

Unit 1

2mark:

1) Define Software Engineering.

Software engineering is an engineering discipline that is concerned with all aspects of software production from the early stages of system specification to maintaining the system after it has gone into use.

2) What is software?

software is not just the programs but also all associated documentation and configuration data that is needed to make these programs operate correctly. Software products may be developed for a particular customer or may be developed for a general market.

3) Mention any two differences between software Engineering and System Engineering.

Feature	Software Engineering	Systems Engineering
Focus	Software applications	Complex systems
Approach	Structured and disciplined	Flexible and adaptable
Focus domains.	developing good software.	users and

4) Define software Process.

A set of activities whose goal is the development or evolution of software. There are four fundamental process activities :: Software specification ,Software development ,Software validation and Software evolution

5) What is Software Process Model?

A simplified representation of a software process, presented from a specific perspective. Process models may include activities that are part of the software process, software products and the roles of people involved in software engineering. types are dataflow and workflow model.

6) What is CASE?

The CASE stands for Computer-Aided Software Engineering. It covers a wide range of different types of programs that are used to support software process activities. CASE systems are often used for method support.

7) Give any two attributes of good software

Maintainability, Dependability, Efficiency and Usability

8) What is Waterfall Model?

the cascade from one phase to another, this model is known as the waterfall model or software life cycle. This was the first process model, here each phase must be completed before the next phase begins and there is no overlapping in the phase. Documentation is produced at each phase.

9) What is Evolutionary Development?

Evolutionary development is an approach to software development that involves creating an initial version of a system, exposing it to user feedback, and then refining it through multiple versions until a satisfactory system is achieved.

10) Define component based Software Engineering.

It is a process that focuses on design and development of CBS with the use of reusable software components. (reusable, easy to use, reduced cost)

11) Give any two advantages of Incremental Development Process.

- Customers do not have to wait until the entire system is delivered before they can gain value from it.
- There is a lower risk of overall project failure.
- It is easy to manage each iteration.
- It is easy to manage risk because of iterations.

12) Define Software specification.

Software specification or requirements engineering is the process of understanding and defining what services are required from the system and identifying the constraints on the system's operation and development.

13) What is Component (or unit) testing?

Component testing (sometimes called unit testing) is the process of testing individual components in the system. Unit testing fixes defects very early in the development phase.

14) What is System testing?

System testing is a type of software testing that evaluates the overall functionality and performance of a complete and fully integrated software solution. It tests if the system meets the specified requirements and if it is suitable for delivery to the end-users.

15) What is Acceptance testing?

Acceptance Testing is the last phase of software testing performed after System Testing . Acceptance Testing is the process of verifying if a product or system meets the customer's requirements and expectations. before making the system available for actual use.

16) What is requirement engineering?

Requirements engineering (RE) is a systematic process for defining, analysing ,documenting, and maintaining requirements in the engineering design process. It's a common role in systems engineering and software engineering.

17) What are Functional and non-functional requirement?

Functional requirements explain how the system must work, while non functional requirements explain how the system should perform.

18) What is SRS?

A software requirements specification (SRS) is a document that captures complete description about how the system is expected to perform. It includes information about all the functional and non-functional requirements for a given piece of software.

19) What is structured language specification?

Structured natural language is a way of writing system requirements where the freedom of the requirements writer is limited and all requirements are written in a standard way(structured form).

4mark:

1) What is software? Explain two types of software products.

Computer programs and associated documentation. Software products may be developed for a particular customer or may be developed for a general market is called software.

Generic products :in software engineering are stand-alone systems that are produced by a development organization and sold on the open market to any customer who is able to buy them. Large number of users may be using this kind of software. They are developed for general use and can be used by multiple customers or industries. example are databases, word processors, drawing packages and project management tools.

Customised products: a customized product is a software solution created specifically for a particular organization or individual. This type of software is used by limited number of users. Examples of this type of software include control systems for electronic devices and air traffic control systems.

2) What is Software Process? Explain Fundamental activities in Software Process.

A set of activities whose goal is the development or evolution of software is called software process.

1. **Software specification** where customers and engineers define the software to be produced and the constraints on its operation.
2. **Software development** where the software is designed and programmed.
3. **Software validation** where the software is checked to ensure that it is what the customer requires.
4. **Software evolution** where the software is modified to adapt it to changing customer and market requirements.

3) Explain attributes of good software?

Maintainability : Software should be written in such a way that it may evolve to meet the changing needs of customers. This is a critical attribute because software change is an inevitable consequence of a changing business environment.

Dependability: Dependability is the most important facet of a computer system. It is a measure of a system's ability to provide services that can be trusted within a time-period. including reliability, security and safety.

Efficiency: It also refers to the ability of the software to use system resources in the most effective and efficient manner. Efficient software is like a smart worker that doesn't waste time or resources. It operates smoothly, using just the right amount of memory and processing power needed to get the job done. Imagine a well-organized office where tasks are completed without unnecessary delays.

Usability: it should be easy to use without unnecessary struggle. Just like a well-designed road map, usable software comes with an appropriate user interface, guiding users smoothly through its features.

4) Explain Key challenges facing software engineering.

Heterogeneity challenge: is a composition of a software system which is made up of different languages and running on different systems by using different standard for communication

delivery challenge :businesses today must be responsive and change very rapidly. Their supporting software must change equally rapidly. The delivery challenge is the challenge of shortening delivery times for large and complex systems without compromising system quality.

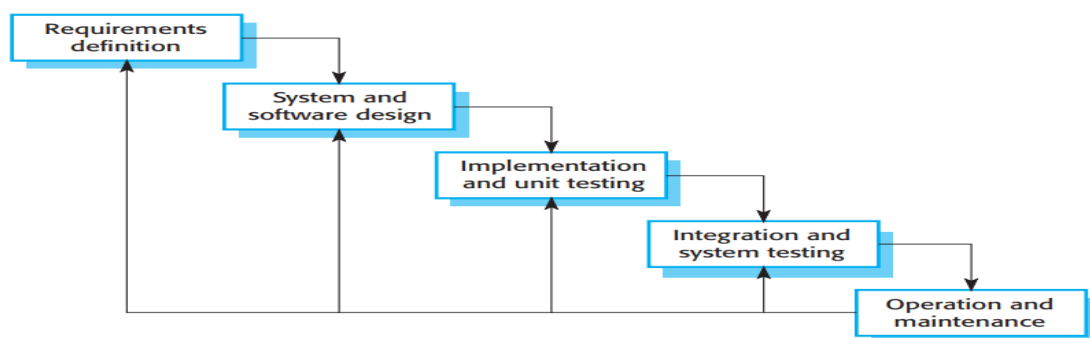
trust challenge :in software engineering is the challenge of developing a technique that demonstrates that software can be trusted by its user. It's important that we trust the software we are using, especially for those software systems which are accessed through a web page or web service.

5) Write a note on Professional and ethical responsibilities.

Software engineers should always aim to produce high-quality work and continuously improve their skills and knowledge. Maintain confidentiality: Engineers should respect the privacy of their clients and users, and protect any

1. **Confidentiality:** whether or not you've signed a formal agreement, it's just good practice to keep things confidential. You should normally respect the confidentiality of your employers or clients.
2. **Competence:** it's important not to pretend to be an expert in something you're not. Taking on tasks you're not competent in can lead to problems down the road.
3. **Intellectual property rights** :You should be aware of local laws governing the use of intellectual property such as patents and copyright. You should be careful to ensure that the intellectual property of employers and clients is protected
4. **Computer misuse:** *You should not use your technical skills to misuse other people's computers. Computer misuse ranges from relatively trivial (game playing on an employer's machine, say) to extremely serious (dissemination of viruses)*

6) Explain waterfall Model with a neat diagram



1. **Requirements analysis and definition** The system's services, constraints and goals are established by consultation with system users. They are then defined in detail and serve as a system specification.
2. **System and software design** The systems design process partitions the requirements to either hardware or software systems. It establishes an overall

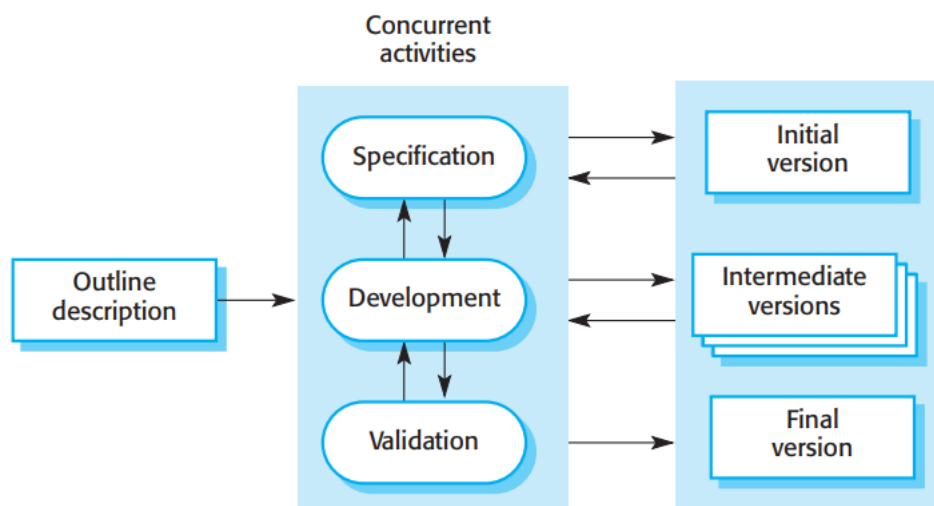
system architecture. Software design involves identifying and describing the fundamental software system abstractions and their relationships.

