and product's final versions.
- Inspect to see whether the customer requirements criterion has been satisfied.
- Last but not least, the primary purpose of testing is to gauge the project and product level of quality.

The goals of software testing may be classified into three major categories as follows:

1. **Immediate Goals**
2. **Long-term Goals**
3. **Post-Implementation Goals**



3. Name the types of bugs.

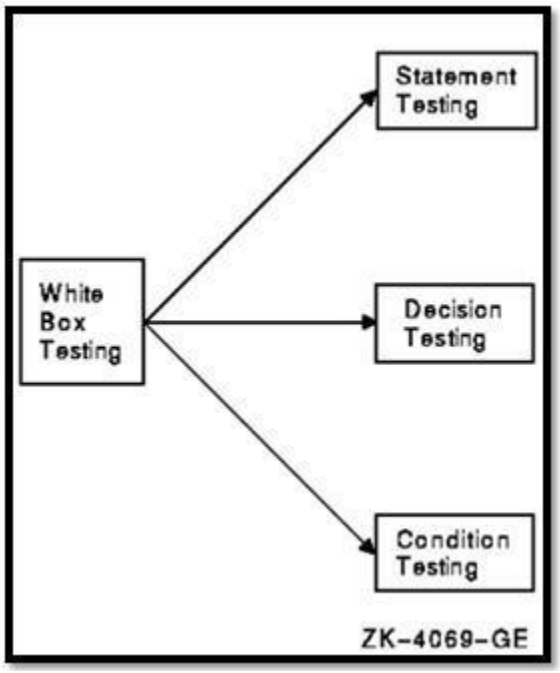
A bug is an unexpected problem with software or hardware. Typical problems are often the result of external interference with the program's performance that was not anticipated by the developer. Minor bugs can cause small problems like frozen screens or unexplained error messages that do not significantly affect usage.

- Functional Bugs.
- Logical Bugs.
- Workflow Bugs.
- Unit Level Bugs.
- System-Level Integration Bugs.
- Out of Bound Bugs.
- Security Bugs.

Software Testing – 2 Marks

4. Define transaction.

A transaction is a unit of work seen from a system user's point of view. A transaction consists of a sequence of operations, some of which are performed by a system, persons or devices that are outside of the system. Transaction begin with Birth-that is they are created as a result of some external act.



5. What is a flow of graphs?

A flow graph in software testing is a graphical representation of the logical flow or control flow of a program. It is used to analyze and design test cases for a given software application. The flow graph helps testers understand the sequence of execution of different statements, branches, and loops within a program.

Here are some key elements in a flow graph:

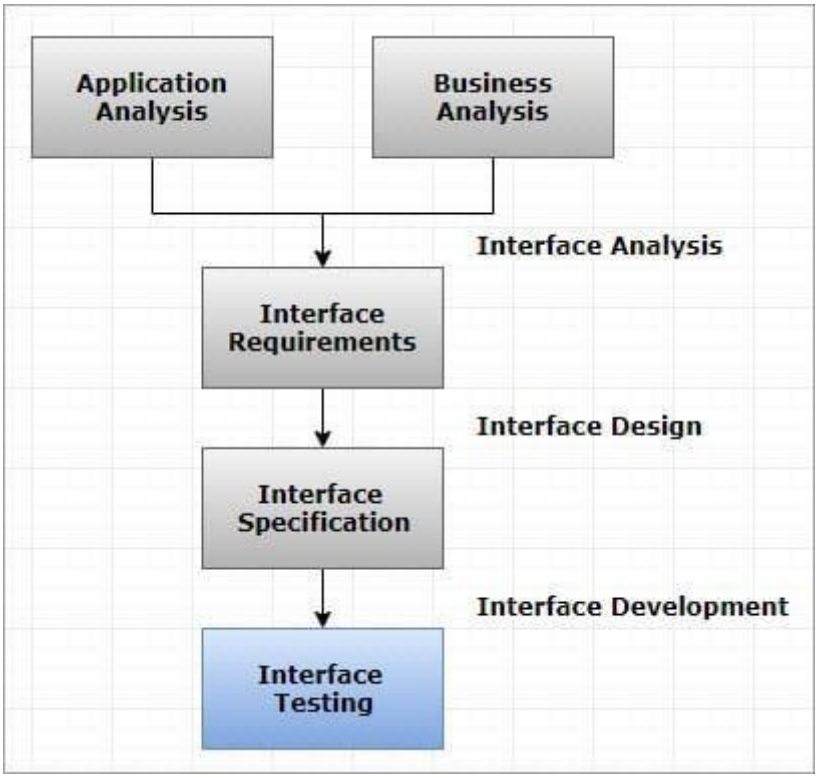
1. **Nodes:** Represent individual statements or blocks of code.
2. **Edges:** Represent the flow of control between nodes. They show the order of execution.
3. **Decision Nodes:** Represent points in the program where a decision is made, typically :ional statements (e.g., if statements). The decision nodes create branches in the flow.
4. **Merge Nodes:** epresent points where different paths of execution come together.

6. What is Interface testing?

Software Testing – 2 Marks

Interface testing is a sort of software testing that confirms the proper connectivity between two separate software systems.

An interface is a link that connects two components. In the computer world, this interface might be anything from APIs to web services. Interface testing is the process of evaluating these connected services or interfaces.



An interface is a software program that contains a collection of instructions, communications, and other properties that allow a device and a user to communicate with one another.

Example of Interface Testing

Assume that the interface for every XYZ application accepts an XML file as input and returns a JSON file as output. All that is required to test the interface of this application are the parameters of the XML file format and the JSON file format.

7. What is the weakness of linguist metrics?

It seems like there might be a slight confusion in your question. It's possible that you meant "linguistic metrics" instead of "linguist metrics." Assuming that's the case, linguistic metrics refer to measurements or quantitative assessments used to analyze various aspects of language, such as readability, complexity, or similarity.

