CMR
INSTITUTE OF
TECHNOLOGY
USN

Internal Assessment Test 1 – August 2023

Sub:	Software Engineering							Sub Code:	22MCA23
Date:	02/08/2023	Duration:	90 min's	Max Marks:	50	Sem:	II	Branch	MCA

Note : Answer FIVE FULL Questions, choosing ONE full question from each Module

		MARKS	OBE	
			CO	RBT
	PART I			
1	a. Define software engineering? Explain the essential attributes of good software. b. Discuss the key challenges facing software engineering professionals.	[6 + 4]	CO1	L1
	OR			
2	Discuss the process involved in waterfall and incremental developmental model with advantages and disadvantages.	[10]	CO1	L1
	PART II			
3	a. Explain the phases of RUP with a neat diagram. b. Explain ACM/IEEE code of ethics.	[5+5]	CO2	L1
	OR			
4	Explain Prototyping model of software development with a neat diagram. Discuss the different types of Prototype models.	[10]	CO1	L1
	PART III			
5	Differentiate between Agile and Plan-driven methodologies	[10]	CO2	L3
	OR			
6	Explain in detail about extreme programming.	[10]	CO2	L3
	PART IV			
7	What are Functional and Non-Functional Requirements? Explain the different types of Non-functional requirements.	[10]	CO2	L2
	OR			
8	Explain the format and characteristics of good SRS.	[10]	CO1	L1
	PART V			
9	Define Requirements Engineering. Explain the activities of requirement engineering process.	[10]	CO2	L1
	OR			
10	Explain the different Techniques for Requirement validation?	[10]	CO2	L2

SOLUTION

1. a. Define software engineering? Explain the essential attributes of good software

Ans. Software Engineering is the study of developing software. It is the application of a systematic,

disciplined, quantifiable approach to the development, operation and maintenance of software.

Essential attributes of a good software:

Product characteristic	Description
Acceptability	Software must be acceptable to the type of users for which it is designed. This means that it must be understandable, usable, and compatible with other systems that they use.
Dependability and security	Software dependability includes a range of characteristics including reliability, security, and safety. Dependable software should not cause physical or economic damage in the event of system failure. Software has to be secure so that malicious users cannot access or damage the system.
Efficiency	Software should not make wasteful use of system resources such as memory and processor cycles. Efficiency therefore includes responsiveness, processing time, resource utilization, etc.
Maintainability	Software should be written in such a way that it can evolve to meet the changing needs of customers. This is a critical attribute because software change is an inevitable requirement of a changing business environment.

b. Discuss the key challenges facing software engineering professionals.

Ans. The methods used to develop small or medium-scale projects are not suitable when it comes to the development of large-scale or complex systems.

a. Changes in software development are unavoidable. In today's world, changes occur rapidly and accommodating these changes to develop complete software is one of the major challenges faced by the software engineers.
b. The advancement in computer and software technology has necessitated for the changes in nature of software systems. The software systems that cannot accommodate changes are not of much use. Thus, one of the challenges of software engineering is to produce high quality software adapting to the changing needs within acceptable schedules. To meet this challenge, the object-oriented approach is preferred, but accommodating changes to software and its maintenance within acceptable cost is still a challenge.
c. Informal communications take up a considerable portion of the time spent on software projects. Such wastage of time delays the completion of projects in the specified time.
d. The user generally has only a vague idea about the scope and requirements of the software system. This usually results in the development of software, which does not meet the user's requirements.

2. Discuss the process involved in waterfall and incremental developmental model with advantages and disadvantages.

Waterfall Model:

- ✧ There are separate identified phases in the waterfall model:
  - Requirements analysis and definition
  - System and software design
  - Implementation and unit testing
  - Integration and system testing

- Operation and maintenance

✧ The main drawback of the waterfall model is the difficulty of accommodating change after the process is underway. In principle, a phase has to be complete before moving onto the next phase.