3. Implementation and unit testing During this stage, the software design is realised as a set of programs or program units. Unit testing involves verifying that each unit meets its specification.

4. Integration and system testing The individual program units or programs are integrated and tested as a complete system to ensure that the software requirements have been met. After testing, the software system is delivered to the customer.

5. Operation and maintenance this is the longest life-cycle phase. The system is installed and put into practical use. Maintenance involves correcting errors which were not discovered in earlier stages of the life cycle, improving the implementation of system units and enhancing the system's services as new requirements are discovered.

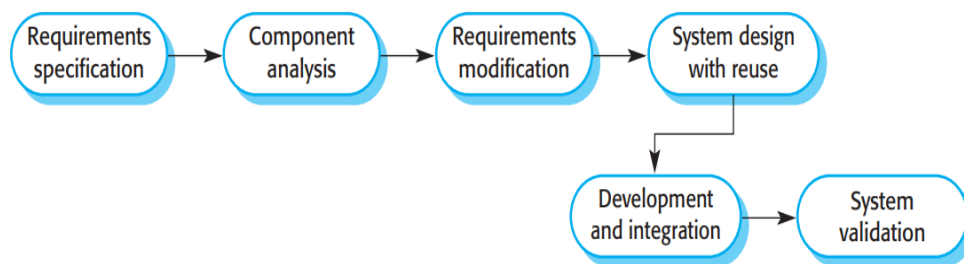
7) With a diagram, explain evolutionary Development.



- **Specification:** The specification activity defines the requirements for the software system. This includes the functional requirements (what the system should do), the non-functional requirements (how the system should perform), and the constraints (limitations on the system).
- **Design:** The design activity develops the architecture and design of the software system. This includes defining the components of the system, the interfaces between the components, and the internal logic of each component.
- **Implementation:** The implementation activity implements the software system in a programming language. This involves writing code to implement the design of the system.

- **Validation:** The validation activity tests the software system to ensure that it meets the requirements. This includes unit testing, integration testing, system testing, and acceptance testing.
- **Deployment:** The deployment activity deploys the software system to the target environment. This may involve installing the software on servers, making it available for download, or integrating it with other systems.

8) Explain Component based Software Engineering with diagram.



1. **Component analysis** Given the requirements specification, a search is made for components to implement that specification.

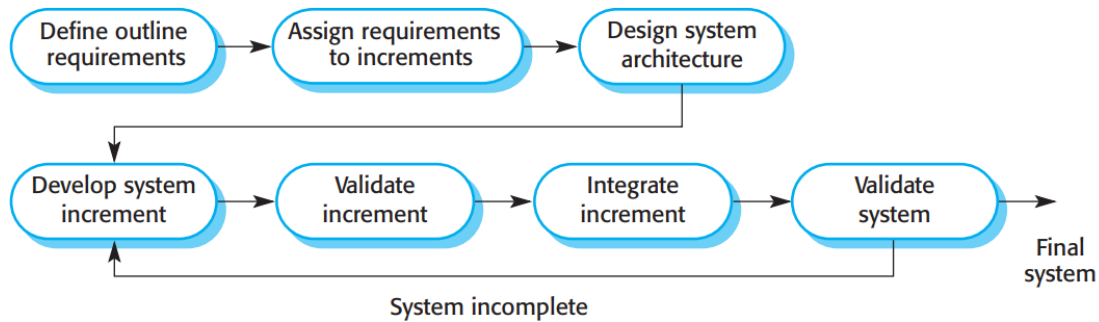
2. **Requirements modification** During this stage, the requirements are analysed using information about the components that have been discovered.

3. **System design with reuse** During this phase, the framework of the system is designed or an existing framework is reused.

4. **Development and integration** Software that cannot be externally procured is developed, and the components and COTS systems are integrated to create the new system.

9) Explain Incremental delivery with a diagram.

The Incremental Development model is an iterative approach to software development in which the system is developed and delivered in increments. The Incremental Development model is well-suited for complex systems where the requirements are not fully known at the beginning of the project. It allows for early feedback from users and stakeholders, and it reduces the risk of developing a system that does not meet the needs of the users.

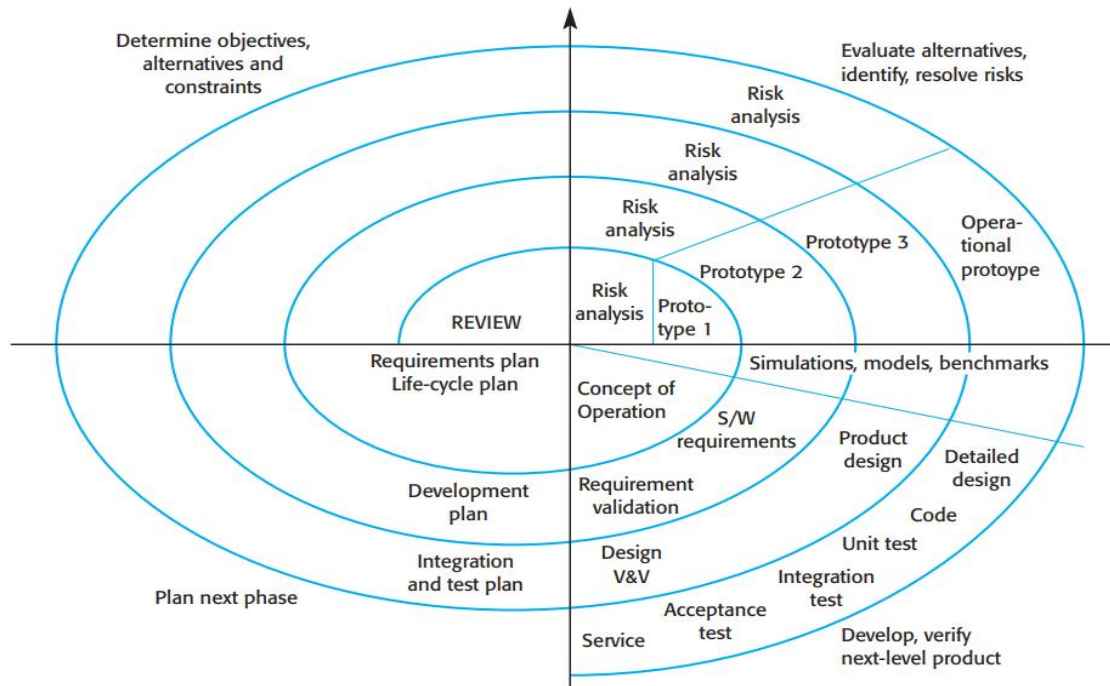


1. **Assign requirements to increments:** The requirements are assigned to different increments, which are smaller pieces of work that can be developed and deployed independently.
2. **Design system architecture:** The architecture of the system is designed, which defines the overall structure and components of the system.
3. **Develop system increment:** The system increment is developed, which involves implementing the features and functionality of the increment.
4. **Validate system increment:** The system increment is tested to ensure that it meets the requirements.
5. **Integrate increment:** The system increment is integrated with the rest of the system.
6. **Validate system:** The system is tested to ensure that it meets all of the requirements.

10) Explain Spiral model with diagram.

The spiral model of the software process was originally proposed by Boehm (Boehm, 1988) the process is represented as a spiral. Each loop in the spiral represents a phase of the software process.

Thus, the innermost loop might be concerned with system feasibility, the next loop with requirements definition, the next loop with system design and so on.



objective setting phase of the spiral model, the project objectives are defined in detail. Constraints on the project timeline, budget, and resources are also identified. A detailed management plan is drawn up to ensure that the project is completed on time and within budget. Requirements are gathered from the customers and the objectives are identified, elaborated, and analyzed at the start of every phase. Then alternative solutions possible for the phase are proposed in this quadrant.

risk assessment and reduction During the second quadrant, all the possible solutions are evaluated to select the best possible solution. Then the risks associated with that solution are identified and the risks are resolved using the best possible strategy. At the end of this quadrant, the Prototype is built for the best possible solution.

Development and validation During the third quadrant, the identified features are developed and verified through testing. At the end of the third quadrant, the next version of the software is available.

