Internal Assessment Test 1 – July 2025

Software Engineering							Sub Code:	MMC204
0107/2025	Duration:	90 min's	Max Marks:	50	Sem:	I	Branch:	MCA

PART I

1. Explain IEEE/ACM code of Software Engineering ethics.

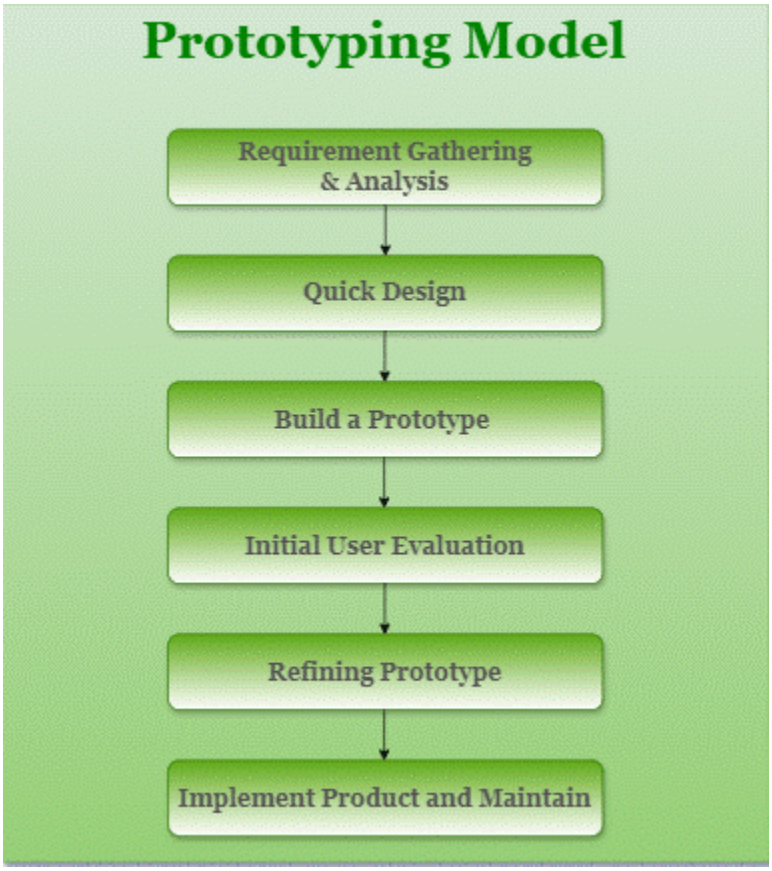
The IEEE (Institute of Electrical and Electronics Engineers) and ACM (Association for Computing Machinery) have jointly established a code of ethics and professional practices for software engineers. This code provides guidelines for ethical behavior and professional conduct in the field of software engineering. While the specific details may evolve over time, the general principles include:

1. PUBLIC: Software engineers shall act consistently with the public interest. They should consider the safety, health, and welfare of the public and disclose any factors that could endanger it.
2. CLIENT AND EMPLOYER: Software engineers shall act in a manner that is in the best interests of their clients and employers and shall ensure that the products of their work meet the highest professional standards.
3. PRODUCT: Software engineers shall ensure that their products and related modifications meet the highest professional standards possible. This includes being diligent in the design, testing, and maintenance of software.
4. JUDGMENT: Software engineers shall maintain integrity and independence in their professional judgment. They should provide honest assessments of software and system implications, ensuring that decisions are made based on objective analysis.
5. MANAGEMENT: Software engineers shall subscribe to fair management practices and not promote any action that is known to be unethical or illegal. They should work to improve their management skills and promote an ethical approach to the management of software development and maintenance.
6. PROFESSION: Software engineers shall advance the integrity and reputation of the profession Consistent with the public interest. They should foster professional development, mentor colleagues, and contribute to the community to enhance the understanding and appreciation of the field.
7. COLLEAGUES: Software engineers shall be fair to and supportive of their colleagues. They should avoid malicious or subversive behavior and strive to foster a positive and collaborative working environment.
8. SELF: Software engineers shall participate in lifelong learning and consistently enhance their Professional skills. They should promote ethical approaches to the practice of software engineering and maintain high standards of professional conduct.