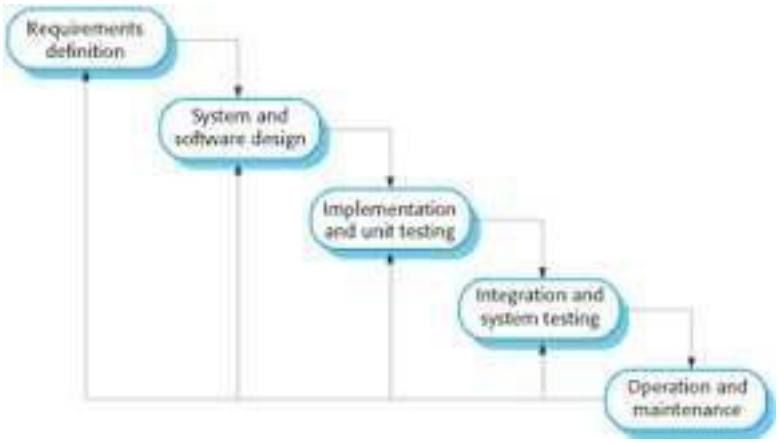
Problems:

✧ Inflexible partitioning of the project into distinct stages makes it difficult to respond to changing customer requirements.

- Therefore, this model is only appropriate when the requirements are well-understood and changes will be fairly limited during the design process.
- Few business systems have stable requirements.

✧ The waterfall model is mostly used for large systems engineering projects where a system is developed at several sites.

In those circumstances, the plan-driven nature of the waterfall model helps coordinate the work.



Incremental Development Model:

✧ The cost of accommodating changing customer requirements is reduced.

- The amount of analysis and documentation that has to be redone is much less than is required with the waterfall model.

✧ It is easier to get customer feedback on the development work that has been done.

- Customers can comment on demonstrations of the software and see how much has been implemented.

✧ More rapid delivery and deployment of useful software to the customer is possible.

- Customers are able to use and gain value from the software earlier than is possible with a waterfall process.

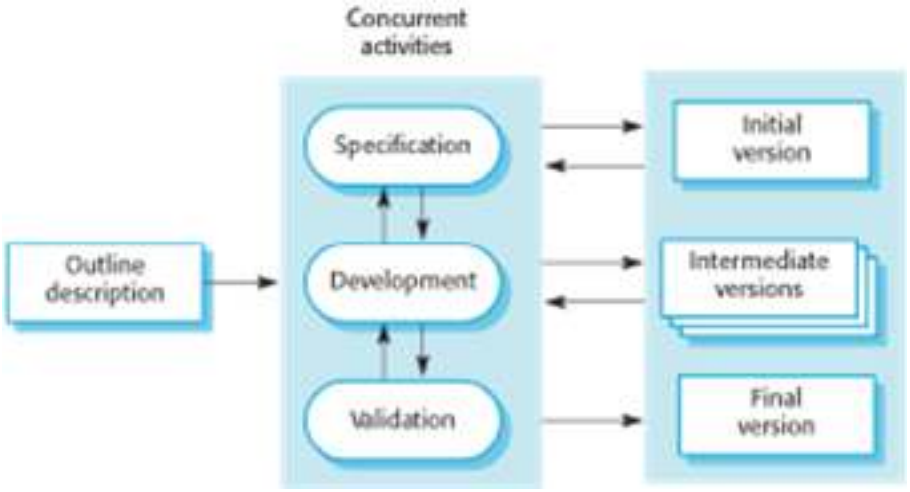
Problems:

✧ The process is not visible.

- Managers need regular deliverables to measure progress. If systems are developed quickly, it is not cost-effective to produce documents that reflect every version of the system.

✧ System structure tends to degrade as new increments are added.

- Unless time and money is spent on refactoring to improve the software, regular change tends to corrupt its structure. Incorporating further software changes becomes increasingly difficult and costly.

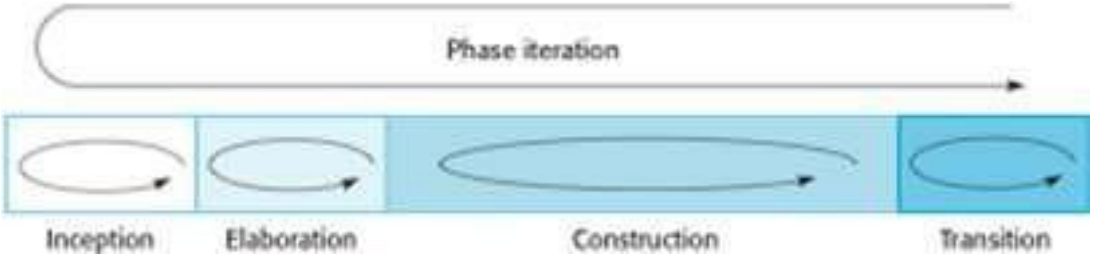

Concurrent activities
Specification
Development
Validation
Outline description
Initial version
Intermediate versions
Final version

e. Changes are usually incorporated in documents without following any standard procedure. Thus, verification of all such changes often becomes difficult.
f. The development ofl1igh-quality and reliable software requires the software to be thoroughly tested. Though thorough testing of software consumes the majority of resources, underestimating it because of any reasons deteriorates the software quality.