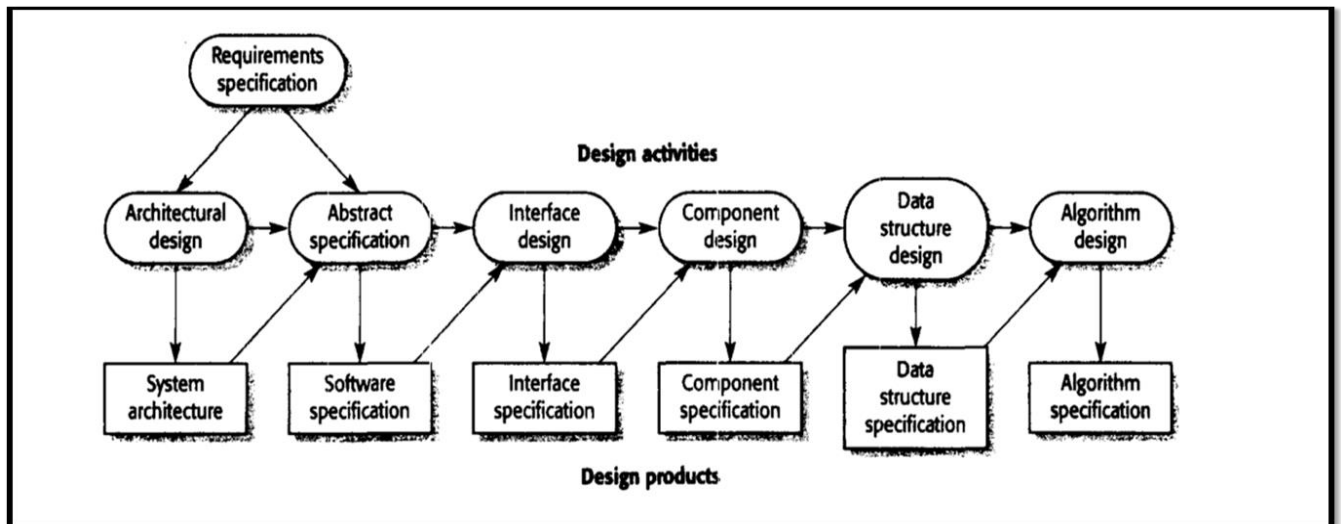
Planning The project is reviewed and a decision made whether to continue with a further loop of the spiral. If it is decided to continue, plans are drawn up for the next phase of the project.

11) Explain four phases of requirement Engineering Process.

Software specification or requirements engineering is the process of understanding and defining what services are required from the system and identifying the constraints on the system's operation and development.

1. **Feasibility study** A feasibility study is an initial assessment of the practicality of a proposed project or system. It evaluates whether the project can be successfully completed within the constraints of available resources, budget, and time. The study also considers the potential benefits and drawbacks of the project, as well as the risks involved. The results of the feasibility study inform the decision of whether to proceed with further development.
2. **Requirements elicitation and analysis** Requirements elicitation and analysis is the process of gathering and understanding the needs of stakeholders for a new or existing system. This involves a variety of techniques, such as interviews, surveys, and observation. The goal is to identify and document all of the requirements for the system, so that the developers can build a system that meets the needs of the users
3. **Requirements specification two types are:** **User requirements** are high-level statements that describe what the user wants or needs from the system. **System requirements** are more detailed specifications that describe how the system should meet those user needs.
For example, a user requirement might be "The system should allow me to create and manage my calendar." A system requirement for this might be "The system shall provide a calendar interface that allows users to create, edit, and delete events."
4. **Requirements validation** Requirements validation is a critical step in the software development process, ensuring that the specified requirements accurately reflect the needs and expectations of stakeholders. It involves checking the requirements for realism, consistency, and completeness. During this process, errors in the requirements document are inevitably discovered. These errors must be identified, analyzed, and corrected to ensure that the final product meets the intended purpose.

12) With diagram, explain the specific design process activities of software design.



architectural design involves identifying and documenting the different subsystems that make up a system and how they interact with each other. This helps to ensure that the system is well-organized, easy to understand, and can be developed efficiently and effectively.

abstract specification is a valuable tool for software engineers because it helps to ensure that the system is designed correctly. The abstract specification can be used to identify and resolve potential problems early in the design process. The abstract specification can also be used to generate test cases that can be used to verify that the system has been implemented correctly.

Interface design is the process of creating interfaces for software and computerized devices. The goal is to create interfaces that are easy to use and pleasurable. A good interface makes using the device easy, efficient, and safe.

Component design is the process of breaking down a software system into smaller, reusable pieces called components. Each component has a specific function and provides a set of services to other components. The interfaces between components are carefully designed to ensure that they can communicate effectively with each other.

Data structure design is an essential part of software engineering, as it determines how data is organized and manipulated within a program. The data structures used in the system implementation are designed in detail and specified.

Algorithm design is a crucial aspect of software engineering that involves creating efficient and optimized step-by-step procedures to solve specific computational problems. It is the process of transforming problem requirements into a sequence of instructions for a computer to execute.

13) Write a note on Rational Unified Process

Rational Unified Process (RUP) is a software development process for object-oriented models. It is also known as the Unified Process Model. It is created by Rational corporation and is designed and documented using UML (Unified Modeling Language).

the phases of rup are:

1. Inception –

- Communication and planning are the main ones.
- Identifies the scope of the project using a use-case model allowing managers to estimate costs and time required.
- Customers' requirements are identified and then it becomes easy to make a plan for the project.
- The project is checked against the milestone criteria and if it couldn't pass these criteria then the project can be either canceled or redesigned.

2. Elaboration –

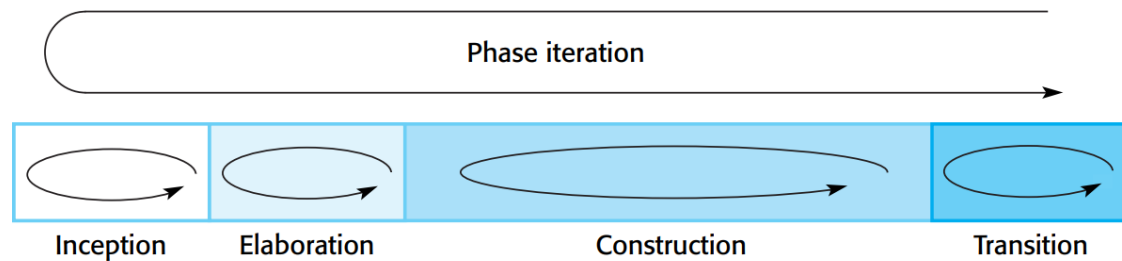
- Planning and modeling are the main ones.
- A detailed evaluation and development plan is carried out and diminishes the risks.
- Revise or redefine the use-case model , business case, and risks.
- Again, checked against milestone criteria and if it couldn't pass these criteria then again project can be canceled or redesigned.

3. Construction –

- The project is developed and completed.
- System or source code is created and then testing is done.
- Coding takes place.

4,Transition –

- The final project is released to the public.
- Transit the project from development into production.
- Update project documentation.
- Beta testing is conducted.
- Defects are removed from the project based on feedback from the public.



14) Explain Functional and Non-functional requirement.

Functional requirements should be written in a simple language, so that it is easily understandable. The examples of functional requirements are authentication, business rules, audit tracking, certification requirements, transaction corrections, etc.

Basic non-functional requirements are - usability, reliability, security, storage, cost, flexibility, configuration, performance, legal or regulatory requirements, etc.

Functional Requirements	Non-functional requirements
Functional requirements help to understand the functions of the system.	They help to understand the system's performance.
Functional requirements are mandatory.	While non-functional requirements are not mandatory.
They are easy to define.	They are hard to define.
They describe what the product does.	They describe the working of product.
It concentrates on the user's requirement.	It concentrates on the expectation and experience of the user.
It helps us to verify the software's functionality.	It helps us to verify the software's performance.
These requirements are specified by the user.	These requirements are specified by the software developers, architects, and technical persons.

There is functional testing such as API testing, system, integration, etc.	There is non-functional testing such as usability, performance, stress, security, etc.
--	--

15) Explain structure of Requirement document

A requirement document is a formal document that outlines the functional and non-functional requirements of a software system. It serves as a contract between the client and the developer, ensuring that the system meets the client's needs and expectations.

1. **Introduction:** This section provides an overview of the project, including its purpose, scope, and stakeholders.
2. **Overall Description:** Describes the software system in general terms, including its main functions and features.
3. **Specific Requirements:** Detailed specifications for each feature or function, including input parameters, output results, and behavioral constraints.
4. **System Requirements:** Specifies the technical and environmental requirements for the system, such as hardware, software, and network requirements.
5. **Non-Functional Requirements:** Describes non-functional aspects of the system, such as performance, security, usability, and maintainability.
6. **Appendices:** Contains supporting information, such as diagrams, glossaries, and references.

16) Write a note on agile methods.

The meaning of Agile is swift or versatile. "Agile process model" refers to a software development approach based on iterative development. Agile methods break tasks into smaller iterations, or parts do not directly involve long term planning.

The principles of agile method

Customer involvement Customers should be closely involved throughout the development process. Their role is provide and prioritize new system requirements and to evaluate the iterations of the system.

Incremental delivery The software is developed in increments with the customer specifying the requirements to be included in each increment.

People not process This means that the skills and experience of the team are valued more than rigid rules and procedures. Teams are encouraged to

find their own ways of working that are efficient and effective. This approach leads to a more adaptable and responsive team that is better able to deliver value to customers.