Adherence to this code of ethics is crucial for maintaining the trust of clients, employers, and the public, and for ensuring the responsible and ethical development of software and technology**.**

2. **Describe Prototyping Model of Software development with neat diagrams.**

The prototyping paradigm begins with communication. Developer and customer meet and define the overall objectives for the software, identify whatever requirements are known, A quick design focuses on a representation of those aspects of the software that will be visible to the customer/user (e.g. Input approaches and output formats). The quick design leads to the construction of a prototype.
The prototype is evaluated by the customer/user and used to refine requirements for the software To be developed. Iteration occurs as the prototype is tuned to satisfy the needs of the customer, While at the same time enabling the developer to better understand what needs to be done. Ideally, the prototype serves as a mechanism for identifying software requirements. If a working prototype is built, the developer attempts to use existing program fragments or applies tools (e.g., report generators, window managers) that enable working programs to be generated quickly.



Advantages:
Requirements can be set earlier and more reliably.
Customer sees results very quickly.
Customer is educated in what is possible helping to refine requirements.
Requirements can be communicated more clearly and completely.
Between developers and clients Requirements and design options can be investigated quickly and cheaply.
Drawbacks of prototyping:
In the first version itself, customer often wants ―few fixes‖ rather than rebuilding of the system

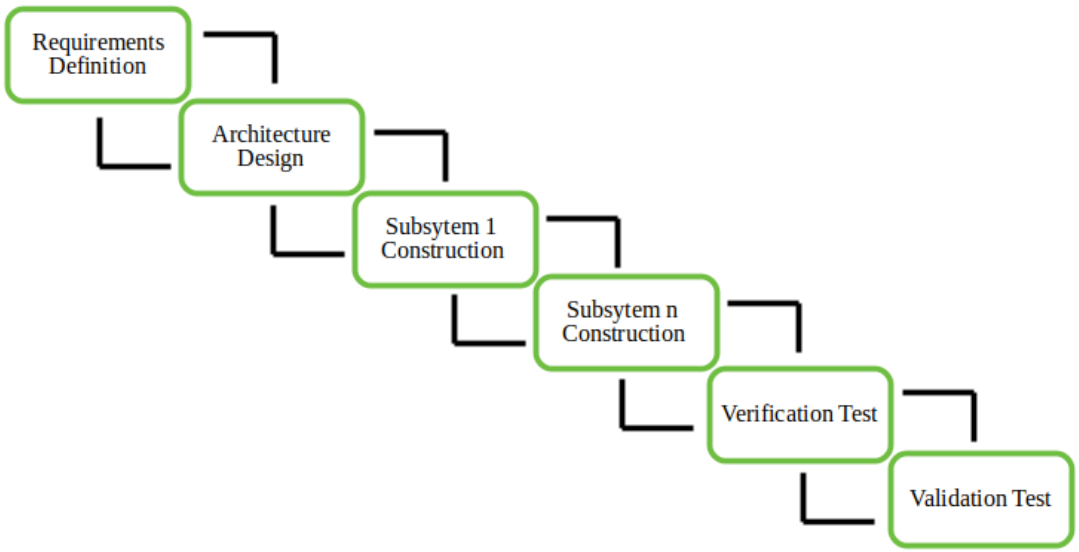
Whereas rebuilding of new system maintains high level of quality.
The first version may have some compromises.
Sometimes developer may make implementation compromises to get prototype working quickly.
Later on developer may become comfortable with compromises and forget why they are inappropriate.

PART II

3. Discuss the process involved in Incremental Model with advantages and disadvantages.

The incremental model delivers a series of releases, called increments that provide progressively more functionality for the customer as each increment is delivered. The incremental model applies linear sequences in a staggered fashion as calendar time progresses. Each linear sequence produces deliverable ―increments‖ of the software [McD93] in a manner that is similar to the increments produced by an evolutionary process flow. The first increment is called core product. In this release the basic requirements are implemented and then in subsequent increments new requirements are added. The core product is used by the customer (or undergoes detailed evaluation). As a result of use and/or evaluation, a plan is developed for the next increment. The plan addresses the modification of the core product to better meet the needs of the customer and the delivery of additional features and functionality. This process is repeated following the delivery of each increment, until the complete product is produced.