The weakness of linguistic metrics can depend on the specific metric and the context in which it is applied. Here are some general considerations:

Software Testing – 2 Marks

1. **Context Sensitivity:** Linguistic metrics may not always account for the context in which language is used. For example, a metric measuring sentence complexity might not consider the appropriateness of that complexity in a given situation.
2. **Subjectivity:** Some linguistic properties, like "clarity" or "conciseness," can be subjective. Different people may have different opinions on what constitutes clear or concise language.
3. **Domain-Specific Limitations:** Linguistic metrics may not be equally effective across different domains. A metric designed for technical writing might not be as relevant or accurate when applied to creative writing or legal documents.
4. **Semantic Understanding:** Many linguistic metrics focus on surface-level features of language and may not capture the deeper nuances of meaning. They might miss the subtleties of sarcasm, irony, or other forms of figurative language.
5. **Dynamic Language Use:** Language is dynamic and evolves over time. Linguistic metrics may become outdated or less relevant as language usage changes.

8. What is stable state?

It seems like there might be a slight confusion in your terminology. The term "stable state" is not a standard term in computer science or related fields. However, I can provide explanations for two related concepts that might be relevant to your question: "stable state" and "steady state."

1. **Stable State:**
   - In a general sense, a stable state refers to a condition or situation that is resistant to change and tends to remain constant over time. This can apply to various systems, not necessarily limited to computer science.
2. **Steady State:**
   - In the context of computer science, particularly in algorithms or systems, a "steady state" refers to a condition where the system has reached a balanced state and remains in that state unless perturbed by external factors. This concept is often used in the analysis of algorithms, simulations, and system behavior.

9. What are the components of decision tables?

Decision tables are a tabular method used in software testing and business rule management to represent complex decision-making logic. A decision table typically consists of several components, which help organize and document the rules and conditions of a decision-making process. The main components of a decision table include:

1. **Conditions:**
   - Also known as input conditions or predicates, these are the factors or variables that influence the decision. Conditions represent the various states or values that can be associated with each input.

2. **Actions:**
   - Also known as output actions, these are the results or outcomes of the decision based on the specified conditions. Actions represent the different actions or values associated with each possible combination of conditions.
3. **Rules:**
   - Each row in a decision table represents a unique combination of conditions and the corresponding actions. These rules define the logic of the decision-making process. A rule is a specific condition-action pair.

10. What is state graph?

A state graph, also known as a state diagram or state machine diagram, is a visual representation used in software engineering and systems design to illustrate the different states that a system or entity can go through and the transitions between those states. State graphs are particularly useful for modeling the behavior of systems that can exist in different states and undergo state transitions based on certain events or conditions.

Here are key components of a state graph:

1. **States:**
   - States represent the different conditions or modes that a system can be in. Each state represents a specific configuration or situation of the system.
2. **Transitions:**
   - Transitions depict the change from one state to another and are triggered by events or conditions. Events are external stimuli or triggers that cause the system to move from one state to another.
3. **Events:**
   - Events are occurrences or triggers that can initiate state transitions. These can include user actions, system inputs, or changes in external conditions.

11. What is transition testing?

Transition testing is a software testing technique that focuses on verifying the correct behavior of a system as it transitions from one state to another. This type of testing is particularly relevant in systems that exhibit different states during their operation, such as finite state machines or systems with stateful behavior.

In transition testing, the goal is to ensure that the system moves smoothly and correctly from one state to another, responding appropriately to transitions triggered by various events or conditions. The primary objectives of transition testing include:

1. **Validating State Transitions**:
   - Verify that the system transitions from one state to another as expected when specific events or conditions occur.
2. Checking **State Consistency:**
   - Ensure that the system maintains consistency within a given state and that the transition to a new state does not result in unexpected behavior or data corruption.
3. **Testing Event Handling:**
   - Verify that the system correctly handles events or inputs that trigger state transitions. This involves checking the response of the system when certain events occur.

12. Write down any two principles of state testing.

State testing, also known as state-based testing, is a software testing technique that focuses on testing the behavior of a system in different states. Here are two principles associated with state testing:

1. **Coverage of States:**
   - The principle of coverage of states emphasizes the importance of testing the system in all possible states it can assume. This includes testing the system when it is in its initial state, as well as testing all intermediate states and the final state. By covering the entire state space, testers can ensure that the system behaves correctly and consistently across different conditions.
2. **Testing Transitions Between States:**
   - This principle highlights the need to test the transitions between different states. It involves verifying that the system correctly moves from one state to another in response to specific events or conditions. Testing transitions ensures that the system responds appropriately to changes in its environment and that the transition logic is implemented correctly.

1. What is debugging?

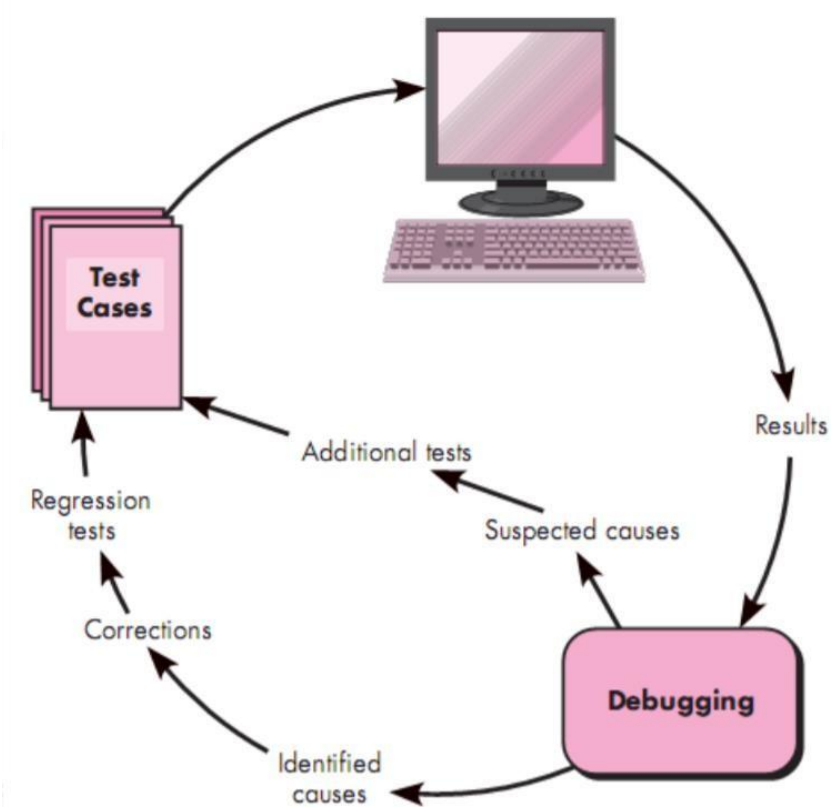
Debugging is the process of identifying and resolving errors, or bugs, in a software system. It is an important aspect of software engineering because bugs can cause a software system to malfunction, and can lead to poor performance or incorrect results. Debugging can be a time-consuming and complex task, but it is essential for ensuring that a software system is functioning correctly.