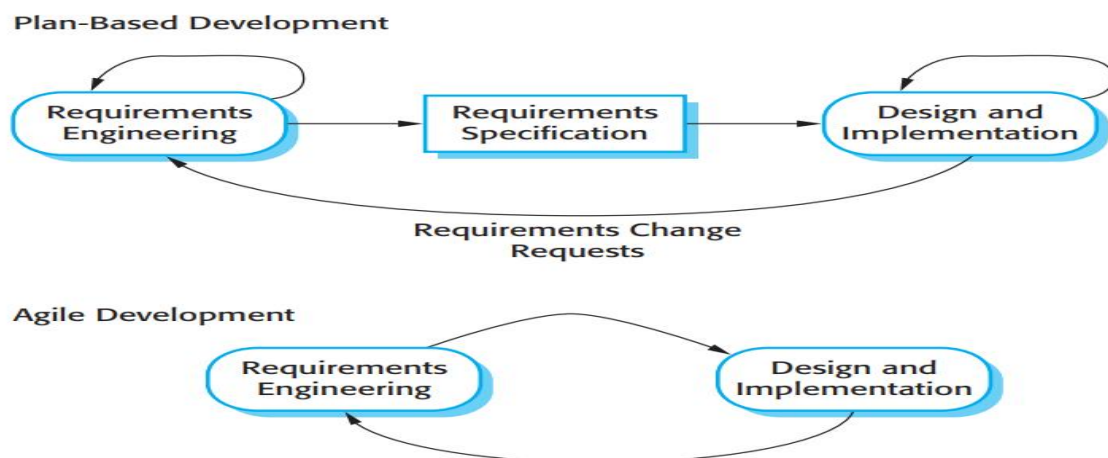
Embrace the change :This means expecting and accepting that requirements will change throughout the project lifecycle. Instead of trying to resist change, agile teams should embrace it as an opportunity to improve the product. In fact, they see change as an opportunity to learn and improve.

Maintain simplicity Focus on simplicity in both the software being developed and in the development process. Wherever possible, actively work to eliminate complexity from the system

17) With diagram explain Plan-driven and agile Specification.

Plan-driven specification is a traditional approach to requirements gathering and documentation. It involves creating a comprehensive requirements document (CRD) upfront, which outlines all of the functional and non-functional requirements of the system. The CRD is then used as a blueprint for development, and changes to the requirements are typically discouraged.

Agile specification is a more iterative and incremental approach to requirements gathering. It involves working with customers and stakeholders to gather and refine requirements throughout the development process. This approach allows for more flexibility and adaptability, as requirements can be changed as needed.



The diagram shows how plan-driven and agile development differ in terms of requirements engineering. In plan-driven development, requirements are documented and frozen early on, and any changes to requirements must be managed through a formal change request process. In agile development, requirements are gathered and refined throughout the development process, and changes to requirements are accommodated through the use of short iterations.

The diagram also shows how plan-driven and agile development differ in terms of design and implementation. In plan-driven development, design and implementation are typically sequential processes, with design completed before implementation begins. In agile development, design and implementation are typically iterative and incremental processes, with design and implementation happening concurrently

Unit 2

2 mark:

1) Define the term requirement elicitation and analysis

This is the process of deriving the system requirement through observation of existing system, discussion with potential user, task analysis and so on. this may involve the development of one or more system models and prototypes. These help you understand the system to be specified.

2) Define the term stakeholder. Mention the different categories of stakeholders

Stakeholder is referred as any person or group who will be affected by the system directly or indirectly . stakeholders include end user who interact with the system and everyone else in the organisation that may affected by its installation.

Customer, employee, vendor and creditor.

3) What is meant by Requirement Discovery and Requirement documentation?

Requirement discovery: This is the process of interacting with stakeholders in the system to collect their requirements.

Requirements documentation The requirements are documented and input into the next round of the spiral. Formal or informal requirements documents may be produced.

4) What is Requirement classification and Requirement prioritisation?

Requirements classification and organization is a crucial step in the requirements engineering process, as it helps to transform the unstructured collection of requirements into a well-structured and organized form. This structured form makes it easier to understand, analyze, prioritize, and track requirements throughout the development process.

Requirements prioritization is the process of assigning a priority level to each requirement in a project, typically based on its importance to the project's goals and objectives. This helps to ensure that the most important requirements are addressed first, and that the project delivers the most value to the stakeholders.

5) Differentiate Validity checks and Consistency checks

Feature	Validity Checks	Consistency Checks
Purpose	Ensure data accuracy and correctness	Ensure data is internally consistent
Focus	Individual data points or values	Relationships between data points or values
Methods	Data validation rules, range checks, type checks, cross-referencing	Data logic checks, data integrity checks, business rules checks
Examples	Checking date format, verifying email addresses, ensuring valid age ranges	Checking order dates vs. shipping dates, verifying product prices, ensuring valid grade ranges

6) Write the difference between completeness checks and realism checks

Feature	Completeness Checks	Realism Checks
Focus	Identifying missing requirements	Assessing feasibility of requirements
Timing	Early in the requirements gathering process	After initial requirements are identified
Goal	Ensure all necessary requirements are captured	Ensure requirements can be implemented within project constraints
Methods	Document review, stakeholder interviews, brainstorming	Technology evaluation, resource assessment, timeline analysis
Outcome	Complete set of requirements	Assessment of requirements feasibility

7) Define verifiability in requirement validation.

Verifiable requirements are crucial for ensuring clear communication and expectations between the parties involved in a software development project. By clearly defining what the system should do and how it should behave, verifiable requirements help to prevent misunderstandings and ensure that the final product meets the customer's needs.

8) What is test-case generation in the requirement validation process?

Test case generation is the process of creating test cases, which are inputs to a software system that are designed to test its behavior and ensure that it meets its requirements. Test cases are typically written in a structured format, such as a table or a list, and they should include a description of the test, the expected outcome, and the actual outcome.

- **Black-box testing:** This method involves testing the software system without knowing its internal workings. The tester creates test cases based on the system's requirements and specifications.
- **White-box testing:** This method involves testing the software system with knowledge of its internal workings. The tester creates test cases based on the code itself.

9) Differentiate formal requirement review and informal requirement review

Feature	Formal Requirements Review	Informal Requirements Review
Structure	Structured and planned	Unstructured and ad hoc
Participants	Typically includes a team of reviewers, stakeholders, and project managers	Typically includes a small group of stakeholders or project team members
Process	Follows a defined process with clear objectives and deliverables	Does not follow a defined process and may have more flexible objectives
Documentation	Produces formal documentation of findings and recommendations	May not produce formal documentation

10) Define the terms Requirement identification and Traceability policy.

Requirement identification is the process of identifying all of the requirements for a software system. This involves gathering information from stakeholders, analyzing existing systems, and reviewing project documentation. Requirements can be functional, non-functional, or a combination of both.

Requirement traceability is the process of linking requirements to their implementation in the software system. This involves tracking the requirements through the development process, from design to coding to testing. Requirement traceability helps to ensure that all requirements are implemented correctly and that changes to requirements are reflected in the software system.

11) What are Mutable Requirements? Give two examples

Mutable requirements are requirements that can change over time. This is typically due to changes in the environment in which the system is operating.

Examples of Mutable Requirements

- **A hospital's billing system:** The requirements for a hospital's billing system may change if the government changes its healthcare regulations or if the hospital adopts a new pricing strategy.
- **A school's online learning platform:** The requirements for a school's online learning platform may change if the school adopts new curriculum standards or if the school's student population changes.

12) What are Emergent requirements and Compatibility Requirements.

Emergent requirements are requirements that arise during the development process as a result of interactions between the system, the environment, and the users. These requirements can be difficult to predict in advance, and they often require creative and innovative solutions.

Compatibility requirements are requirements that specify how the system must interact with other systems or components. These requirements can be related to hardware, software, or data formats.

13) What is Behavioural model? Mention the types

It is used to describe the overall behaviour of system .

data flow model in which model the data processing in the system.

state machine model in which model how the system react to event.

14) What is Data-flow model? List the notations used in Data-flow model

It used to show how data flows through a sequences of processing steps.



15) Which notations are used for state machine modelling? Give its usage.

The rounded rectangles in a model represent system states. They include a brief description of the actions taken in that state.

The labelled arrows represent stimuli that force a transition from one state to another

16) What is ERA modelling? Why it is used?

Entity,relation and attribute is a data modelling technique that describes the relationship between different entities in a system.

ERA are widely used in database design and systems analysis to capture requirements of a system.

17) What is data dictionary? Mention the advantages of using data dictionary

It is useful when developing a system model and may be used to manage all the information from all type of system models.

- Improved efficiency
- Improved collaboration
- Data quality

18) What is object model? List various object models.

object model is an abstract representation of a system's objects, their attributes, their relationships, and their behaviors. It serves as a blueprint for designing and implementing object-oriented software.