Incremental Process Model



i) In the second increment, more sophisticated document producing and processing facilities, file management functionalities are given.

Incremental process Model advantages

1. Produces working software early during the lifecycle.
2. More flexible as scope and requirement changes can be implemented at low cost.
3. Testing and debugging is easier, as the iterations are small.
4. Low risks factors as the risks can be identified and resolved during each iteration.

Incremental process Model disadvantages

1. This model has phases that are very rigid and do not overlap.
2. Not all the requirements are gathered before starting the development; this could lead to Problems related to system architecture at later iterations.

4. Explain in detail the principles of Agile methods.

Agile is an iterative and incremental approach to software development that emphasizes flexibility,
collaboration, and customer satisfaction. Here are the twelve Agile principles:

1. Customer Satisfaction through Early and Continuous Delivery of Valuable Software:
- Deliver working software frequently, with a preference for shorter timescales, to provide tangible value to the customer.

2. Welcome Changing Requirements, Even Late in Development:
- Embrace changes in requirements, even if they occur late in the development process, to provide the customer with a competitive advantage.

3. Deliver Working Software Frequently:
- Aim to deliver a working product as frequently as possible, with a preference for shorter development cycles.

4. Collaboration between Business Stakeholders and Developers:
- Foster a collaborative environment where business stakeholders and developers work together daily throughout the project.

5. Build Projects around Motivated Individuals:
- Give motivated individuals the resources and support they need and trust them to get the job
done.

6. Face-to-Face Communication is Most Effective:
- Maximize face-to-face communication within the team as it is the most efficient and effective
method of conveying information.

7. Working Software is the Primary Measure of Progress:
- Focus on delivering a working product, as it is the ultimate measure of progress in Agile development.

8. Maintain a Sustainable Pace of Work:

🎬 Encourage a sustainable pace of work to maintain a consistent level of productivity and prevent burnout.

9. Continuous Attention to Technical Excellence and Good Design:
🎬 Prioritize technical excellence and good design to ensure the long-term maintainability and
adaptability of the software.

10. Simplicity—the Art of Maximizing the Amount of Work Not Done:

🎬 Emphasize simplicity in design and implementation, maximizing the value of work accomplished and minimizing unnecessary complexity.

11. Self-Organizing Teams:
🎬 Allow the team to self-organize, encouraging collaboration and creativity while recognizing the
expertise of individual team members.
12. Regular Reflection and Adaptation:
🎬 Regularly reflect on the team's performance and adapt processes accordingly, fostering a culture of continuous improvement.