There are several common methods and techniques used in debugging, including:

1. **Code Inspection:** This involves manually reviewing the source code of a software system to identify potential bugs or errors.

Software Testing – 2 Marks

2. **Debugging Tools:** There are various tools available for debugging such as debuggers, trace tools, and profilers that can be used to identify and resolve bugs.
3. **Unit Testing:** This involves testing individual units or components of a software system to identify bugs or errors.



4. **Integration Testing:** This involves testing the interactions between different components of a software system to identify bugs or errors.
5. **System Testing:** This involves testing the entire software system to identify bugs or errors.
6. **Monitoring**: This involves monitoring a software system for unusual behavior or performance issues that can indicate the presence of bugs or errors.
7. **Logging**: This involves recording events and messages related to the software system, which can be used to identify bugs or errors.

2. What is meant by a bug?

A malfunction in the software/system is an error that may cause components or the system to fail to perform its required functions. In other words, if an error is encountered during the test it can cause malfunction. For example, incorrect data description, statements, input data, design, etc.

1. **Acceptance Testing:** Ensuring that the whole system works as intended.

Software Testing – 2 Marks

2. **Integration Testing:** Ensuring that software components or functions work together.
3. **Unit Testing:** To ensure that each software unit is operating as expected. The unit is a testable component of the application.
4. **Functional Testing:** Evaluating activities by imitating business conditions, based on operational requirements. Checking the black box is a common way to confirm tasks.
5. **Performance Testing:** A test of how the software works under various operating loads. Load testing, for example, is used to assess performance under real-life load conditions.
6. **Re-Testing:** To test whether new features are broken or degraded. Hygiene checks can be used to verify menus, functions, and commands at the highest level when there is no time for a full reversal test.

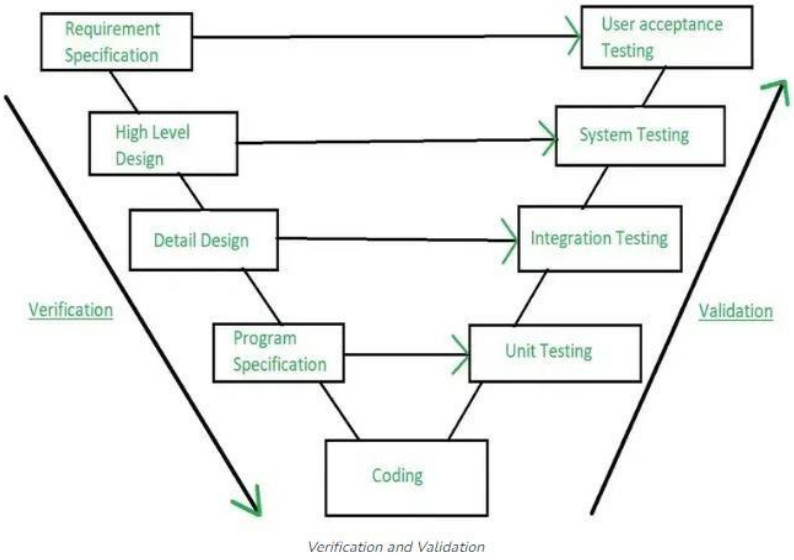
3. Define : Verification.

Verification is the process of checking that software achieves its goal without any bugs. It is the process to ensure whether the product that is developed is right or not. It verifies whether the developed product fulfills the requirements that we have. Verification is simply known as **Static Testing.**

Static Testing

Verification Testing is known as Static Testing and it can be simply termed as checking whether we are developing the right product or not and also whether our software is fulfilling the customer's requirement or not. Here are some of the activities that are involved in verification.

- Inspections
- Reviews
- Walkthroughs
- Desk-checking



Verification and Validation

Validation

Software Testing – 2 Marks

Validation is the process of checking whether the software product is up to the mark or in other words product has high-level requirements. It is the process of checking the validation of the product i.e. it checks what we are developing is the right product. it is a validation of actual and expected products. Validation is simply known as **Dynamic Testing.**

Dynamic Testing

Validation Testing is known as Dynamic Testing in which we examine whether we have developed the product right or not and also about the business needs of the client. Here are some of the activities that are involved in Validation.

1. Black Box Testing
2. White Box Testing
3. Unit Testing
4. Integration Testing

Note: Verification is followed by Validation.

Validation" width="inherit" height="inherit">

5. **How data flow is measured?**

The measurement of data flow in a computing system can be approached in different ways depending on the context and the specific aspects of data flow that you are interested in. Here are a few common methods and metrics used to measure data flow:

1. **Data Flow Diagrams (DFDs):**

- DFDs are graphical representations that illustrate the flow of data within a system. They use symbols to represent processes, data stores, data flow, and external entities. By creating and analyzing DFDs, you can gain insights into how data moves through a system and identify potential bottlenecks or areas for optimization.

2. **Data Flow Metrics:**

- Various metrics can be used to quantitatively measure aspects of data flow, such as data volume, data velocity, and data variety. These metrics are often employed in the context of big data systems, where the efficient movement and processing of large volumes of data are crucial.

3. **Throughput and Bandwidth:**

- Throughput is a measure of the amount of data that can be transferred through a system within a given time frame. Bandwidth refers to the capacity of a communication channel to transmit data. Measuring throughput and bandwidth helps assess the efficiency of data flow in networks and communication systems.

