**Module-1 ( Introduction and Fundamentals)**

1. **Introduction of Software Engineering**

Software Engineering is a discipline that applies engineering principles to the design, development, testing, deployment, and maintenance of software systems. It focuses on creating reliable, efficient, scalable, and user-friendly software through systematic processes, methodologies, and tools. Key aspects include requirement analysis, project management, coding, debugging, and quality assurance, ensuring that software meets user needs and performs effectively in its intended environment.

1. **What is Testing?**

Testing is the process of evaluating a software application to identify and resolve defects, ensure its functionality aligns with requirements, and verify that it performs as expected under specified conditions.

The primary goals of testing are to ensure:

1. **Functionality**: The software behaves as intended.
2. **Quality**: It meets the required standards for reliability, performance, and security.
3. **Usability**: It provides a satisfactory user experience.
4. **Compatibility**: It works across various devices, operating systems, or configurations.

Testing can be manual or automated and is conducted at different levels, such as unit testing, integration testing, system testing, and acceptance testing.

1. **Software Architecture**

Software Architecture refers to the high-level structure of a software system, defining its components, their relationships, and the principles guiding its design and evolution. It acts as a blueprint for both the system and the project, ensuring that the system meets technical and business requirements.

**Key Aspects of Software Architecture:**

1. **Structure**: Organizing components, modules, or layers.
2. **Behavior**: Defining how components interact and function together.
3. **Non-functional Requirements:** Addressing performance, scalability, security, maintainability, and other quality attributes.
4. **Design Principles:** Applying patterns and guidelines for efficient and reusable solutions.

**Common Architectural Styles:**

1. **Layered Architecture:** Divides the system into layers (e.g., presentation, business logic, data).
2. **Microservices Architecture:** Breaks the system into smaller, independent services.
3. **Client-Server Architecture:** Splits the system into client and server components.
4. **Event-Driven Architecture:** Responds to events or changes in state.

A well-designed software architecture helps ensure the system's scalability, maintainability, and robustness.

1. **Testing Activities**

Testing Activities are the tasks and processes involved in verifying and validating a software system to ensure its quality and reliability. These activities span the entire software development lifecycle and aim to identify and fix defects while ensuring the software meets requirements.

**Key Testing Activities:**

1. **Test Planning**
   * Defining the testing objectives and scope.
   * Preparing the test plan, schedule, and resources.
   * Identifying tools, techniques, and the team for testing.
2. **Test Design**
   * Analyzing requirements and specifications to create test scenarios and cases.
   * Designing test data for various input combinations.
   * Setting up test environments.
3. **Test Implementation**
   * Writing and organizing test scripts or cases.
   * Configuring tools for automated testing (if applicable).
   * Preparing the test environment for execution.
4. **Test Execution**
   * Running test cases manually or using automation tools.
   * Logging the results of each test case.
   * Reporting defects or anomalies.
5. **Defect Tracking and Reporting**
   * Documenting identified defects.
   * Assigning defects to developers for resolution.
   * Retesting fixed issues.
6. **Test Monitoring and Control**
   * Tracking the progress of testing against the plan.
   * Adjusting activities based on deviations or challenges.
   * Ensuring all test cases are executed as planned.
7. **Test Evaluation and Reporting**
   * Analyzing test results to determine software quality.
   * Creating test summary reports for stakeholders.
   * Providing recommendations for improvements.
8. **Test Closure Activities**
   * Ensuring all planned tests are completed.
   * Documenting learnings and metrics for future projects.
   * Archiving test artifacts for reference.

Each activity contributes to delivering a high-quality software product that meets user expectations.

1. **Software Development Life Cycle (SDLC)**

The Software Development Life Cycle (SDLC) is a structured process for developing software applications, ensuring high quality, efficiency, and alignment with user requirements. It defines a series of stages, each with specific tasks and deliverables, to guide the software development process from inception to deployment and maintenance.