3. a. Explain the phases of RUP with a neat diagram.

The Rational Unified Process:

- ✧ A modern generic process derived from the work on the UML and associated process.
- ✧ Brings together aspects of the 3 generic process models discussed previously.
- ✧ Normally described from 3 perspectives
  - A dynamic perspective that shows phases over time;
  - A static perspective that shows process activities;
  - A practive perspective that suggests good practice.



3.b. Explain ACM/IEEE code of ethics.

Software Engineering Code of Ethics and Professional Practice

ACM/IEEE-CS Joint Task Force on Software Engineering Ethics and Professional Practices

PREAMBLE

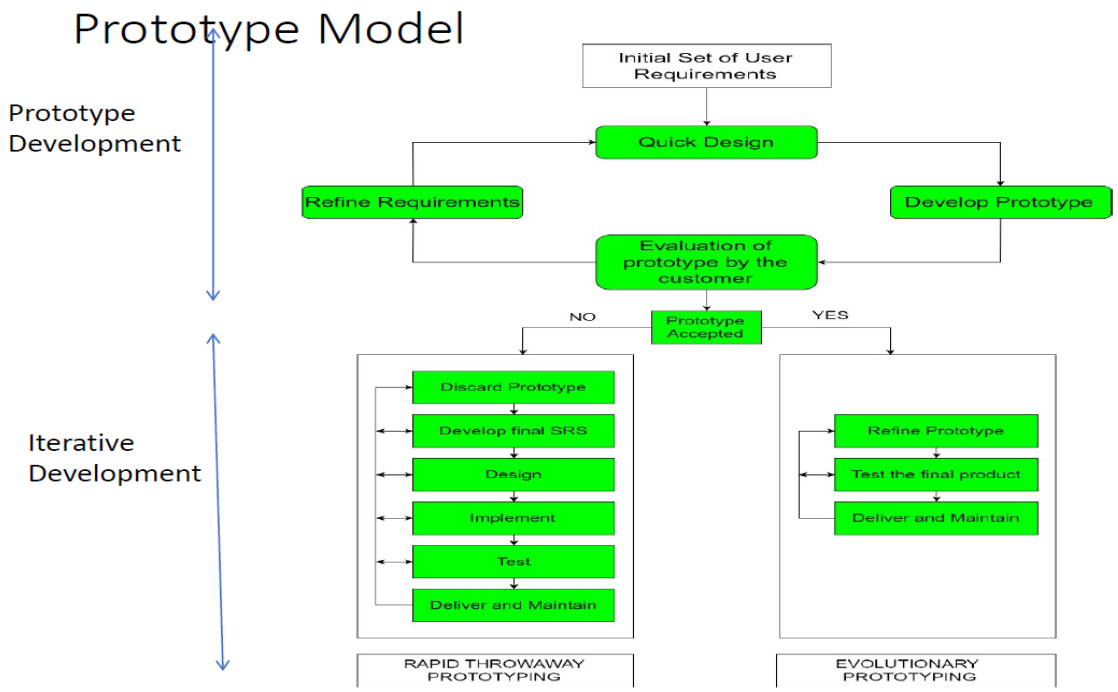
The short version of the code summarizes aspirations at a high level of the abstraction; the clauses that are included in the full version give examples and details of how these aspirations change the way we act as software engineering professionals. Without the aspirations, the details can become legalistic and tedious; without the details, the aspirations can become high sounding but empty; together, the aspirations and the details form a cohesive code.

Software engineers shall commit themselves to making the analysis, specification, design, development, testing, and maintenance of software a beneficial and respected profession. In accordance with their commitment to the health, safety, and welfare of the public, software engineers shall adhere to the following Eight Principles:

1. PUBLIC – Software engineers shall act consistently with the public interest.
2. CLIENT AND EMPLOYER – Software engineers shall act in a manner that is in the best interests of their client and employer consistent with the public interest.
3. PRODUCT – Software engineers shall ensure that their products and related modifications meet the highest professional standards possible.
4. JUDGMENT – Software engineers shall maintain integrity and independence in their professional judgment.
5. MANAGEMENT – Software engineering managers and leaders shall subscribe to 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.
7. COLLEAGUES – Software engineers shall be fair to and supportive of their colleagues.
8. SELF – Software engineers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession.