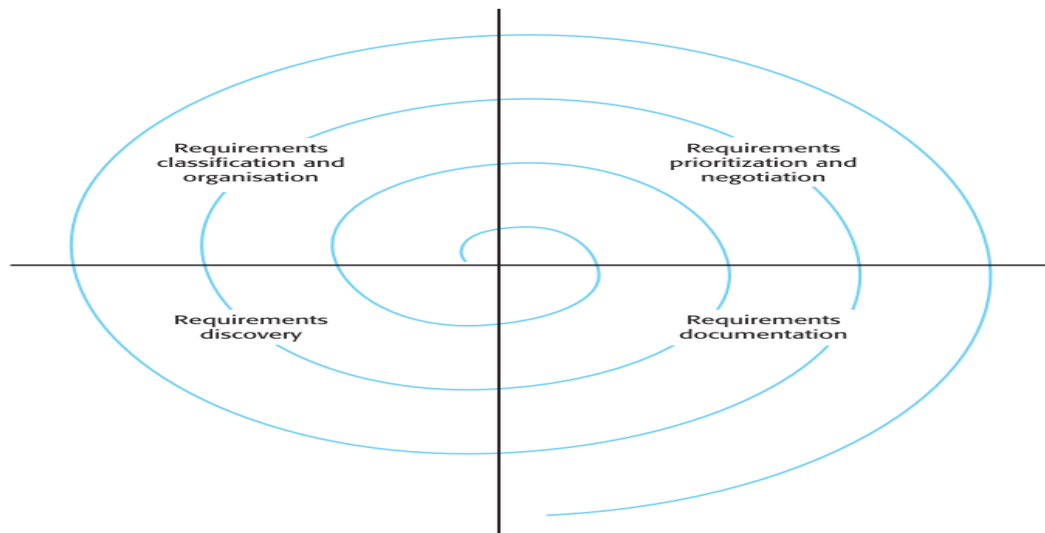
Inheritance models, Object aggregation and Object behaviour modelling

4 mark :

1) Explain the reasons which cause difficulty in understanding stakeholder requirements.

1. Stakeholder don't know what they want from computer system they may find it difficult to explain what they want the system to do and they make unrealistic demand because they don't understand what is possible
2. Stakeholders use their own words and knowledge to describe what they want from system requirement engineer who don't know the stakeholder business may not understand this requirement.
3. Different stakeholders have different requirement and they may express this in different way. requirement engineer find all possible source of need and identify similarity and conflicts in srs
4. Political factor can affect to what a system needs to do manager may ask for specific requirement because they want to have more power in organisation
5. The economy and business world are constantly changing so the requirement for a system may change during analysis process this means some requirement may become more or less important and new requirement may come up from stakeholder who were not consulted at first

2) Explain the general process model of the elicitation and analysis process with the diagram?



Requirement discovery: This is the process of interacting with stakeholders in the system to collect their requirements. It is an important step in the system development life cycle to ensure that the system will meet the needs of users and other stakeholders.

Requirements classification and organization is a crucial step in the requirements engineering process, as it helps to transform the unstructured collection of requirements into a well-structured and organized form and grouping related requirements together. This helps to make requirements easier to understand, manage, and prioritize.

Requirement prioritization and negotiation: When multiple stakeholders are involved, requirements may conflict. It is a process of ranking requirements in order of importance and resolving conflicts through negotiation to ensure the system will meet the important needs of stakeholders and that requirements are feasible to implement within the project.

Requirement documentation: The requirements are documented and input into the next round of the spiral. Formal or informal requirements documents may be produced. This is important in the development process as it ensures that requirements are well-defined and understood by all stakeholders.

3) What is requirement validation? Explain the various validation process checks to be carried on the requirements document.

Requirement validation is the process of confirming that requirements are complete, correct, and consistent. It also ensures that requirements are in line with the customer's needs. It is important to validate the requirements early in the development process as the error in the requirements can lead to costly rework or even project failure.

Validity checks are a crucial step in ensuring that the specified requirements are accurate, complete, consistent, and unambiguous. They help to identify

and eliminate any errors or inconsistencies in the requirements early on, preventing them from causing problems later in the development process.

Consistency checks are a crucial aspect of software requirements specification (SRS) to ensure that the specified requirements are consistent with each other and do not conflict with any other specifications or assumptions.

Completeness checks are a crucial aspect of software requirements specification (SRS) to ensure that all necessary requirements for the system are captured and adequately documented. These checks aim to identify and eliminate any missing requirements that may impact the system's functionality or user experience.

Realism check: A realism check in Software Requirement Specifications (SRS) ensures that requirements can be implemented using existing technology, budget, and schedule. The goal of this process is to find errors in the requirements document

Verifiability: to reduce the potential for dispute between customer and contractor, system requirements should always be written so that they are verifiable.

4) List and explain requirement validation techniques.

Requirements reviews are an essential part of the software development lifecycle (SDLC). They help to ensure that the software meets the needs of the users and stakeholders, and that it is developed in a way that is consistent with the project's goals and objectives.

Prototyping is an iterative process of developing a working model of a software system to gather user feedback and refine the requirements before committing to a full-scale development effort. It is a valuable tool in the software requirements specification (SRS) process, as it allows stakeholders to visualize and interact with the system early on, leading to better understanding, improved requirements, and reduced risk of project failure.

Test-case generation Test case generation is the process of creating test cases, which are inputs to a software system that are designed to test its behavior and ensure that it meets its requirements. Test cases are typically written in a structured format, such as a table or a list, and they should include a description of the test, the expected outcome, and the actual outcome.

5) Briefly explain the concept of requirements review along with reviewer checks.

A requirement review is a formal process of evaluating and verifying the completeness, accuracy, and clarity of software requirements. It is an

essential part of the software development lifecycle (SDLC) and helps to ensure that the final product meets the needs of the users and stakeholders.

Verifiability Ability to determine whether or not a requirement has been met requirement is verifiable if there is a clear and objective way to test it?

Comprehensibility: ensure their system will meet the need for stakeholder if the stakeholder don't know understand the requirement they cannot provide feedback.

Traceability: Is the origin of the requirement clearly stated? .Ability to track the origin of requirement and to see how it is used in the system because it allows stakeholder to understand the impact of change to recruitment on the rest of the system.

Adaptability: Is the requirement adaptable? It is ability of recruitment to be changed without causing problems to a system because they allow the system to be easily modified to meet the changing needs

6) Briefly explain requirements management

Requirements management is a critical aspect of software development, ensuring that the software meets the needs of its users and stakeholders. It involves the process of gathering, analyzing, documenting, prioritizing, and tracking requirements throughout the software development lifecycle

1. Large system have diverse user community where users have different requirement and priority this may conflict so requirement are compromised. srs should be updated to replace change in user need over time this will help to ensure that system meets the need of users to best extend possible
2. Customer and user often have different needs and requirement may change after delivery SRS should be updated to reflect this change this will help ensure the system meet the need of both customer and user that is able to adapt to changing business needs.
3. The business and technical environment of the system changes after installation, and these changes must be reflected in the system. new hardware may be introduced other system may need be integrated, business priorities may shift and new laws and regulations may be enacted.

7) Explain the concept of requirements management planning.

Planning is an essential first stage in the requirements management process. Requirements management is very expensive. For each project, the planning stage establishes the level of requirements management detail that is required.

Requirements identification is an essential step in the software development lifecycle (SDLC) as it helps to establish a clear understanding of the project's goals, scope, and functionality. By defining clear and measurable requirements, the development team can ensure that the final product meets the needs of the stakeholders. Additionally, requirements identification helps to identify and eliminate unnecessary features, which can save time and money during the development process.

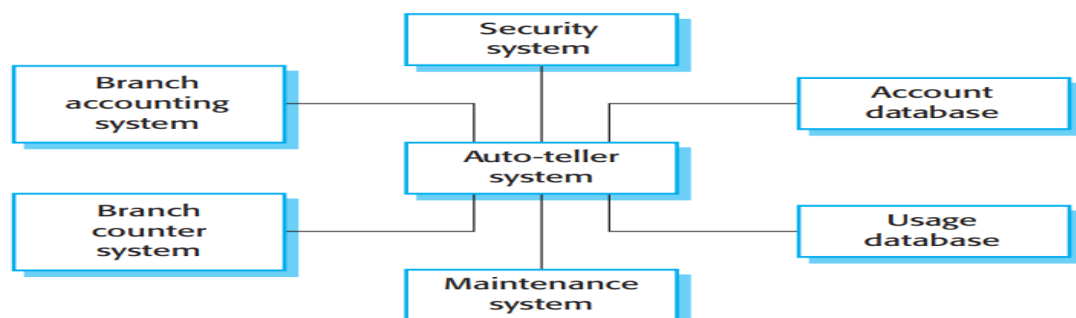
A change management process is a structured approach to handling changes in a system or project. It involves planning, implementing, and evaluating changes to ensure that they are implemented effectively and that their impact is minimized.

Traceability policies These policies define the relationships between requirements, and the system design that should be recorded and how these records should be maintained.

CASE tool support Requirements management involves the processing of large amounts of information about the requirements. Tools like MS Excel, spreadsheets, or a simple database system can be used.

8) Explain the context of an ATM system with the diagram

A context model defines how context data are structured and maintained. It aims to produce a formal description of the context information that is present in a context-aware system.