PART III

5. Describe requirements elicitation and analysis process with example.

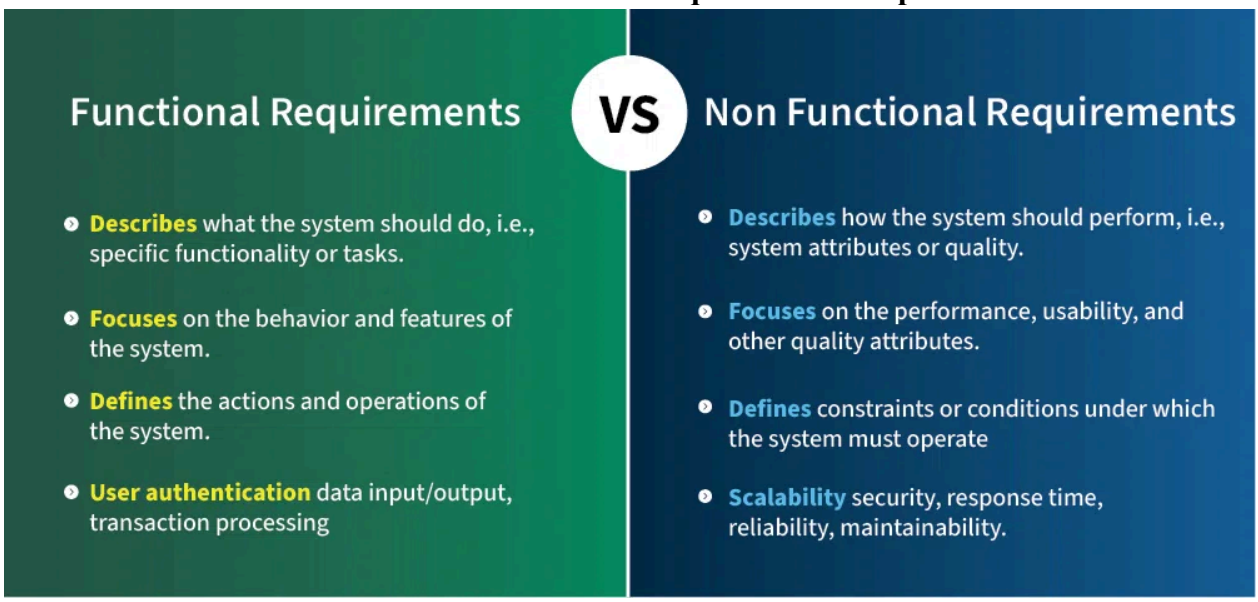
Requirements Elicitation

🕮 It is related to the various ways used to gain knowledge about the project domain and requirements.
🕮 The various sources of domain knowledge include customers, business manuals, the existing software of the same type, standards, and other stakeholders of the project.
🕮 The techniques used for requirements elicitation include interviews, brainstorming, task analysis, Delphi technique, prototyping, etc.
🕮 Requirements elicitation is the process of gathering information about the needs and expectations of stakeholders for a software system.
Several techniques can be used to elicit requirements, including:
• Interviews: These are one-on-one conversations with stakeholders to gather information about their needs and expectations.
• Surveys: These are questionnaires that are distributed to stakeholders to gather information about their needs and expectations.
• Focus Groups: These are small groups of stakeholders who are brought together to discuss their needs and expectations for the software system.
• Observation: This technique involves observing the stakeholders in their work environment to gather information about their needs and expectations.
• Prototyping: This technique involves creating a working model of the software system, which can be used to gather feedback from stakeholders and to validate requirements.

Activities involved in Software Requirement Analysis

Steps involved in the Requirement Analysis Process
Step 1: Identifying Stakeholders and Communicating Needs
Step 2: Gathering Requirements
Step 3: Analyzing and Prioritizing Requirements
Step 4: Documenting Requirements
Step 5: Validating Requirements
Step 6: Creating Use Cases or User Stories
Step 7: Getting Sign-off and Approval
Step 8: Managing Changes

6. **What are Functional and Non-Functional requirements? Explain in detail..**



Functional Requirements?
These are the requirements that the end user specifically demands as basic facilities that the system should offer. All these functionalities need to be necessarily incorporated into the system as a part of the contract.

- These are represented or stated in the form of input to be given to the system, the operation performed and the output expected.
- They are the requirements stated by the user which one can see directly in the final product, unlike the non-functional requirements.

Examples:

- What are the features that we need to design for this system?
- What are the edge cases we need to consider, if any, in our design?

Non-Functional Requirements?
These are the quality constraints that the system must satisfy according to the project contract. The priority or extent to which these factors are implemented varies from one project to another. They are also called non-behavioral requirements. They deal with issues like:

- Portability

- Security
- Maintainability
- Reliability
- Scalability
- Performance
- Reusability
- Flexibility

Examples:

- Each request should be processed with the minimum latency?
- System should be highly available.

PART IV

7. **Explain the format and characteristics of good SRS.**

Structure of a Typical SRS Document (IEEE Standard 830)
1. Introduction
1.1 Purpose
Describes the purpose of the SRS and the intended audience.
Example:
"This SRS describes the requirements for an online bookstore system, intended for use by customers, staff, and administrators."
1.2 Scope
Describes the software to be developed, its objectives, and its benefits.
Example:
"The system allows users to browse, search, purchase books online, and enables administrators to manage inventory and orders."
1.3 Definitions, Acronyms, and Abbreviations
Terms used throughout the document for clarity.
1.4 References
Documents or standards referred to in the SRS.
1.5 Overview
What the rest of the document contains.