4. **Latency:**

- Latency measures the time delay between the initiation of a data transfer and the actual reception or processing of the data. Low latency is desirable in systems where real-time or near-real-time data flow is crucial, such as in communication networks or streaming applications.

5. **Queue Length:**

- In systems with queues, such as message queues or data processing pipelines, measuring the length of the queue can provide insights into how efficiently data is flowing through the system. Long queues may indicate bottlenecks or resource constraints.

6. Define metrics.

Software testing metrics are quantifiable indicators of the software testing process progress, quality, productivity, and overall health. The purpose of software testing metrics is to increase the efficiency and effectiveness of the software testing process while also assisting in making better decisions for future testing by providing accurate data about the testing process. A metric expresses the degree to which a system, system component, or process possesses a certain attribute in numerical terms. A weekly mileage of an automobile compared to its ideal mileage specified by the manufacturer is an excellent illustration of metrics. Here, we discuss the following points:

1. **Importance of Metrics in Software Testing.**
2. **Types of Software Testing Metrics.**
3. **Manual Test Metrics: What Are They and How Do They Work?**
4. **Other Important Metrics.**
5. **Test Metrics Life Cycle.**
6. **Formula for Test Metrics.**
7. **Example of Software Test Metrics Calculation**

7. Name any four software testing tools.

1. **TestComplete:** TestComplete developed by SmartBear Software is a functional automated testing tool that ensures the quality of the application without sacrificing quality or agility.
2. **LambdaTest:** LambdaTest is a cross-browser testing tool that helps to evaluate how web application responds when accessed through a variety of different browsers.

3. **TestRail:** TestRail is a test management tool that helps to streamline software testing processes, get visibility into QA. This tool is used by testers, developers, and team leads to manage, track, and organize software testing efforts.

4. Xray: Xray is a test management app for Jira that helps to plan, execute, and track quality assurance with requirements traceability.

5. Zephyr Scale: Zephyr Scale is a test management provides a smarter and more structured way to plan, manage, and measure tests inside Jira.

6. Selenium: Selenium provides a playback tool for authoring tests across most web browsers without the need to learn a test scripting language.

7. Ranorex: Ranorex Studio is a GUI test automation framework used for testing web-based, desktop, and mobile applications. It does not have its own scripting language to automate application.

.

8. TestProject: TestProject is a test automation tool that allows users to create automated tests for mobile and web applications. It is built on top of popular frameworks like Selenium and Appium.

9. Katalon Platform: Katalon Platform is a comprehensive quality management platform that enables team to easily and efficiently test, launch, and optimize the best digital experiences.

10. UFT/QTP: Micro Focus UFT is a software that provides functional and regression tests automation for software applications and environments.

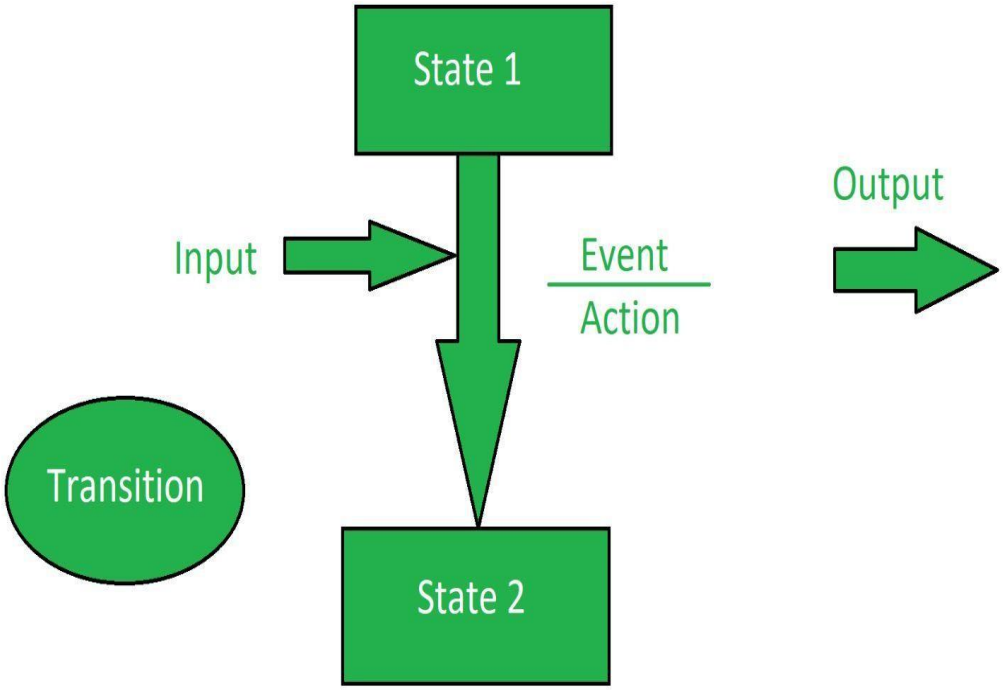
8. Define : States.

State Transition Testing is a type of software testing which is performed to check the change in the state of the application under varying input. The condition of input passed is changed and the change in state is observed.

State Transition Testing is basically a black box testing technique that is carried out to observe the behavior of the system or application for different input conditions passed in a sequence. In this type of testing, both positive and negative input values are provided and the behavior of the system is observed.

State Transition Testing is basically used where different system transitions are needed to be tested.

Software Testing – 2 Marks



Objectives of State Transition Testing:

The objective of State Transition testing is:

- To test the behavior of the system under varying input.
- To test the dependency on the values in the past.
- To test the change in transition state of the application.
- To test the performance of the system.

Transition States:

- **Change Mode:**
  When this mode is activated then the display mode moves from TIME to DATE.
- **Reset:**
  When the display mode is TIME or DATE, then reset mode sets them to ALTER TIME or ALTER DATE respectively.
- **Time Set:**
  When this mode is activated, display mode changes from ALTER TIME to TIME.
- **Date Set:**
  When this mode is activated, display mode changes from ALTER DATE to DATE.