4. Explain Prototyping model of software development with a neat diagram. Discuss the different types of Prototype models.

   Prototyping is defined as the process of developing a working replication of a product or system that has to be engineered. It offers a small-scale facsimile of the end product and is used for obtaining customer feedback



Steps of Prototype Model

Step 1: Requirement Gathering and Analysis:This is the initial step in designing a prototype model. In this phase, users are asked about what they expect or what they want from the system. Step 2: Quick Design:This is the second step in Prototyping Model. This model covers the basic design of the requirement through which a quick overview can be easily described.

Step 3: Build a Prototype:This step helps in building an actual prototype from the knowledge gained from prototype design.

Step 4: Initial User Evaluation:This step describes the preliminary testing where the investigation of the performance model occurs, as the customer will tell the strength and weaknesses of the design, which was sent to the developer.

Step 5: Refining Prototype:If any feedback is given by the user, then improving the client's response to feedback and suggestions, the final system is approved.
Step 6: Implement Product and Maintain:This is the final step in the phase of the Prototyping Model where the final system is tested and distributed to production, here program is run regularly to prevent failures.

Types of prototyping models

1.RapidThrowawayPrototyping

This technique offers a useful method of exploring ideas and getting customer feedback for each of them. In this method, a developed prototype need not necessarily be a part of the ultimately accepted prototype. Customer feedback helps in preventing unnecessary design faults and hence, the final prototype developed is of better quality.

2.EvolutionaryPrototyping

In this method, the prototype developed initially is incrementally refined on the basis of customer feedback till it finally gets accepted. In comparison to Rapid Throwaway Prototyping, it offers a better approach that saves time as well as effort. This is because developing a prototype from scratch for every iteration of the process can sometimes be very frustrating for the developers.

3.IncrementalPrototyping

In this type of incremental Prototyping, the final expected product is broken into different small pieces of prototypes and developed individually. In the end, when all individual pieces are properly developed, then the different prototypes are collectively merged into a single final product in their predefined order.

The time interval between the project's beginning and final delivery is substantially reduced because all parts of the system are prototyped and tested simultaneously.

4. Extreme Prototyping

This method is mainly used for web development. It consists of three sequential independent phases:

In this phase, a basic prototype with all the existing static pages is presented in HTML format.

In the 2nd phase, Functional screens are made with a simulated data process using a prototype services layer.

This is the final step where all the services are implemented and associated with the final prototype.

This Extreme Prototyping method makes the project cycling and delivery robust and fast and keeps the entire developer team focused and centralized on product deliveries rather than discovering all possible needs and specifications and adding unnecessitated features.

5. Differentiate between Agile and Plan-driven methodologies

	Software Process Model	Advantages	Disadvantages
Plan Driven	• Waterfall • Incremental Development • Iterative development • Spiral Development • Prototype Model • Rapid Application Development	• Suitable for large systems and teams. • Handles highly critical systems effectively. • Appropriate for stable development environment. • Require experienced personnel at the beginning. • Success achieved through structure and order.	• Longer length in each iteration or increment. • Cannot accommodate changes any time. • Lack of user involvement throughout the life cycle of the product. • Costly for the dynamic development environment. • Assume that, future changes will not occur.
Agile	• Scrum model • Extreme Programming (XP) • Dynamic System Development Method • Kanban • Feature Driven Development	• Suitable for small to medium systems and teams. • Can accommodate changes at any time. • Effective for the dynamic development environment. • Required expert agile personnel throughout the life cycle. • Success achieved through freedom and chaos.	• Not suitable for large systems (except FDD). • Shorter length in each iteration. • Can accommodate changes at any time. • Costly for the stable development environment. • Assume that, frequent future changes will occur.