2. Overall Description

2.1 Product Perspective
How the software fits into the broader system or context (standalone, part of a system, etc.).
2.2 Product Functions
High-level overview of system functions (e.g., login, checkout, etc.).
2.3 User Classes and Characteristics
Types of users and their technical background.
2.4 Operating Environment
Software, hardware, and network requirements.
2.5 Design and Implementation Constraints
Limitations due to hardware, legal, or regulatory constraints.
2.6 Assumptions and Dependencies

External factors the system depends on (e.g., third-party payment APIs).

3. Specific Requirements
3.1 Functional Requirements
Describes the behavior of the system under specific conditions.
Example:
o FR1: The system shall allow users to register using a valid email address and password.
o FR2: The system shall send a confirmation email upon successful registration.
3.2 Non-Functional Requirements
Includes performance, reliability, usability, and security aspects.
Example:
o NFR1: The system shall respond to user actions within 2 seconds.
o NFR2: Passwords shall be stored using SHA-256 hashing.
3.3 Interface Requirements
Describes interactions with hardware, software, or other systems.
3.4 User Interface Requirements
Layout, color schemes, responsiveness, accessibility, etc.

3.5 Hardware and Communication Requirements
Minimum system requirements, data transfer protocols, etc.

4. Appendices
Supporting information, mockups, data dictionaries, etc.
5. Index or Glossary
Terms and definitions used throughout the document.
Example for SRS DOCUMENT
Project Title: Library Management System
1. Introduction
1.1 Purpose
The purpose of this document is to define the requirements for the Library Management System. This SRS will be used by the development team, testers, and the client to ensure a common understanding of the system functionality.
1.2 Scope
This system is designed to manage the daily operations of a library. It will handle book inventory, member records, issue and return of books, and report generation. It is meant for librarians and library members.
1.3 Definitions, Acronyms, and Abbreviations
LMS: Library Management System
ISBN: International Standard Book Number
UI: User Interface
1.4 References
IEEE SRS Template
Client-provided requirements document
1.5 Overview

This document is organized into sections covering system description, specific requirements, and design constraints.

2. Overall Description
2.1 Product Perspective
The system is a standalone desktop/web application and does not rely on any external systems.
2.2 Product Functions
Add/view/edit/delete books
Issue and return books
Register/view members
Generate reports
2.3 User Classes and Characteristics
Admin (Librarian): Full access to all features
Member (Student/Faculty): Limited access, can view/search books, request issue
2.4 Operating Environment
Web browser (Chrome/Firefox)
Server with MySQL and PHP/Node.js backend
Windows/Linux OS
2.5 Design and Implementation Constraints
System should be developed using JavaScript (Node.js) and MySQL
Responsive UI using HTML/CSS/Bootstrap
2.6 Assumptions and Dependencies
Internet connectivity required for web version
Database server must be running during operations

3. Specific Requirements
3.1 Functional Requirements
FR-1: System shall allow the admin to add a new book with details like title, author, ISBN, and quantity.
FR-2: System shall allow the admin to register a new member with ID, name, and contact info.
FR-3: System shall allow members to search books by title, author, or category.
FR-4: System shall allow the admin to issue and return books, updating the inventory accordingly.
FR-5: System shall send an alert when a book is overdue.
3.2 Non-Functional Requirements
Performance: System should support up to 100 simultaneous users.
Security: Login required for admin; data encrypted during transmission.
Usability: UI should be clean and easy to navigate for non-technical users.
Reliability: Backup system must run daily.
3.3 Interface Requirements
User Interface: Login page, dashboard, book/member management pages
Hardware Interface: Barcode scanner (optional)
Software Interface: MySQL database connection
Communication Interface: Email service for overdue alerts