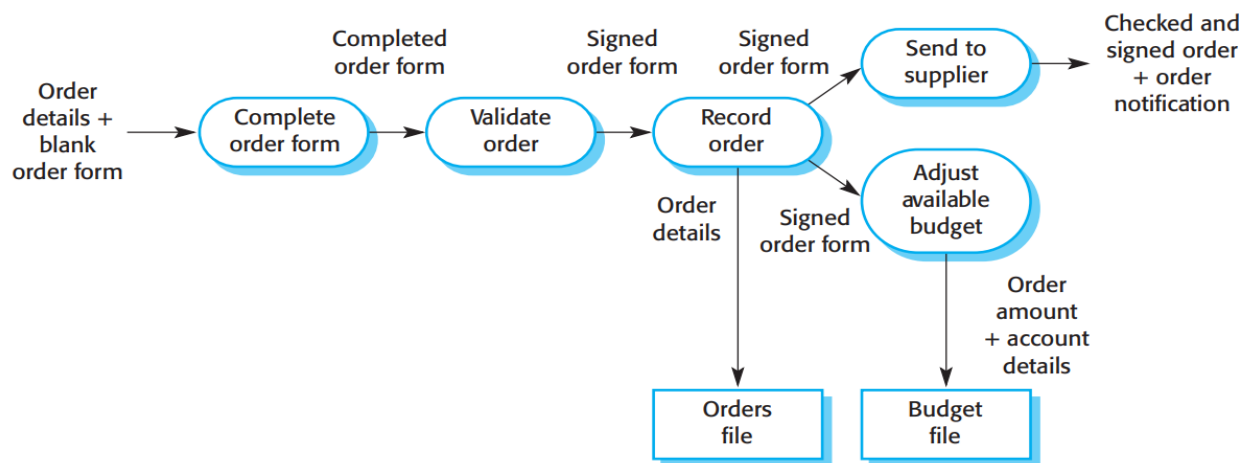
1. **Security System:** This component safeguards the bank's assets and data from unauthorized access, ensuring the integrity of customer information and financial transactions.
2. **Branch Accounting System:** This system handles transactions initiated at the bank's branches, including deposits, withdrawals, account maintenance, and balance inquiries. It maintains records of customer accounts and transaction details.
3. **Auto-Teller System (ATM System):** This system facilitates self-service transactions through ATMs, allowing customers to access their

accounts, withdraw cash, and make deposits without interacting with bank staff. It communicates with the branch accounting system to process transactions and update account balances.

4. **Branch Usage Counter System:** This system tracks customer usage of the branch, providing insights into transaction volumes, peak usage times, and customer preferences. This data can be used to optimize staffing and resource allocation.
5. **Maintenance System:** This system manages the maintenance of hardware and software components in the banking system, ensuring the system's uptime and performance. It schedules regular maintenance tasks, logs system events, and tracks the resolution of any issues.

9) Explain the data- flow diagram of order processing

It is used to show data flows through a sequences of processing steps.



1. Gather order details. This may include the product(s) being ordered, the quantity of each product, and the delivery address.
2. Complete order form. The order form should include all of the required information, such as the order details, the supplier's contact information, and the buyer's signature.
3. Validate order. The buyer should review the order form to ensure that all of the information is accurate.
4. Record order. The buyer should record the order in their system so that they can track its status.
5. Adjust available budget. The buyer should adjust their available budget to reflect the cost of the order.
6. Send signed order form to supplier. The buyer should send the signed order form to the supplier so that they can begin processing the order.
7. Receive signed order form from supplier. The buyer should receive a signed copy of the order form from the supplier to confirm that the order has been received and accepted.

10) Write a note on state-machine model?

A state machine model describes how a system responds to internal or external events. The state machine model shows system states and events

that cause transitions from one state to another. It does not show the flow of data within the system. This type of model is often used for modelling real-time systems because these systems are often driven by stimuli from the system's environment.

Advantage:

- **Clarity:** They provide a clear and concise way to represent the system's behavior.
- **Formalism:** They can be represented formally using mathematical notation, which makes them amenable to analysis and verification.
- **Modularity:** They can be decomposed into smaller, more manageable components, which makes them easier to understand and maintain.
- **Reusability:** They can be reused for modeling different systems with similar behavior

11) Write a note on data model. Or state data model?

The most widely used data modelling technique is Entity-Relation-Attribute modelling (ERA modelling), which shows the data entities, their associated attributes and the relations between these entities.

ERA modeling is a valuable tool for several reasons:

- It helps to identify and define the data requirements of an information system.
- It provides a clear and concise representation of the data in a system.
- It can be used to identify and resolve potential data inconsistencies and redundancies.
- It can serve as a blueprint for the implementation of a database.

12) Write a note on object model

An object model defines the classes, attributes, methods, and their relationships in a system. It is a blueprint for creating and using objects in a software application.

The Unified Modeling Language (UML) used in this unified method has become a standard for object modelling.

The three main components of an object model are:

- **Classes:** Classes are blueprints that define the attributes and methods of objects.
- **Objects:** Objects are instances of classes. They have state, which is represented by their attributes, and behavior, which is represented by their methods.
- **Relationships:** Relationships define how objects are connected to each other. There are three main types of relationships: aggregation, composition, and inheritance.

An object class in UML, represented as a vertically oriented rectangle with three sections:

1. The name of the object class is in the top section.
2. The class attributes are in the middle section.
3. The operations associated with the object class are in the lower section of the rectangle.

Benefit:

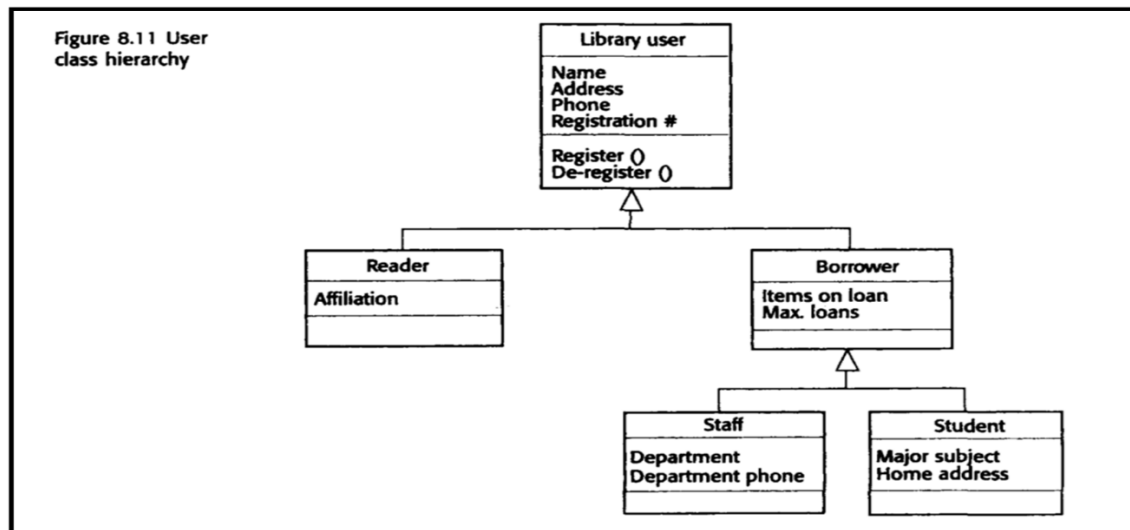
- **Improved understanding of complex systems:** Object models provide a way to break down complex systems into smaller, more manageable pieces. This can make it easier to understand how the system works and how to change it.
- **Increased code reusability:** Object models can be used to create reusable classes and objects. This can save time and effort when developing new software systems.
- **Improved maintainability of code:** Object models can make code easier to maintain by making it easier to understand how the code works and how to change it.

13) Write a note on inheritance model.

Inheritance is a mechanism in object-oriented programming (OOP) that allows new objects to take on the properties of existing objects. This means that new objects can inherit the attributes and methods of existing objects, and can also add their own additional attributes and methods.

- **Reusability:** Inheritance allows programmers to reuse code that is already available, which can save time and effort.
- **Maintainability:** Inheritance can make code more maintainable by making it easier to understand and change.

- **Extensibility:** Inheritance allows programmers to extend existing classes by adding new attributes and methods.
- **Polymorphism:** Inheritance is one of the key concepts that makes polymorphism possible. Polymorphism is the ability of objects to take on many forms.



14) Briefly explain object aggregation with the diagram.

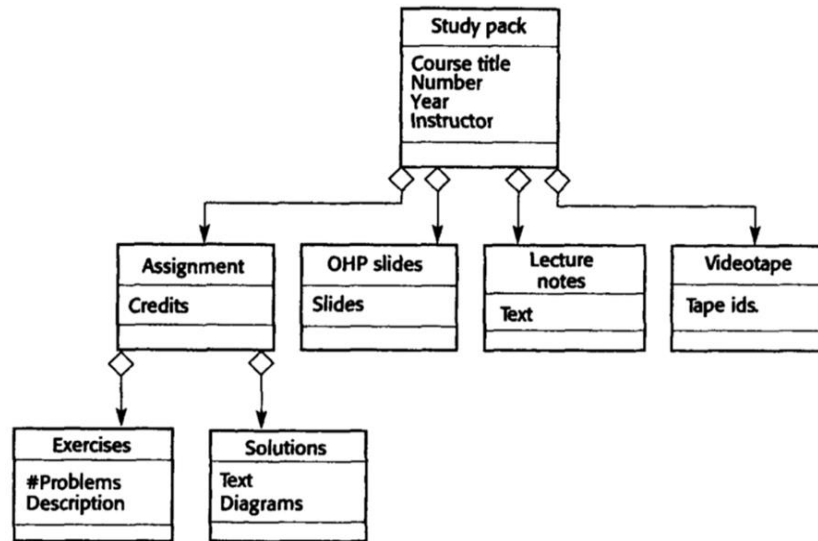
It deals with the relationships between a whole object and its parts or components.