**Key Stages of SDLC:**

1. **Requirement Analysis**
   * Gather and analyze user and business requirements.
   * Define functional and non-functional requirements.
   * Document requirements in a Software Requirement Specification (SRS).
2. **Planning**
   * Develop a project plan, including timeline, resources, and budget.
   * Identify risks and create mitigation strategies.
   * Set goals and milestones.
3. **System Design**
   * Define the architecture and design of the software.
   * Create diagrams (e.g., flowcharts, ER diagrams) and mockups.
   * Specify hardware, software, and system requirements.
4. **Implementation (Coding)**
   * Write the code for the application based on the design.
   * Follow coding standards and guidelines.
   * Integrate individual modules to build the complete system.
5. **Testing**
   * Validate and verify the software through various testing methods (e.g., unit, integration, system, acceptance).
   * Identify and fix bugs to ensure quality and reliability.
6. **Deployment**
   * Release the software to the production environment.
   * Perform necessary configurations and installations.
   * Provide user training and support as needed.
7. **Maintenance**
   * Monitor performance and resolve issues post-deployment.
   * Implement updates, enhancements, and patches.
   * Ensure the software continues to meet evolving user needs.

**Popular SDLC Models:**

1. **Waterfall Model**: Sequential, phase-by-phase approach.
2. **Agile Model**: Iterative and incremental development.
3. **Spiral Model**: Combines iterative development with risk analysis.
4. **V-Model**: Verification and validation occur side-by-side.
5. **Iterative Model**: Develops software in repeated cycles.

Each SDLC stage plays a crucial role in delivering a robust, efficient, and user-centric software product.

1. **Test Objectives**

Test Objectives are the specific goals and purposes of testing in a software development project. These objectives guide the testing process and ensure that it aligns with the project's requirements and quality expectations.

**Key Test Objectives:**

1. **Verify Functionality:**
   * Ensure the software behaves as intended and meets the specified functional requirements.
2. **Identify Defects:**
   * Detect errors, bugs, and inconsistencies in the software to ensure it works correctly under various conditions.
3. **Validate Performance:**
   * Confirm that the system performs efficiently under expected and stress conditions, including response times and throughput.
4. **Ensure Compatibility:**
   * Check the software’s ability to function across different environments, platforms, devices, and browsers.
5. **Enhance Security:**
   * Validate the software’s ability to protect data and maintain confidentiality, integrity, and availability against potential threats.
6. **Verify Usability:**
   * Ensure the software is user-friendly and meets user expectations for ease of use and accessibility.
7. **Validate Non-functional Requirements:**
   * Confirm compliance with quality attributes such as reliability, scalability, portability, and maintainability.
8. **Prevent Future Defects:**
   * Identify patterns or root causes of defects to improve development practices and prevent similar issues.
9. **Build Confidence:**
   * Provide stakeholders with assurance that the software is ready for release and will meet business needs.
10. **Meet Compliance Standards:**
    * Verify that the software adheres to industry regulations, standards, and organizational policies.

These objectives collectively ensure that the software is of high quality, meets user needs, and functions reliably in its intended environment.

1. **Water fall, Iterative and Incremental, Spiral, Agile, Use Case**

**Water fall:-**

Waterfall Model is a sequential model that divides software development into pre-defined phases. Each phase must be completed before the next phase can begin with no overlap between the phases. Each phase is designed for performing specific activity during the SDLC phase. It was introduced in 1970 by Winston Royce.

**Different Phases of Waterfall Model in Software Engineering**

**1-Requirement Gathering stage**

During this phase, detailed requirements of the software system to be developed are gathered from client

**2-Design Stage**

Plan the programming language, for Example Java, PHP, .net

or database like Oracle, MySQL, etc.

Or other high-level technical details of the project

**3-Built Stage**

After design stage, it is built stage, that is nothing but coding the software

**4-Test Stage**

in this phase, you test the software to verify that it is built as per the specifications given by the client.

**5-Deployment stage**

Deploy the application in the respective environment

**6-Maintenance stage**

Once your system is ready to use, you may later require change the code as per customer request

**When to use SDLC Waterfall Model?**

Requirements are not changing frequently

Application is not complicated and big

Project is short

Requirement is clear

Environment is stable

Technology and tools used are not dynamic and is stable

Resources are available and trained

**Advantages Waterfall Model**

Before the next phase of development, each phase must be completed

Suited for smaller projects where requirements are well defined

They should perform quality assurance test (Verification and Validation) before completing each stage

Elaborate documentation is done at every phase of the software’s development cycle