4. Appendices
Sample book entry form
Entity-Relationship (ER) diagram
UI mockups

CHARACTERISTICS OF THE SRS

1. Completeness: A SRS is complete if everything the software the software is supposed to do and the responses of the software to all class of input data are specified in SRS. It ensures that everything is indeed specified. It is one of the most difficult properties to spot. To ensure completeness, one has to detect the absence of specification which is much harder to determine.
2. Clarity: The documented requirement should lead to only a single interpretation, independent of the person or the time when the interpretation is done. The SRS needs to be unambiguous to the authors, the users, other reviewers as well as the developers and testers who will use the document. So SRS writer should be careful about ambiguity. One way to avoid ambiguity is to use some formal requirements specification language, but it is the major drawback. The formal languages require more effort to write an SRS more cost and the increased difficulty in reading and understanding formally stated requirements.
3. Correctness: The SRS can be considered as correct if every requirements stated in the SRS is required in the proposed system. Correctness of an SRS:
Ensures that what is specified is done correctly.
Is an easier property to establish as it basically involves examining each requirement to make sure that it represents the use requirements?
There are no real tools or procedures that ensure correctness. If there are any preceding documents then the requirements from those earlier documents need to be traced to the SRS:
4. Consistency: Requirements at all levels must be consistent with each other .any conflict between requirements within the SRS must be identified and resolved. The types of conflicts that generally occur are:
Characteristics (format details) of real word entity interfacing with the system maybe conflicting. For an example, one requirements states that an individual can work up o 6 hours whereas another requirement state is as 8 hours.
The terminology used for some entities events may be different for example different requirements may use different terms to refer to the same objects.
5. Verifiability: A SRS is verifiable if and only if every stated requirement is verifiability. A requirement is verifiable if there exists some cost effective process that can check whether the final software meets that requirements un ambiguity is essential for verifiability of requirements

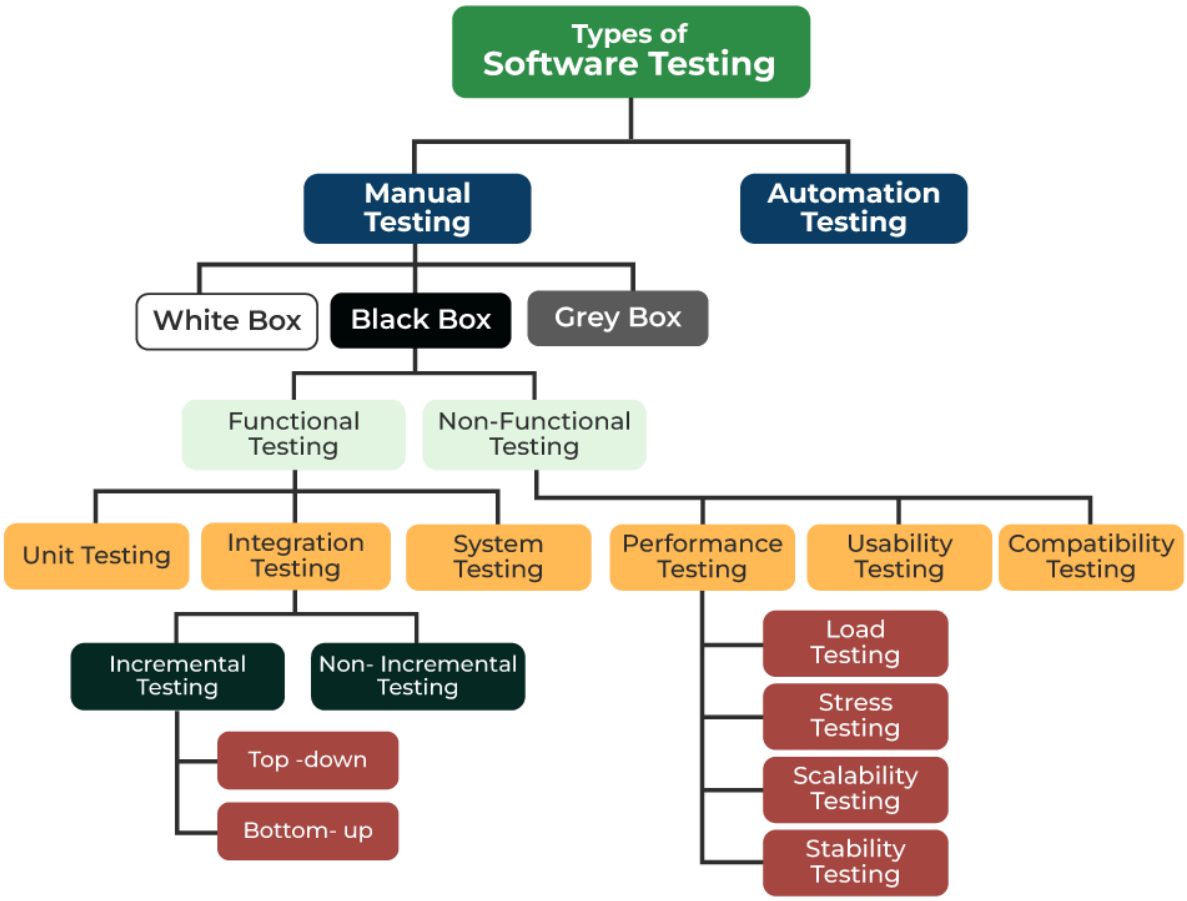
is often done through reviews it also implies that SRS in understandable, at least by the developer the client and the users.
6.Ranking : Generally, the requirements are stated according to their priorities are critical, other are important but not critical, and there are some which are desirable butnot very important. Similarly some requirements are core requirements which are not likely to change as time passes, while others are more dependent on time. A SRS is ranked for importance and or stability if for each requirement the importance and the stability of the requirements are indicated.
7. Modifiability: The SRS needs to be documented in such a manner that a history of changes can be contained in the document. It will also necessary to be able to highlight and tr5ace the