State Transition Diagram:

State Transition Diagram shows how the state of the system changes on certain inputs. It has four main components:

1. States
2. Transition
3. Events
4. Actions

Software Testing – 2 Marks

9. Give any two applications of decision tables.

1**. **Software Testing:****

- Decision tables are widely used in software testing to systematically design and document test cases, especially for systems with complex decision-making logic. The tables help testers consider all possible combinations of input conditions and corresponding actions, ensuring comprehensive test coverage. By representing different scenarios and conditions in a structured tabular format, decision tables assist in identifying and executing test cases that validate the correctness of software under varying conditions.

2. **Business Rules Management:**

- Decision tables find application in managing and implementing business rules, particularly in domains where rules and conditions govern processes. In business rule management systems, decision tables are used to model and represent the rules that guide decision-making within an organization. This facilitates clear documentation, easy modification, and effective communication of business rules among stakeholders. Decision tables provide a visual and structured way to represent complex rule sets, making it easier for business analysts and domain experts to understand and manage business logic.

11. What are decision tables?

A decision table is a brief visual representation for specifying which actions to perform depending on given conditions. The information represented in decision tables can also be represented as decision trees or in a programming language using if-then-else and switchcase statements.

A decision table is a good way to settle with different combination inputs with their corresponding outputs and is also called a cause-effect table. The reason to call causeeffect table is a related logical diagramming technique called cause-effect graphing that is basically used to obtain the decision table.

Decision Table in test designing:

CONDITIONS STEP 1 STEP 2 STEP 3 STEP 4

Condition 1 Y Y N N Condition

2 Y N Y N

Condition 3 Y N N Y Condition

4 N Y Y N

Software Testing – 2 Marks

1. What is the purpose of Debugging?

The purpose of debugging is to identify, isolate, and fix errors, bugs, or defects in a computer program or system. Debugging is a crucial part of the software development process and is performed to ensure that the software functions correctly and meets the specified requirements. The primary goals of debugging include:

1. **Error Identification:**

- The main purpose of debugging is to identify errors or unexpected behaviors in a program. These errors may arise due to coding mistakes, logical flaws, or other issues that prevent the program from working as intended.

2. **Isolation of Issues:**

- Debugging helps developers isolate the source of the problem within the code. By examining the program's behavior, variable values, and execution flow, developers can narrow down the location and cause of the error.

3. **Correction of Defects:**

- Once the root cause of an issue is identified, debugging facilitates the correction of defects. Developers modify the code to eliminate the errors and ensure that the program behaves as expected.

4. **Verification of Fixes:**

- After making changes to the code, debugging is used to verify that the corrections are effective. Developers run the program through the debugger to check if the identified issues have been resolved and if new problems have not been introduced.

5. **Optimization and Performance Improvement:**

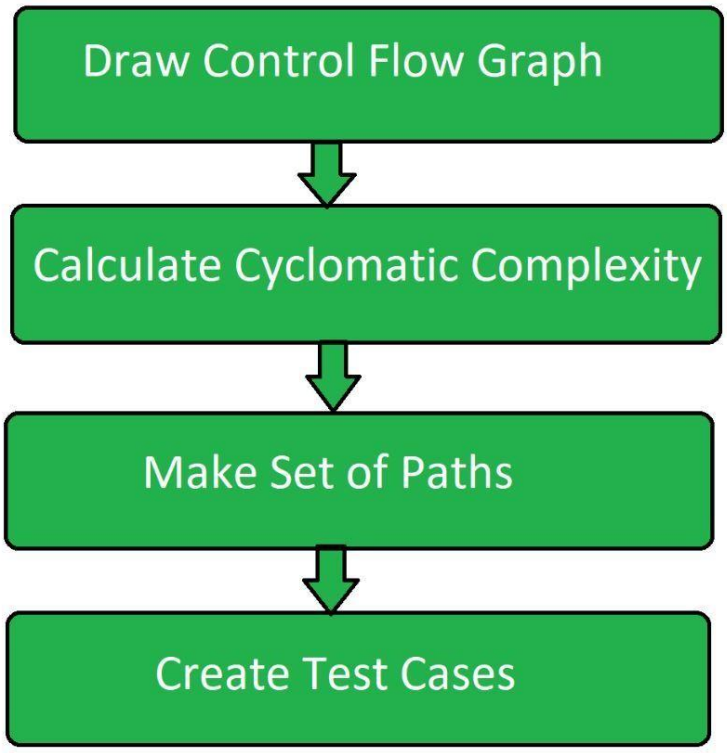
- Debugging is not only about fixing errors but also about optimizing code for better performance. Developers may use debugging tools to analyze the execution of the program, identify bottlenecks, and make improvements to enhance overall efficiency.

3. What is path testing?

Path Testing is a method that is used to design the test cases. In path testing method, the control flow graph of a program is designed to find a set of linearly independent paths of execution. In this method Cyclomatic Complexity is used to determine the number of linearly independent paths and then test cases are generated for each path.
It give complete branch coverage but achieves that without covering all possible paths of the control flow graph. McCabe’s Cyclomatic Complexity is used in path testing. It is a structural

testing method that uses the source code of a program to find every possible executable path.

Path Testing Process:



- **Control Flow Graph:**
  Draw the corresponding control flow graph of the program in which all the executable paths are to be discovered.

- **Cyclomatic Complexity:**

After the generation of the control flow graph, calculate the cyclomatic complexity of the program using the following formula. McCabe's Cyclomatic Complexity = E - N + 2P

Where,

E = Number of edges in control flow graph

N = Number of vertices in control flow graph

P = Program factor

- **Make Set:**

  Make a set of all the path according to the control flow graph and calculated cyclomatic complexity. The cardinality of set is equal to the calculated cyclomatic complexity.
- **Create Test Cases:**
  Create test case for each path of the set obtained in above step.