Project is completely dependent on project team with minimum client intervention

Any changes in software is made during the process of the development

**DisAdvantages Waterfall Model**

Error can be fixed only during the phase

It is not desirable for complex project where requirement changes frequently

Testing period comes quite late in the developmental process

Documentation occupies a lot of time of developers and testers

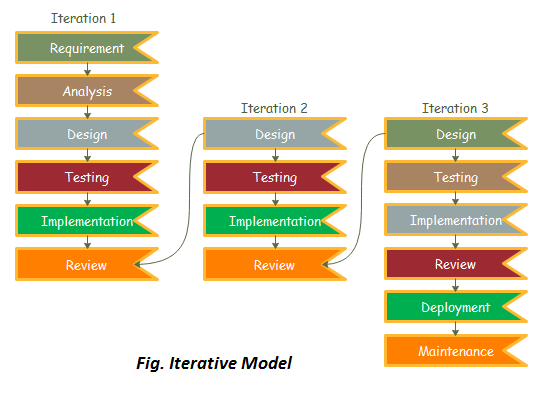
Clients valuable feedback cannot be included with ongoing development phase

Small changes or errors that arise in the completed software may cause a lot of problems

**Iterative:-**

you can start with some of the software specifications and develop the first version of the software. After the first version if there is a need to change the software, then a new version of the software is created with a new iteration. Every release of the Iterative Model finishes in an exact and fixed period that is called iteration.

The Iterative Model allows the accessing earlier phases, in which the variations made respectively. The final output of the project renewed at the end of the Software Development Life Cycle (SDLC) process



**The various phases of Iterative model are as follows:**

**1. Requirement gathering & analysis:** In this phase, requirements are gathered from customers and check by an analyst whether requirements will fulfil or not. Analyst checks that need will achieve within budget or not. After all of this, the software team skips to the next phase.

**2. Design:** In the design phase, team design the software by the different diagrams like Data Flow diagram, activity diagram, class diagram, state transition diagram, etc.

**3. Implementation:** In the implementation, requirements are written in the coding language and transformed into computer programmes which are called Software.

**4. Testing:** After completing the coding phase, software testing starts using different test methods. There are many test methods, but the most common are white box, black box, and grey box test methods.

**5. Deployment:** After completing all the phases, software is deployed to its work environment.

**6. Review:** In this phase, after the product deployment, review phase is performed to check the behaviour and validity of the developed product. And if there are any error found then the process starts again from the requirement gathering.

**7. Maintenance:** In the maintenance phase, after deployment of the software in the working environment there may be some bugs, some errors or new updates are required. Maintenance involves debugging and new addition options.

**When to use the Iterative Model?**

When requirements are defined clearly and easy to understand.

When the software application is large.

When there is a requirement of changes in future.

**Advantage(Pros) of Iterative Model:**

Testing and debugging during smaller iteration is easy.

A Parallel development can plan.

It is easily acceptable to ever-changing needs of the project.

Risks are identified and resolved during iteration.

Limited time spent on documentation and extra time on designing.

**Disadvantage(Cons) of Iterative Model:**

It is not suitable for smaller projects.

More Resources may be required.

Design can be changed again and again because of imperfect requirements.

Requirement changes can cause over budget.

Project completion date not confirmed because of changing requirements.

**Spiral:-**

Spiral Model is a risk-driven software development process model. It is a combination of waterfall model and iterative model. Spiral Model helps to adopt software development elements of multiple process models for the software project based on unique risk patterns ensuring efficient development process.

Each phase of spiral model in software engineering begins with a design goal and ends with the client reviewing the progress. The spiral model in software engineering was first mentioned by Barry Boehm in his 1986 paper.

The development process in Spiral model in SDLC, starts with a small set of requirement and goes through each development phase for those set of requirements. The software engineering team adds functionality for the additional requirement in every-increasing spirals until the application is ready for the production phase.

**Spiral Model Phases**

**Planning - It includes estimating the cost, schedule and resources for the iteration. It also involves understanding the system requirements for continuous communication between the system analyst and the customer**

**Risk Analysis - dentification of potential risk is done while risk mitigation strategy is planned and finalized**

**Engineering - It includes testing, coding and deploying software at the customer site**

**Evaluation - Evaluation of software by the customer. Also, includes identifying and monitoring risks such as schedule slippage and cost overrun**