changes of the requirements as we progress through the project. Certain good practices (that can lead to high modifiability are minimal redundancy and the numbering of the requirements. According to IEEE, standard 830-1993 (recommended practice for software requirements specification) SRS is modifiable if its structure and style aresuch that any necessary changes to the requirements can be made easily while preserving completeness an consistency while retaining its structure and style.
8. Traceability: As SRS is traceable if the origin of each its requirements is clear and if it facilitates the referencing of each requirements in future development. There are two types of traceability.
9. Feasibility: Though it may not be possible to confirm the feasibility of implementation of all the requirements any requirement which is apparent infeasible, should be eliminated from the SRS.

8. What is testing? Explain the testing fundamentals.

Testing is the process of executing a program to find errors. To make our software perform well
it should be error-free. If testing is done successfully it will remove all the errors from the
software. In this article, we will discuss first the principles of testing and then we will discuss, the
different types of testing.

Principles of Testing
- All the tests should meet the customer's requirements.
- To make our software testing should be performed by a third party.
- Exhaustive testing is not possible. As we need the optimal amount of testing based on the risk assessment of the application.
- All the tests to be conducted should be planned before implementing it
- It follows the Pareto rule(80/20 rule) which states that 80% of errors come from 20%
of program components.
- Start testing with small parts and extend it to large parts.
- Types of Testing