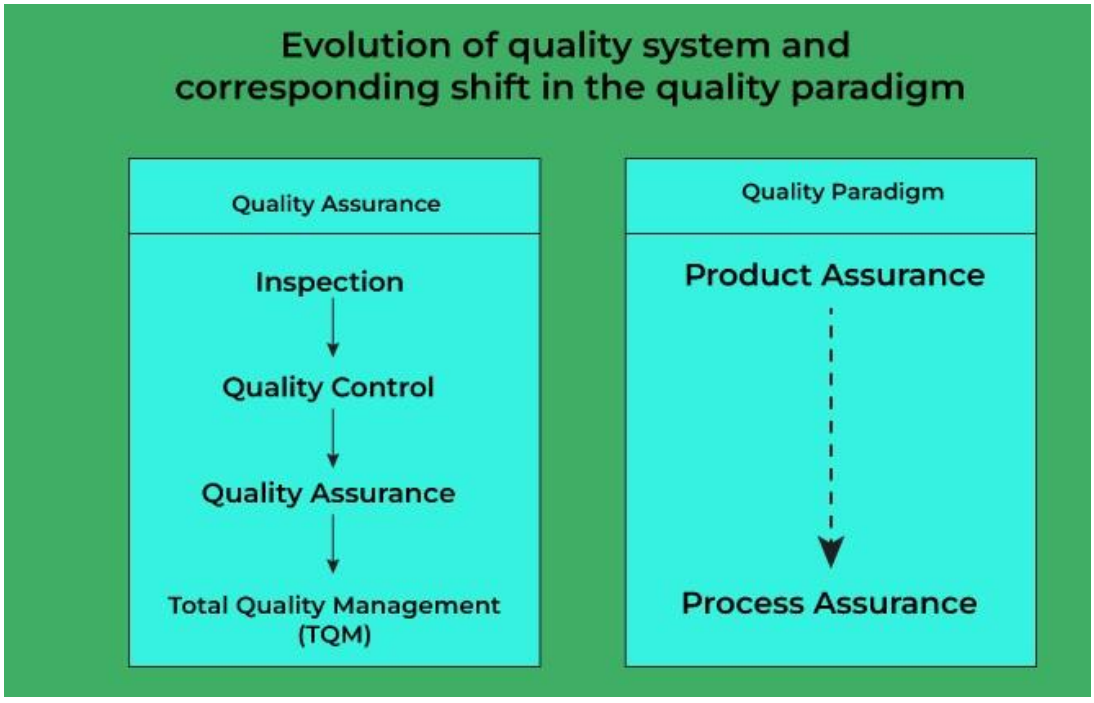
Software Testing – 2 Marks

Path Testing Techniques:

- **Control Flow Graph:**
  The program is converted into control flow graph by representing the code into nodes and edges.
- **Decision to Decision path:**
  The control flow graph can be broken into various Decision to Decision paths and then collapsed into individual nodes.
- **Independent paths:**
  Independent path is a path through a Decision to Decision path graph which cannot be reproduced from other paths by other methods.

4. Define software Quality.

Software quality refers to the degree to which a software product meets specified requirements and fulfills the needs or expectations of its users. It encompasses various attributes and characteristics that contribute to the overall excellence, reliability, and performance of a software application. Achieving high software quality is crucial for ensuring customer satisfaction, minimizing defects, and meeting business objectives.



Factors of Software Quality

The modern read of high-quality associates with software many quality factors like the following:

- **Portability:** A software is claimed to be transportable, if it may be simply created to figure in several package environments, in several machines, with alternative code merchandise, etc.
- **Usability:** A software has smart usability if completely different classes of users (i.e.

knowledgeable and novice users) will simply invoke the functions of the merchandise.

- **Reusability:** A software has smart reusability if completely different modules of the merchandise will simply be reused to develop new merchandise.
- **Correctness:** Software is correct if completely different needs as laid out in the SRS document are properly enforced.
- **Maintainability:** A software is reparable, if errors may be simply corrected as and once they show up, new functions may be simply added to the merchandise, and therefore the functionalities of the merchandise may be simply changed, etc
- **Reliability.** Software is more reliable if it has fewer failures. Since software engineers do not deliberately plan for their software to fail, reliability depends on the number and type of mistakes they make. Designers can improve reliability by ensuring the software is easy to implement and change, by testing it thoroughly, and also by ensuring that if failures occur, the system can handle them or can recover easily.
- **Efficiency.** The more efficient software is, the less it uses of CPU-time, memory, disk space, network bandwidth, and other resources. This is important to customers in order to reduce their costs of running the software, although with today's powerful computers, CPU time, memory and disk usage are less of a concern than in years gone by.

5. Define data flow testing.

Data Flow Testing is a type of structural testing. It is a method that is used to find the test paths of a program according to the locations of definitions and uses of variables in the program. It has nothing to do with data flow diagrams.
It is concerned with:

- Statements where variables receive values,
- Statements where these values are used or referenced.

To illustrate the approach of data flow testing, assume that each statement in the program assigned a unique statement number. For a statement number S-

DEF(S) = {X | statement S contains the definition of X}

USE(S) = {X | statement S contains the use of X}

If a statement is a loop or if condition then its DEF set is empty and USE set is based on the condition of statement s.

Data Flow Testing uses the control flow graph to find the situations that can interrupt the flow of the program.
Reference or define anomalies in the flow of the data are detected at the time of associations between values and variables. These anomalies are:

A variable is defined but not used or referenced,

- A variable is used but never defined,
- A variable is defined twice before it is used

Example:

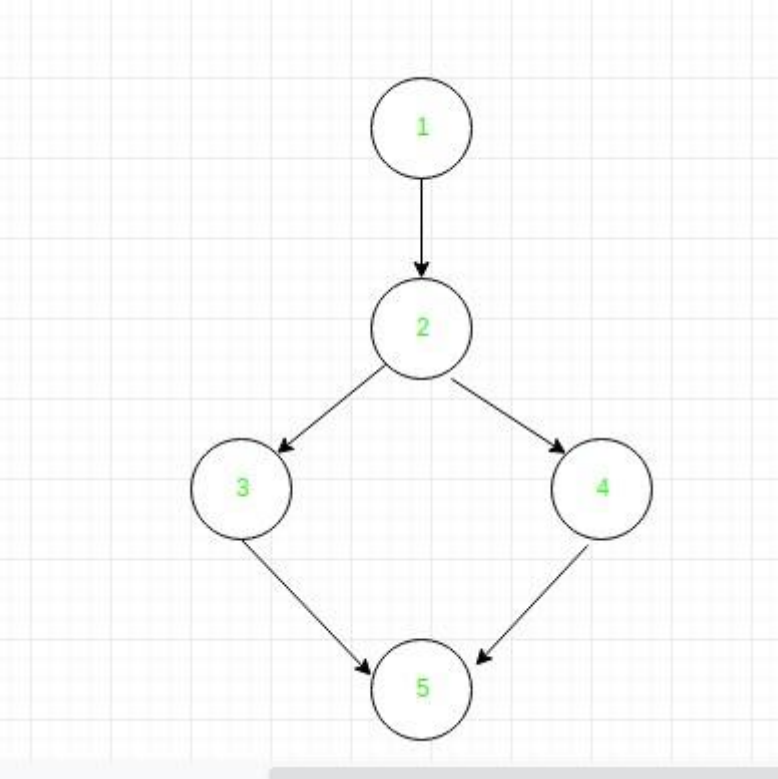
Software Testing – 2 Marks

1. read x, y;
2. if(x>y) **3.** a = x+1 else **4.** a = y-1
5. print a;

6. What is Data flow graph?

A data-flow graph is a collection of arcs and nodes in which the nodes are either places where variables are assigned or used, and the arcs show the relationship between the places where a variable is assigned and where the assigned value is subsequently used.

Control flow graph of above example:



Define/use of variables of above example:

Variable	Defined at node	Used at node
x	1	2, 3
y	1	2, 4
a	3, 4	5

7. What is Domain testing?

Software Testing – 2 Marks

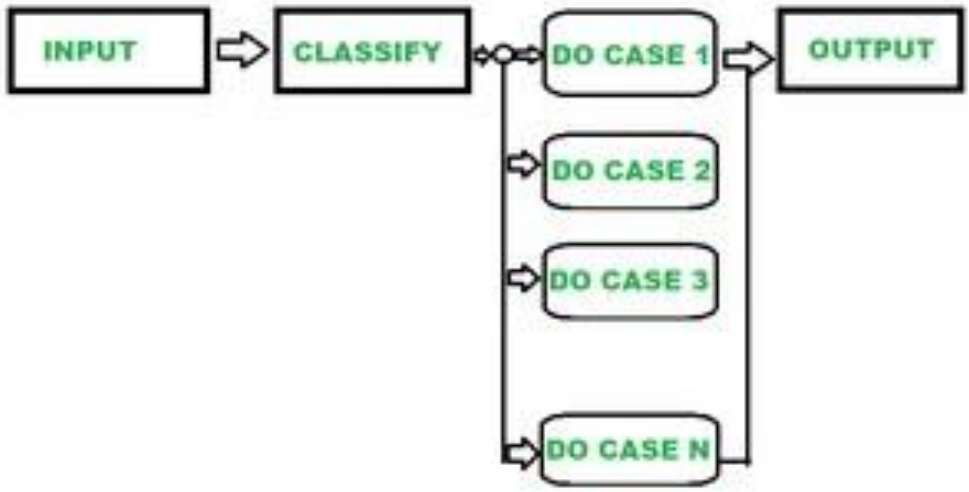
It is a software testing technique where minimum numbers of inputs are used to access appropriate output of a system, to ensure the system does not accept invalid input values. The system is expected to give required outputs blocking the invalid inputs.

A real-lifeon **example of Domain testing :**

Let there be a group of students on a study tour. For entertainment purposes, they have been given a ticket to perform a specific activity based on gender and age inputs. Here the entertainment facility acts as the test, age groups will be boundary values with numerous possible scenarios. Students perform activities in the following manner:

- Children less than 5 years old are to tell a poem
- Boys 5>=10 are to draw
- Girls 5>=10 are to sing a song
- Boys >10 are to compete in a sport
- Girls >10 are to participate in the quiz
- The remaining children >15 are to participate in an essay competition

Based on the given algorithm, the specialist groups the values into classes i.e. age groups, and then boundary values are picked i.e. highest and lowest age values in a group. Then different scenarios are built with expected results for each.



Skills required for Domain testing :

To be a good domain tester, one must have the following skills:

- Quick learner
- Domain knowledge
- Can work under pressure
- Technical and Programming skills
- Automation skill
- Bug hunting skill
- Communication skill

8. Write the objectives of syntax testing.

Software Testing – 2 Marks

Syntax testing, also known as syntax checking or syntactic testing, is a type of testing that focuses on verifying the syntactic structure of a program or code. The primary objectives of syntax testing include:

1. **Verification of Correct Syntax:**

- The fundamental goal of syntax testing is to ensure that the code or program adheres to the rules and structure of the programming language in which it is written. This involves checking for correct syntax, including proper usage of keywords, punctuation, and language-specific constructs.

2. **Identification of Syntax Errors:**

- Syntax testing aims to identify and locate syntax errors within the code. Syntax errors are mistakes in the structure of the program that prevent it from being compiled or interpreted correctly. Common syntax errors include misspelled keywords, missing semicolons, or incorrect use of brackets.

3. **Prevention of Compilation or Interpretation Issues:**

- By detecting and addressing syntax errors early in the development process, syntax testing helps prevent issues that may arise during the compilation or interpretation of the code. Ensuring correct syntax is crucial for generating executable programs.

4. **Support for Automated Tools:**

- Syntax testing facilitates the use of automated tools, such as integrated development environments (IDEs) or code editors, that provide real-time syntax checking. These tools can highlight syntax errors as developers write code, enabling them to make corrections promptly.

5. **Enhancement of Code Readability:**

- Ensuring correct syntax contributes to code readability. Syntax testing helps maintain a consistent and standardized coding style, making it easier for developers to understand and collaborate on the codebase.

9. Define Path Expressions.

A path expression consists of a series of one or more steps that are separated by either a slash character (/) or two slash characters (//). The path expression can begin with a slash character (/), two slash characters(//), or a step. A slash character (/) is used to separate individual steps.

Ex: In concurrency control, path expressions are a mechanism for expressing permitted sequences of execution. For example, a path expression like " {read}, write " might specify

that either multiple simultaneous executions of read or a single execution of write but not both are allowed at any point in time.

11.What do you mean by state Graph?