**When to use Spiral Model?**

**A Spiral model in software engineering is used when project is large**

**When releases are required to be frequent, spiral methodology is used**

**When creation of a prototype is applicable**

**When risk and costs evaluation is important**

**Spiral methodology is useful for medium to high-risk projects**

**When requirements are unclear and complex, Spiral model in SDLC is useful**

**When changes may require at any time**

**When long term project commitment is not feasible due to changes in economic priorities**

**Spiral Model Advantages**

**Additional functionality or changes can be done at a later stage**

**Cost estimation becomes easy as the prototype building is done in small fragments**

**Continuous or repeated development helps in risk management**

**Development is fast and features are added in a systematic way in Spiral development**

**There is always a space for customer feedback**

**Spiral Model DisAdvantages**

**Risk of not meeting the schedule or budget**

**Spiral development works best for large projects only also demands risk assessment expertise**

**For its smooth operation spiral model protocol needs to be followed strictly**

**Documentation is more as it has intermediate phases**

**Spiral software development is not advisable for smaller project, it might cost them a lot**

1. **Objectives and purpose**

**Objectives**

Objectives refer to specific, measurable goals or targets that an activity or process aims to achieve. In the context of software testing or development, objectives guide the direction and effort toward achieving desired outcomes.

**Examples of Objectives:**

1. Ensure the software meets specified requirements.
2. Identify and resolve defects to improve quality.
3. Validate performance under various conditions.
4. Ensure compatibility across devices and platforms.
5. Meet project deadlines and deliverables.

**Purpose**

**Purpose** refers to the overall reason or rationale behind an activity or process. It answers the "why" and focuses on the bigger picture or long-term vision.

**Examples of Purpose:**

1. Deliver a reliable and user-friendly product to end-users.
2. Reduce costs and time associated with defect fixes post-release.
3. Build trust and confidence among stakeholders.
4. Comply with legal, regulatory, or industry standards.
5. Achieve business goals through effective software solutions.
6. **Software Testing Methodologies**

**Software Testing Methodologies** are strategies or approaches used to test a software application to ensure it meets requirements, is free of defects, and performs as expected. These methodologies help testers systematically plan, execute, and analyze tests.

**Key Software Testing Methodologies:**

**1. Manual Testing**

* Testing performed manually by a tester without using automation tools.
* Ensures the application behaves as expected by simulating real-world user actions.
* Includes exploratory testing, usability testing, and ad-hoc testing.

**2. Automated Testing**

* Involves using software tools and scripts to perform tests.
* Useful for repetitive and regression tests to save time and improve accuracy.
* Tools: Selenium, JUnit, TestNG, Cypress.

**3. Black-Box Testing**

* Tests the software's functionality without knowledge of the internal code or structure.
* Focuses on inputs and expected outputs.
* Types: Functional, regression, and user acceptance testing.

**4. White-Box Testing (Glass-Box)**

* Involves testing the internal logic and structure of the code.
* Requires knowledge of programming and code logic.
* Types: Unit testing, code coverage analysis, and mutation testing.

**5. Gray-Box Testing**

* Combines elements of black-box and white-box testing.
* Testers have partial knowledge of the code or system architecture.

**6. Functional Testing**

* Validates that the software performs its intended functions correctly.
* Focuses on user requirements and features.
* Includes smoke testing, integration testing, and system testing.

**7. Non-Functional Testing**

* Assesses non-functional aspects such as performance, security, and scalability.
* Types: Load testing, stress testing, usability testing, and compatibility testing.

**8. Exploratory Testing**

* Performed without predefined test cases; testers explore the application on the fly.
* Useful for uncovering unexpected issues.

**9. Regression Testing**

* Ensures that recent code changes have not adversely affected existing functionality.
* Performed after bug fixes or new feature implementations.

**10. Acceptance Testing**

* Verifies the software is ready for release and meets user needs.
* Types: Alpha testing (internal) and beta testing (external by users).

**11. Agile Testing**

* Testing performed in iterative and incremental development cycles.
* Emphasizes collaboration between developers and testers.
* Examples: Continuous testing, sprint testing.

**12. Performance Testing**

* Evaluates the system’s speed, responsiveness, and stability under load.
* Types: Load testing, stress testing, and scalability testing.

**Choosing the Right Methodology**

The choice of testing methodology depends on factors such as:

* Project size and complexity.
* Time and budget constraints.
* Type of application (e.g., web, mobile, embedded).
* Development model (e.g., Agile, Waterfall).

Each methodology complements the others and contributes to delivering a high-quality software product.