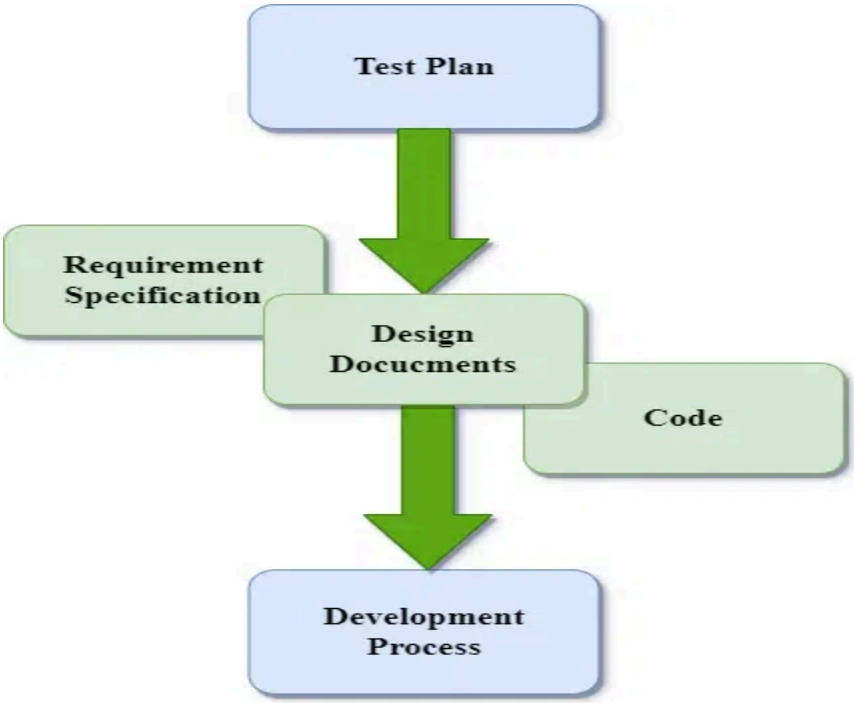


PART V

9. State and explain development testing and its levels.

Development Testing - It is a method of applying testing practices consistently throughout the software development life cycle process. This testing ensures the detection of bugs or errors at the right time which further ensures delay of any kind of risk in terms of time and cost. Development Testing aims to establish a framework to verify whether the requirements of a given project are met in accordance with the rules of the mission to be accomplished. This testing is performed by the software developers or other engineers during the construction phase of the software development lifecycle (SDLC). Development Testing is a continuous or a running process in the development of a product in the entire software development life cycle. This testing is done only once as compared to other testings which can be performed many times. To meet the deadline date, development testing is performed during the development phase of a software product,

In Development Testing, the phases are more tightly integrated so that code that is being written and checked in is automatically tested. In this way, the problems can be more quickly discovered and can be addressed.



Development testing requires some metric depending upon the organization to organization, and these may include the following :

1. **Static code Analysis :** Static code analysis is a technique of debugging by analyzing the source code before running a program. It is carried out by analyzing a set of code against a set or multiple sets of coding rules. This involves analyzing the source code, without actually executing the program.
   By performing static code analysis, the developers will know early on if there are any problems in their code and by this, it will be easier to fix those problems.

2. **Data Flow Analysis :** This concept uses the Control flow Graph mechanism to check the flow of the program, at different levels. 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. This testing uses the control flow graph to check the anomalies in the code which can interrupt the flow of the program.

3. **Metric Analysis :** Metric is a synonym for measurement. To calculate the efficiency of a program, various software metrics like calculating cyclomatic complexity, counting Lines of code (LOC), function points, etc. are used in that case. In metric analysis, Test metrics are used in taking the decision for the next phase of activities like cost estimating & future projects, recognizing the type of improvement required to succeed the project, or in taking a decision on the process or technology to be modified, etc.

4. **Code review :** The source code is inspected and is checked for any flaws in it. It can be used to find and remove flaws in the code such as memory leaks and buffer overflows. It is very important to do a code review in the early phase like a peer review, carry out this step earlier than you send your code to be tested for development. Also, do some

functionality testing of your code so that it becomes easy for code review. There are various approaches to do code reviews such as The Email thread, Pair programming, Over shoulder, and Tool-assisted.

10. What is debugging? Explain the steps involved in debugging process.

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.

What is Debugging ?

In the context of software engineering, debugging is the process of fixing a bug in the software. When there's a problem with software, programmers analyze the code to figure out why things aren't working correctly. They use different debugging tools to carefully go through the code, step by step, find the issue, and make the necessary corrections.

Why is it called debugging?

The term "debugging" originated from an incident involving Grace Hopper in the 1940s when a moth caused a malfunction in the Mark II computer at Harvard University. The term stuck and is
now commonly used to describe the process of finding and fixing errors in computer programs. In simpler terms, debugging got its name from removing a moth that caused a computer problem.

Methods and Techniques Used in Debugging

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.
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.

Process of Debugging

The steps involved in debugging are:

- Problem identification and report preparation.
- Assigning the report to the software engineer defect to verify that it is genuine.
- Defect Analysis using modeling, documentation, finding and testing candidate flaws,

etc.
● Defect Resolution by making required changes to the system.

● Validation of corrections.

The debugging process will always have one of two outcomes :
1. The cause will be found and corrected.
2. The cause will not be found.