A state graph, also known as a state diagram or state machine diagram, is a visual representation used in software engineering to depict the different states that a system or entity can transition through during its lifecycle. It is a graphical model that illustrates the dynamic behaviour of a system, showing how it responds to various events, inputs, or conditions.

Key elements of a state graph include:

1. **States:**

- Represent different conditions or modes that the system can exist in. Each state is a distinct configuration of the system.

2. **Transitions:**

- Depict the movement or change from one state to another in response to events or inputs. Transitions are triggered by specific conditions or actions.

3. **Events:**

- External stimuli or triggers that initiate state transitions. Events can include user inputs, system signals, or changes in the environment.

4. **Actions/Activities:**

- Represent the tasks or operations performed when transitioning between states. Actions may include updating variables, triggering processes, or changing the system's behavior.

5. **Initial State:**

- The starting point of the system, representing its initial condition before any events or inputs occur.

6. **Final State:**

- The end point of a process or the termination of the system, indicating that a specific sequence of events or tasks has been completed.

1. What is bug prevention?

Software Testing – 2 Marks

Bug prevention in software testing refers to the set of activities, processes, and practices implemented throughout the software development life cycle to minimize the occurrence of defects, errors, or bugs in the final product. The goal is to identify and address potential issues early in the development process, reducing the likelihood of bugs reaching the later stages of development or the production environment. Bug prevention is a proactive approach that emphasizes quality assurance and aims to improve the overall reliability and stability of the software.

Key strategies for bug prevention include:

1. **Code Reviews:**

- Conducting thorough code reviews allows developers to identify and address issues in the source code before it is integrated into the larger codebase. Peer reviews provide an opportunity to catch coding errors, improve coding standards adherence, and share knowledge among team members.

2. **Coding Standards:**

- Enforcing coding standards helps maintain consistency in the code, making it more readable and less prone to errors. Adhering to established coding guidelines reduces the likelihood of common mistakes and promotes a uniform coding style across the development team.

3. **Static Code Analysis:**

- Utilizing static code analysis tools can automatically scan the source code for potential issues, such as coding rule violations, security vulnerabilities, or performance bottlenecks. These tools provide early feedback to developers, allowing them to address issues before the code is executed.

4. **Unit Testing:**

- Writing and executing unit tests for individual components or modules of the software helps catch defects early in the development process. Unit testing ensures that each unit of code performs as expected and helps maintain the correctness of the codebase.

5. **Requirements Analysis:**

- Conducting thorough requirements analysis helps ensure that the project team has a clear understanding of user expectations and system requirements. Misunderstandings or ambiguities in requirements can lead to errors, and addressing these issues early helps prevent bugs.

3. What are processing Bugs?

Software Testing – 2 Marks

In software engineering, processing bugs refer to defects or errors in a software application that are related to how the program processes data, information, or tasks. These bugs often arise from issues in the algorithmic logic, data manipulation, or overall flow of the program. Processing bugs can impact the correctness, efficiency, or reliability of the software, and they are typically identified during testing or when the software is in use.

Here are some common types of processing bugs:

1. **Logic Errors:**

- Logic errors occur when there is a flaw in the algorithm or decision-making logic of the program. This can lead to incorrect computations, unexpected results, or inaccurate processing of data.

2. **Data Processing Errors:**

- These errors occur when the program mishandles data, such as incorrect calculations, improper data manipulation, or errors in data storage and retrieval. Data processing errors can lead to incorrect outputs or corrupted data.

3. **Algorithmic Errors:**

- Algorithmic errors involve mistakes in the design or implementation of algorithms. This can lead to suboptimal performance, inefficiencies, or incorrect results during the execution of the program.

4. **Concurrency Issues:**

- In multi-threaded or concurrent software, processing bugs may arise due to improper synchronization or coordination between different threads. These bugs can result in race conditions, deadlocks, or other concurrency-related problems.

5. **Resource Management Bugs:**

- Processing bugs related to resource management involve issues with allocating, using, or releasing resources such as memory, file handles, or network connections. Memory leaks, resource contention, or improper handling of resources can lead to instability and performance issues.

6. What is meant by path instrumentation.

In software engineering, processing bugs refer to defects or errors in a software application that are related to how the program processes data, information, or tasks. These bugs often arise from issues in the algorithmic logic, data manipulation, or overall flow of the program.

Processing bugs can impact the correctness, efficiency, or reliability of the software, and they are typically identified during testing or when the software is in use.

Software Testing – 2 Marks

7. List the strategies for data flow testing.

Data flow testing is a white-box testing technique that focuses on the paths through which data is transferred and manipulated within a program. The goal is to identify potential issues related to the flow of data, such as uninitialized variables, data dependencies, and data inconsistencies. Here are some strategies for data flow testing:

1. **All Definitions-Use (All-DU) Pairs:**
   - Identify all variable definitions and uses in the program and create pairs of definitions and corresponding uses. Test cases are designed to ensure that the values assigned to variables at their definitions are correctly used at their corresponding uses.
2. **All-C Use Pairs:**
   - Focus on variable uses in conditional statements (e.g., if conditions, switch statements). Test cases are designed to verify the correct handling of data in different branches of conditional statements.
3. **All-P Use Pairs:**
   - Concentrate on variable uses in loop predicates. Test cases are designed to explore the handling of data within loop structures, ensuring that loops are executed the correct number of times based on the values of variables.

8. Define : Test cases

Software testing is known as a process for validating and verifying the working of a software/application. It makes sure that the software is working without any errors, bugs, or any other issues and gives the expected output to the user. The software testing process isn't limited to finding faults in the present software but also finding measures to upgrade the software in various factors such as efficiency, usability, and accuracy. So, to test software the software testing provides a particular format called a **Test Case**.

9. What is Linguistics Metrics?

It seems there might be a slight confusion in your question. If you're referring to "Linguistic Metrics," it's not a standard term in the context of software development or testing. However, if you're interested in linguistic analysis applied to metrics, there are areas where linguistics and metrics intersect, such as natural language processing (NLP) and sentiment analysis.

On the other hand, if you meant "Linguistic Metrics" in a different context, please provide additional details or clarify, and I'll do my best to assist you.