1. **When and Why Testing?**

**When to Perform Testing?**

Testing should be conducted throughout the **Software Development Life Cycle (SDLC)** to identify and fix defects early, ensuring quality and efficiency. Testing typically occurs at various stages:

1. **Requirement Analysis**
   * Perform validation testing to ensure requirements are clear, complete, and testable.
2. **Design Phase**
   * Conduct review and walkthrough testing of design documents.
3. **Development Phase**
   * Perform unit testing to validate individual components.
   * Begin integration testing to ensure modules work together.
4. **Post-Development**
   * Conduct system testing to verify the software as a whole.
   * Perform acceptance testing to confirm it meets user needs.
5. **Pre-Deployment**
   * Perform regression testing after bug fixes or changes.
   * Conduct performance and load testing to evaluate readiness for production.
6. **Post-Deployment**
   * Execute maintenance testing to verify updates, patches, or enhancements.
   * Conduct compatibility testing when introducing new environments.

**Why is Testing Important?**

Testing ensures the **quality, reliability, and usability** of the software. Key reasons include:

1. **Identify Defects Early**
   * Detect and fix bugs before they escalate into costly problems.
2. **Ensure Quality**
   * Validate that the software meets functional and non-functional requirements.
3. **Enhance User Satisfaction**
   * Deliver a smooth, error-free experience, improving user trust and satisfaction.
4. **Prevent Failures**
   * Reduce the risk of software failures in production environments, minimizing downtime.
5. **Save Costs and Time**
   * Fixing bugs during the development phase is significantly cheaper than post-deployment fixes.
6. **Ensure Security**
   * Identify vulnerabilities to protect sensitive data and systems from threats.
7. **Meet Compliance**
   * Verify the software adheres to industry regulations and standards.
8. **Improve Performance**
   * Test for speed, responsiveness, and scalability under different workloads.
9. **Support Business Goals**
   * Ensure the software aligns with business objectives and supports operational success.

Testing is critical for delivering high-quality software that performs effectively, remains secure, and satisfies end-users.

1. **Software Requirement Specifications (SRS)**

**Software Requirements Specification (SRS)** is a detailed document that outlines the functional and non-functional requirements for a software system. It serves as a blueprint for both the development and testing teams, ensuring that all stakeholders have a clear understanding of the software's features, capabilities, and constraints.

**Key Components of an SRS Document:**

1. **Introduction**
   * **Purpose**: Describes the purpose of the document and the software system.
   * **Scope**: Defines the boundaries of the software, including what will and won't be included.
   * **Definitions, Acronyms, and Abbreviations**: Lists key terms used in the document.
   * **References**: Citations of relevant documents, standards, or previous versions.
2. **Overall Description**
   * **Product Perspective**: Describes the context of the software within a larger system, if applicable.
   * **Product Features**: A high-level list of the software's key features.
   * **User Classes and Characteristics**: Identifies different types of users and their needs.
   * **Operating Environment**: Specifies hardware, software, and network environments where the software will operate.
   * **Design and Implementation Constraints**: Any technical constraints, such as programming languages, platforms, or regulations.
   * **Assumptions and Dependencies**: Lists assumptions made during development and dependencies on other systems.
3. **System Features**
   * **Functional Requirements**: Detailed description of each feature or function the software must perform. Each requirement should be clear, concise, and testable.
     + Example: *The system shall allow users to log in using an email address and password.*
   * **Use Cases or User Stories**: Specific scenarios that describe how users interact with the software.
4. **External Interface Requirements**
   * **User Interfaces**: Describes the user interface (UI) design, including mockups or wireframes if available.
   * **Hardware Interfaces**: Defines hardware requirements or specifications for interaction with the system.
   * **Software Interfaces**: Specifies integration with other software or third-party services.
   * **Communication Interfaces**: Details the communication protocols, formats, or APIs used in the system.
5. **System Attributes (Non-Functional Requirements)**
   * **Performance**: Defines performance requirements such as response time, throughput, and scalability.
   * **Security**: Describes security requirements, such as authentication, encryption, and access control.
   * **Reliability**: Defines uptime, error rates, and recovery requirements.
   * **Maintainability**: Specifies how easy it should be to update, debug, and enhance the software.
   * **Usability**: Defines how user-friendly the system should be, including accessibility considerations.
   * **Portability**: Defines how the software should work across different platforms or environments.
6. **Other Requirements**
   * **Data Requirements**: Describes data inputs, outputs, storage, and database structure.
   * **Legal and Regulatory Requirements**: Specifies any legal, regulatory, or compliance constraints (e.g., GDPR, HIPAA).
   * **Internationalization**: Defines support for different languages, regions, or locales if applicable.