As well as acquiring attributes and services through an inheritance relationship with other objects, some objects are groupings of other objects.

That is, an object is an aggregate of a set of other objects. The classes representing these objects may be modelled using an object aggregation model.

The UML notation for aggregation is to represent the composition by including a diamond shape on the source of the link.

Figure 8.13
Aggregate object
representing a
course



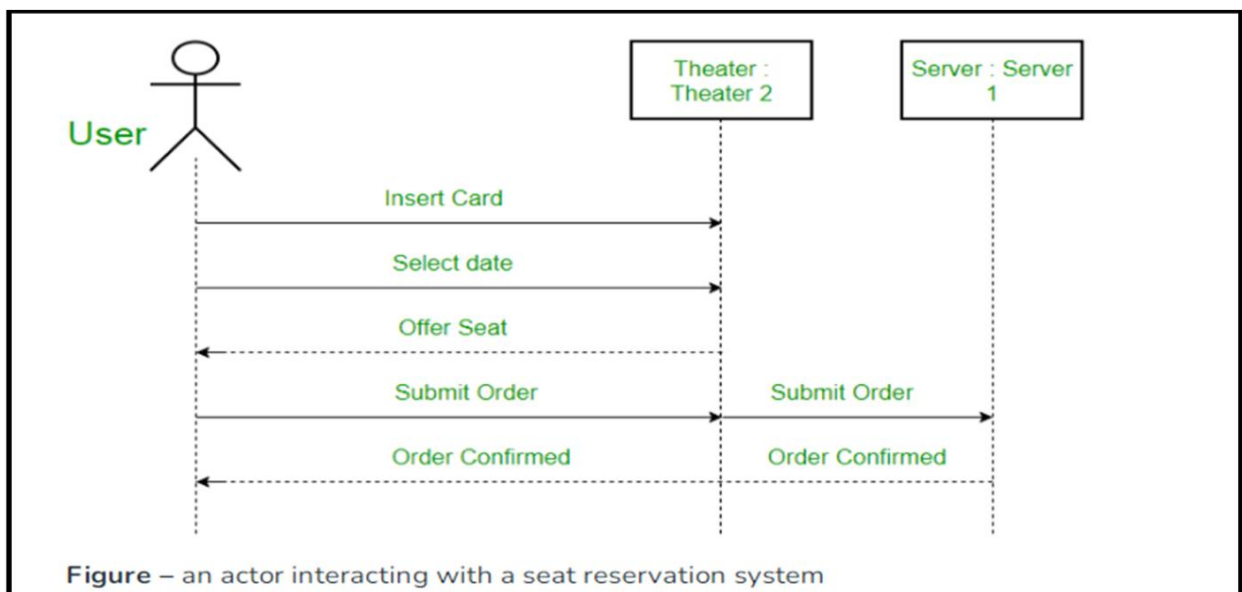
1. **Course:** Represents the overarching structure of a course, encompassing various aspects and materials.
2. **Assignments:** Refers to the specific tasks or exercises assigned to the students within the course.
3. **OHP Slides:** Denotes the overhead projector slides used for presentation or lecture purposes.
4. **Lecture Notes:** Represents the written notes compiled during lectures or presentations.
5. **Videotapes:** Indicates any recorded materials or lectures provided for reference.
6. **Credits:** Denotes the credit points allocated for completing the course successfully.
7. **Exercises:** Refers to practice problems or exercises provided for students to reinforce their understanding.
8. **Solutions:** Represents the provided answers or solutions to the exercises for students' reference.
9. **Problems:** Indicates the specific problems or questions posed within the course material.
10. **Descriptions:** Refers to the detailed explanations or descriptions provided for the course's content.

11. **Diagrams:** Denotes any visual representations or diagrams used to explain concepts or structures within the course.

15) Explain object behaviour modelling with the help of a diagram.

Object behavior modeling is a technique for representing the behavior of objects in a software system. It involves using diagrams to visualize how objects interact with each other and how their state changes over time.

- **Improved understanding of complex systems:** Object behavior modeling can make it easier to understand how complex systems work by breaking them down into smaller, more manageable pieces.
- **Increased code reusability:** Object behavior models can be used to create reusable components that can be used in multiple software systems.
- **Improved maintainability of code:** Object behavior models can make code easier to maintain by making it easier to understand how the code works and how to change it.



Here is a more detailed explanation of each step:

1. **Insert card:** The user inserts their card into the system. The system then verifies the user's identity and checks to see if the user has a balance on their account.
2. **Select date and time:** The user selects the date and time of the show that they want to attend.
3. **Select seats:** The system displays a map of the theater with the available seats highlighted. The user selects the seats that they want to reserve.
4. **Confirm reservation:** The system confirms the user's reservation and charges the user's account.
5. **Receive confirmation number:** The user is given a confirmation number for their reservation.

16) What is structured method? Explain the weaknesses of structured methods.

A structured method is a systematic way of producing models of an of a system that is to be built. Structured methods provide a framework for detailed system modelling as part of requirements elicitation and analysis.

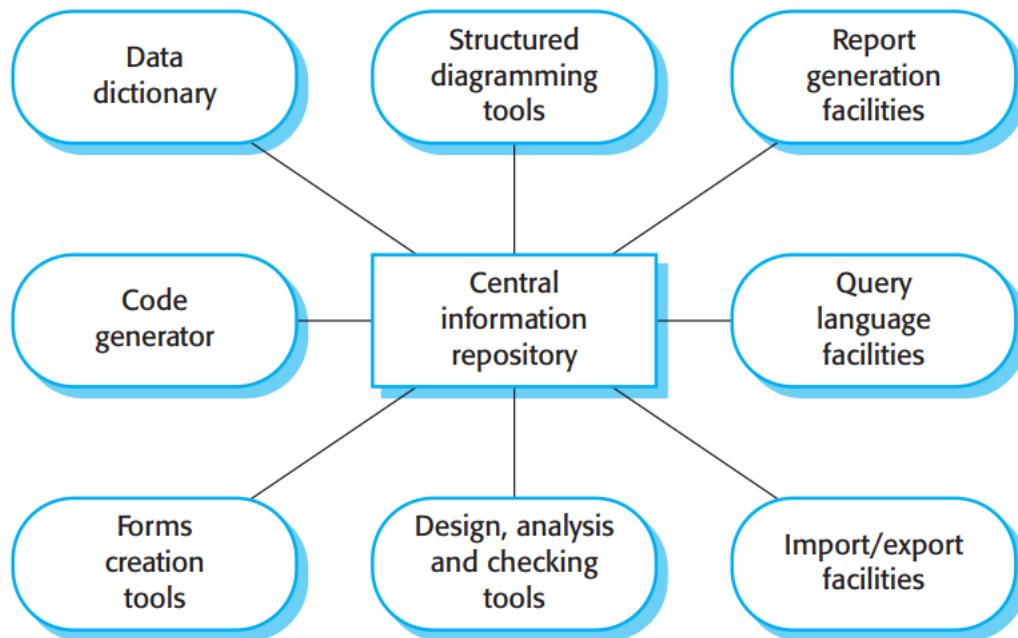
1. They don't effectively help you understand or create non-functional system requirements.
2. They don't give clear guidance on whether a method is suitable for a specific problem. Also, they lack advice on adapting them for a particular environment.
3. They often create too many documents, making it hard to find the essential system requirements among all the details.
4. The models that are produced are very detailed, and users often find them difficult to understand. These users therefore cannot check the realism of these models.

17) Briefly explain the components of a CASE tool for structured method support with a diagram

Analysis and design CASE tools support the creation, editing and analysis of the graphical notations used in structured methods.

1. **Diagram editors** used to create object models, data models, behavioural models, and so on. These editors are not just drawing tools but are aware of the types of entities in the diagram. They capture information about these entities and save this information in the central repository.
2. **Design analysis and checking tools :** Tools for design analysis and checking find errors and issues in the design. They can be part of the editing system, catching user mistakes early in the process.

3. **Repository query languages:** help designers locate designs and related information in the repository.
4. **A data dictionary** :that maintains information about the entities used in a system design.



5. **Report definition and generation tools** that take information from the central store and automatically generate system documentation.
6. **Forms definition tools:** Tools for defining forms let you specify how screens and documents should look.
7. **Import/export facilities** that allow the interchange of information from the central repository with other development tools.
8. **Code generators** that generate code or code skeletons automatically from the design captured in the central store

Unit 3

2 mark:

1) What is architectural design?

Architectural design where we determine how a system should be organized and design its overall structure. This is a crucial step, being the bridge between requirements engineering and detailed design. It identifies the main structural components of a system and outlines their relationships.