Comparison of the advantages and disadvantages of the plan-driven and agile processes

6. Explain in detail about extreme programming.

The most significant approach to changing software development culture was the development of Extreme Programming (XP). The name was coined by Kent Beck (Beck 1998) because the approach was developed by pushing recognized good practice, such as iterative development, to "extreme" levels. For example, in XP, several new versions of a system may be developed by different programmers, integrated, and tested in a day. In XP, requirements are expressed as scenarios (called user stories), which are implemented directly as a series of tasks. Programmers work in pairs and develop tests for each task before writing the code. All tests must be successfully executed when new code is integrated into the system. There is a short time gap between releases of the system. Extreme programming was controversial as it introduced a number of agile practices that were quite different from the development practice of that time.

ple or practice	Description
ctive ownership	The pairs of developers work on all areas of the system, so that no islands of expertise develop and all the developers take responsibility for all of the code. Anyone can change anything.
nuous ation	As soon as the work on a task is complete, it is integrated into the whole system. After any such integration, all the unit tests in the system must pass.
nental planning	Requirements are recorded on "story cards," and the stories to be included in a release are determined by the time available and their relative priority. The developers break these stories into development "tasks." See Figures 3.5 and 3.6.
te customer	A representative of the end-user of the system (the Customer) should be available full time for the use of the XP team. In an extreme programming process, the customer is a member of the development team and is responsible for bringing system requirements to the team for implementation.
rogramming	Developers work in pairs, checking each other's work and providing the support to always do a good job.
toring	All developers are expected to refactor the code continuously as soon as potential code improvements are found. This keeps the code simple and maintainable.
e design	Enough design is carried out to meet the current requirements and no more.
releases	The minimal useful set of functionality that provides business value is developed first. Releases of the system are frequent and incrementally add functionality to the first release.
inable pace	Large amounts of overtime are not considered acceptable, as the net effect is often to reduce code quality and medium-term productivity.
irst opment	An automated unit test framework is used to write tests for a new piece of functionality before that functionality itself is implemented.

7. What are Functional and Non-Functional Requirements? Explain the different types of Non-functional requirements.

Ans. Functional requirements

a. Statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations.
b. May state what the system should not do.

Non-functional requirements

c. Constraints on the services or functions offered by the system such as timing constraints, constraints on the development process, standards, etc.
d. Often apply to the system as a whole rather than individual features or services.

Functional Requirements	Non Functional Requirements
A functional requirement defines a system or its component.	A non-functional requirement defines the quality attribute of a software system.
It specifies "What should the software system do?"	It places constraints on "How should the software system fulfill the functional requirements?"
Functional requirement is specified by User.	Non-functional requirement is specified by technical peoples e.g. Architect, Technical leaders and software developers.
It is mandatory.	It is not mandatory.
It is captured in use case.	It is captured as a quality attribute.
Defined at a component level.	Applied to a system as a whole.
Helps you verify the functionality of the software.	Helps you to verify the performance of the software.
Functional Testing like System, Integration, End to End, API testing, etc are done.	Non-Functional Testing like Performance, Stress, Usability, Security testing, etc are done.
Usually easy to define.	Usually more difficult to define.
Example1) Authentication of user whenever he/she logs into the system. 2) System shutdown in case of a cyber attack. 3) A Verification email is sent to user whenever he/she registers for the first time on some software system.	Example1) Emails should be sent with a latency of no greater than 12 hours from such an activity. 2) The processing of each request should be done within 10 seconds 3) The site should load in 3 seconds when the number of simultaneous users are > 10000

8. Explain the format and characteristics of good SRS.
   Following are the characteristics of a good SRS document:

a. Correctness:

User review is used to ensure the correctness of requirements stated in the SRS. SRS is said to be correct if it covers all the requirements that are actually expected from the system.

b. Completeness:

Completeness of SRS indicates every sense of completion including the numbering of all the pages, resolving the to be determined parts to as much extent as possible as well as covering all the functional and non-functional requirements properly.

c. Consistency:

Requirements in SRS are said to be consistent if there are no conflicts between any set of requirements. Examples of conflict include differences in terminologies used at separate places, logical conflicts like time period of report generation, etc.

d. Unambiguousness:

A SRS is said to be unambiguous if all the requirements stated have only 1 interpretation. Some of the ways to prevent unambiguousness include the use of modelling techniques like ER diagrams, proper reviews and buddy checks, etc.

e. Ranking for importance and stability:

There should a criterion to classify the requirements as less or more important or more specifically as desirable or essential. An identifier mark can be used with every requirement to indicate its rank or stability.

f. Modifiability:

SRS should be made as modifiable as possible and should be capable of easily accepting changes to the system to some extent. Modifications should be properly indexed and cross-referenced.

g. Verifiability:

A SRS is verifiable if there exists a specific technique to quantifiably measure the extent to which every requirement is met by the system. For example, a requirement stating that the system must be user-friendly is not verifiable and listing such requirements should be avoided.

h. Traceability:

One should be able to trace a requirement to design component and then to code segment in the program. Similarly, one should be able to trace a requirement to the corresponding test cases.

i. Design Independence:

There should be an option to choose from multiple design alternatives for the final system. More specifically, the SRS should not include any implementation details.

j. Testability:

A SRS should be written in such a way that it is easy to generate test cases and test plans from the document.

k. Understandable by the customer:

An end user maybe an expert in his/her specific domain but might not be an expert in computer science. Hence, the use of formal notations and symbols should be avoided to as much extent as possible. The language should be kept easy and clear.

l. Right level of abstraction:

If the SRS is written for the requirements phase, the details should be explained explicitly. Whereas, for a feasibility study, fewer details can be used. Hence, the level of abstraction varies according to the purpose of the SRS.

Structure of SRS:

Introduction :

(i) Purpose of this Document –

At first, main aim of why this document is necessary and what's purpose of document is explained and described.
(ii) Scope of this document –
In this, overall working and main objective of document and what value it will provide to customer is described and explained. It also includes a description of development cost and time required.
(iii) Overview –
In this, description of product is explained. It's simply summary or overall review of product.

General description :
In this, general functions of product which includes objective of user, a user characteristic, features, benefits, about why its importance is mentioned. It also describes features of user community.

Functional Requirements :
In this, possible outcome of software system which includes effects due to operation of program is fully explained. All functional requirements which may include calculations, data processing, etc. are placed in a ranked order.

Interface Requirements :
In this, software interfaces which mean how software program communicates with each other or users either in form of any language, code, or message are fully described and explained.
Examples can be shared memory, data streams, etc.

Performance Requirements :
In this, how a software system performs desired functions under specific condition is explained. It also explains required time, required memory, maximum error rate, etc.

Design Constraints :
In this, constraints which simply means limitation or restriction are specified and explained for design team.
Examples may include use of a particular algorithm, hardware and software limitations, etc.

Non-Functional Attributes :
In this, non-functional attributes are explained that are required by software system for better performance.
An example may include Security, Portability, Reliability, Reusability, Application compatibility, Data integrity, Scalability capacity, etc.

Preliminary Schedule and Budget :
In this, initial version and budget of project plan are explained which include overall time duration required and overall cost required for development of project.

Appendices :
In this, additional information like references from where information is gathered, definitions of some specific terms, acronyms, abbreviations, etc. are given and explained.

9. Define Requirements Engineering. Explain the activities of requirement engineering process**.**

Requirement Engineering is the process of defining, documenting and maintaining the requirements.

The requirements engineering process is an iterative process that involves several steps, including:

- **Requirements Elicitation**: This is the process of gathering information about the needs and expectations of stakeholders for the software system. This step involves interviews, surveys, focus groups, and other techniques to gather information from stakeholders.
- **Requirements Analysis**: This step involves analyzing the information gathered in the requirements elicitation step to identify the high-level goals and objectives of the software system. It also involves identifying any constraints or limitations that may affect the development of the software system.
- **Requirements Specification**: This step involves documenting the requirements identified in the analysis step in a clear, consistent, and unambiguous manner. This step also involves prioritizing and grouping the requirements into manageable chunks.
- **Requirements Validation**: This step involves checking that the requirements are complete, consistent, and accurate. It also involves checking that the requirements are testable and that they meet the needs and expectations of stakeholders.
- **Requirements Management**: This step involves managing the requirements throughout the software development life cycle, including tracking and controlling changes, and ensuring that the requirements are still valid and relevant.

10. Explain the different Techniques for Requirement validation?

Requirements reviews

11. Systematic manual analysis of the requirements. This technique involves reviewing the requirements document with a group of experts, looking for errors, inconsistencies, and missing information.
12. Regular reviews should be held while the requirements definition is being formulated.
13. Both client and contractor staff should be involved in reviews.
14. Reviews may be formal (with completed documents) or informal

Prototyping

Using an executable model of the system to check requirements. This technique involves creating a working prototype of the system and testing it to see if it meets the requirements.

Test-case generation

a. Developing tests for requirements to check testability. If the test is difficult or impossible to design, this usually means that the requirements will be difficult to implement and should be reconsidered.