**Purpose of an SRS Document:**

* **Clear Communication**: Provides a clear, concise, and unambiguous description of the software's functionality and constraints to all stakeholders, including developers, testers, and clients.
* **Scope Management**: Helps prevent scope creep by defining what is included in the software and what is not.
* **Basis for Design and Testing**: Serves as the foundation for software design, implementation, and testing, ensuring that the final product meets the specified requirements.
* **Legal and Contractual Agreement**: Can act as a contractual agreement between the client and the development team, ensuring both parties have the same understanding of the project's goals and deliverables.

An SRS document is a crucial artifact in software development that helps align the entire team towards a common vision and ensures the software meets both user needs and business goals.

1. **When to start and stop Testing?**

**When to Start Testing?**

Testing should begin early in the **Software Development Life Cycle (SDLC)** and continue throughout the process. The goal is to identify defects early and ensure that the software meets all requirements. Key points for when to start testing include:

1. **During the Requirement Analysis Phase**
   * **Start early** by reviewing and validating the requirements to ensure they are clear, complete, and testable.
   * This helps in identifying ambiguities or incomplete requirements before development begins.
2. **In the Design Phase**
   * **Start planning** test cases based on the design documents (e.g., use cases, system architecture).
   * Perform **design verification** to ensure the design aligns with the requirements and is testable.
3. **During Development (Coding)**
   * **Unit Testing** begins as soon as developers write code. Each unit or module is tested independently to ensure correctness.
   * As development progresses, **integration testing** can start to ensure that individual components work together correctly.
4. **Throughout the Development Process**
   * **Continuous Testing**: In Agile or Continuous Integration (CI) environments, testing is continuous with frequent iterations. Automated tests can be run frequently to catch defects early.

**When to Stop Testing?**

Testing does not have a fixed end point, but there are several key indicators that testing can be concluded:

1. **When All Test Cases Have Been Executed**
   * All planned test cases (functional, non-functional, regression, etc.) should be executed. If there are no outstanding test cases and all have passed, it may be time to stop.
2. **When Defects are Resolved**
   * If no significant defects remain after testing and all high-priority issues have been addressed, testing can be considered complete. However, some minor defects may remain, and they can be deferred or fixed in future releases.
3. **When the Risk of Defects is Acceptable**
   * If the software is stable, meets requirements, and has undergone thorough testing with no major risks, you may stop testing. The decision depends on the risk tolerance of stakeholders.
4. **Test Results Are Consistent**
   * If the results of repeated tests (e.g., regression or load testing) are consistent and no new issues are found, testing can be stopped.
5. **The Budget and Time Constraints Are Met**
   * Testing must often be stopped due to time or budget limitations. When testing has reached its planned scope or allocated resources, it may need to be concluded even if all test cases haven't been executed.
6. **When Test Coverage Meets Requirements**
   * If testing has covered all critical functionality and user requirements, and the product has met the acceptance criteria, testing may be stopped.
7. **When Software is Deemed Ready for Release**
   * After completing system and acceptance testing, and ensuring the software works as expected for the user, testing can be concluded. It’s essential to ensure that the software is production-ready.
8. **Structure Query Language (SQL)**

**Structured Query Language (SQL)** is a standard programming language used for managing and manipulating relational databases. It provides a set of commands to interact with the database, such as retrieving, inserting, updating, and deleting data.

**Advantages of SQL:**

* **Standardized**: SQL is widely used and standardized by ANSI and ISO.
* **Powerful and Flexible**: Handles complex queries, joins, and operations across multiple tables.
* **Ease of Use**: Provides a simple syntax for querying and manipulating data

1. **7 Key Principles of Testing**

**General Testing Principles**

1. Testing shows presence of Defects

2. Exhaustive Testing is Impossible

3. Early Testing

4. Defect Clustering

5. The Pesticide Paradox

6. Testing is Context Dependent

7. Absence of Errors Fallacy

**15) OOPS**

* Identifying objects and assigning responsibilities to these objects.
* Objects communicate to other objects by sending messages.
* Messages are received by the methods of an object
* An object is like a black box.
* The internal details are hidden.
* Object is derived from abstract data type Object-oriented programming has a web of interacting objects, each house-keeping its own state.
* Objects of a program interact by sending messages to each other.