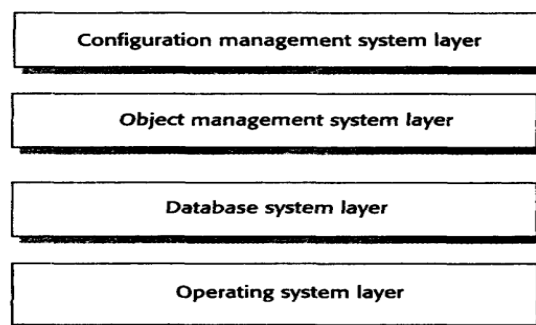
2) What are the major components of client-server model?

- A set of servers that offer services to other sub-systems.
- A set of clients that call on the services offered by servers.
- A network that allows the clients to access these services.

3) Expand APSE

Ada Programming Support Environment

4) Give the layered model of a version management system



The layered model of an architecture (sometimes called an abstract machine model) organises a system into layers, each of which provide a set of services.

5) Write the distinction between sub-systems and modules.

Feature	Subsystem	Module
Size	Large	Small
Independence	Independent	Dependent

Complexity	Complex	Simple
Boundaries	Well-defined	Less well-defined

6) Name the two main strategies while decomposing a sub-system into modules

- **Object-oriented decomposition** where you decompose a system into a set of communicating objects.
- **Function-oriented pipelining** where you decompose a system into functional modules that accept input data and transform it into output data.

7) Write any 2 advantages of function-oriented pipelining architecture

1. It supports the reuse of transformations.
2. It is naturally that many people think of their work in terms of input and output processing.
3. Evolving the system by adding new transformations is usually straightforward.
4. It is simple to implement either as a concurrent or a sequential system.

8) Write the stages of object-oriented design process

- Definition of the context of the system
- Designing system architecture
- Identification of the objects in the system
- Construction of design models
- Specification of object interfaces

9) What is the use of static model and dynamic model w.r.t. system context?

Static models are used to represent systems that do not change over time. They are typically represented by equations or diagrams that capture the relationships between the different components of the system.

Dynamic models are used to represent systems that change over time. They are typically represented by differential equations or difference equations that capture the rate of change of the system's state variables

10) What is object identification?

Object identification in software engineering refers to the process of uniquely identifying objects within a software system. Objects are instances of classes in object-oriented programming, and they encapsulate data and behavior.

11) What is grammatical analysis of a natural language

Use a grammatical analysis of a natural language description of the system to be constructed. Objects and attributes are nouns; operations or services are verbs.

- Objects: customer, book, employee, user
- Attributes: title, author, publication date, due date
- Operations: add, remove, update, search, borrow, return

12) What is behavioural approach?

Use a behavioural approach where the designer first understands the overall behaviour of the system. The various behaviours are assigned to different parts of the system and that focuses on understanding the behavior of the system before designing its components. This approach is based on the idea that the behavior of a system is more important than its structure.

13) What is scenario-based analysis?

scenario-based analysis is a software design technique that involves identifying and analyzing various scenarios of system use. This analysis helps designers to understand the system's requirements and to identify the objects, attributes, and operations that the system needs.

14) What is the use of design models w.r.t.object oriented design?

Design models are abstract representations of a system that help designers to understand, communicate, and analyze the system's structure and behavior. In object-oriented design (OOD), design models play a crucial role in creating well-structured, maintainable, and scalable software systems.

15) Name the types of design models that describe an object oriented design

- Static models describe the static structure of the system using object classes and their relationships.
- Dynamic models describe the dynamic structure of the system and show the interactions between the system objects.

16) What is the function of static model and dynamic model w.r.t.object oriented design?

Static models describe the static structure of the system using object classes and their relationships.

Dynamic models describe the dynamic structure of the system and show the interactions between the system objects.

17) What is the use of subsystem model and sequence models?

- ❖ Subsystem models that show logical groupings of objects into ordered sub-systems. These are represented using a form of class diagram where each sub-system is shown as a package. Subsystem models are static models.
- ❖ Sequence models that show the sequence of object interactions. These are represented using a UML sequence or a collaboration diagram. Sequence models are dynamic: models

18) Name the four elements of design patterns.

- **A name** that is a meaningful reference to the pattern
- **A description** of the problem area that explains when the pattern may be applied
- **A solution description** of the parts of the design solution, their relationships and their responsibilities.
- **A statement of the consequences**-the results and trade-offs-of applying the pattern. This can help designers understand whether a pattern can be effectively applied in a particular situation.

4 mark:

1) What are the various points that are considered while developing architectural model?

1. System Requirements:

- Understand and analyze the functional and non-functional requirements of the system. Identify the key features and constraints that the architecture must address.

2. Stakeholder Needs:

- Consider the needs and expectations of different stakeholders, including end-users, customers, developers, and system operators. The architecture should align with their goals and priorities.

3. Scalability:

- Evaluate the scalability requirements of the system. Design the architecture to handle potential growth in data volume, user base, and other aspects without significant performance degradation.

4. Performance:

- Address performance considerations by analyzing response times, throughput, and resource utilization. Design the architecture to meet or exceed performance expectations.

5. Security:

- Incorporate security measures to protect the system against unauthorized access, data breaches, and other security threats. Consider encryption, authentication, authorization, and other security mechanisms.

6. Flexibility and Extensibility:

Create an architecture that allows for flexibility and extensibility. Design components and modules that can be easily modified or extended to accommodate changing requirements.

7. Maintainability:

- Design the system with maintainability in mind. Use modular and well-structured components, adhere to coding standards, and consider the ease of future enhancements and updates.

2) Write the advantages and disadvantages of a shared repository model.

A shared repository model, also known as a monorepo, is a software development approach that involves storing all of the source code for a project in a single repository. This can be contrasted with a multi-repo approach, where the source code is divided into multiple repositories.

Advantages of a Shared Repository Model

- **Improved code reuse:** Developers can easily access and reuse code from other parts of the project.
- **Enhanced code consistency:** All of the code is in a single location, which can help to ensure that it is consistent and well-documented.
- **Simplified dependency management:** There are fewer dependencies to manage, which can make it easier to build and deploy the project.
- **Stronger collaboration:** Developers can more easily collaborate on code, as they are all working from a single source of truth.

Disadvantages of a Shared Repository Model

- **Increased complexity:** Large repositories can become complex and difficult to manage.
- **Performance issues:** Large repositories can impact build times and performance.

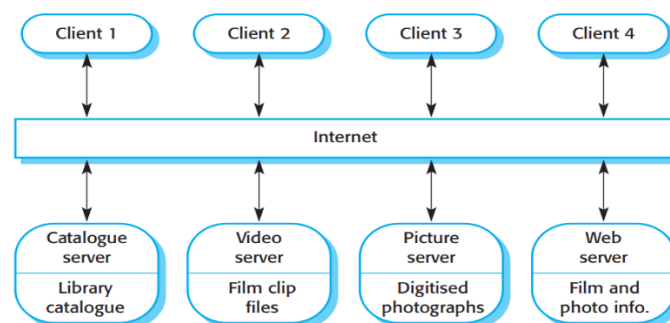
- **Difficulties with branching and merging:** Branching and merging can be more complex with a single repository.
- **Limited scalability:** Monorepos can become difficult to scale as the project grows.

3) Explain client-server model with an example diagram.

The client-server model is a distributed application architecture that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients. In this model, the client initiates a request for a specific service or resource from the server, and the server processes the request and delivers the requested data or service back to the client.

Clients: The clients represent the user's computers or devices that access the website. In this case, there are four clients, labeled Client 1, Client 2, Client 3, and Client 4.

Internet: The internet serves as the communication channel between the clients and the servers. It provides the network infrastructure for the clients to send requests to the servers and receive responses.



Catalogue, Video, Picture, Web Servers: These are the servers that provide specific services to the clients.

Catalogue Server: Stores information about library books, including titles, authors, genres, and availability.

Video Server: This server stores and delivers video content, such as movie trailers or instructional videos.

Picture Server: This server stores and delivers picture content, such as still images or digital artwork.

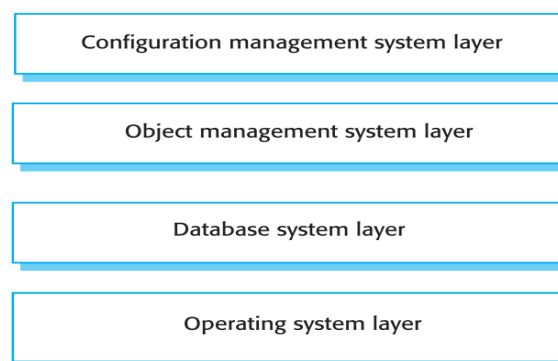
Web Server: This server handles web requests and delivers web pages to the clients. It is responsible for presenting the website's content to the users.

4) Explain layered model with a diagram.

The Layered Architecture is a model that organizes a system into layers, each of which provides a set of services. The model is also known as an abstract machine model. Each layer can be thought of as an abstract machine whose machine language is defined by the services provided by the layer.

The configuration management system (CMS) layer is like the control center in a software setup. It's crucial for keeping everything organized and reliable. Imagine it as a central hub that stores and keeps track of all the important settings and changes in the software. This ensures that everything in the system stays consistent and works smoothly together.

The object management system (OMS) layer is like the manager of a software team. Its main job is to handle the creation, changes, and removal of different pieces of information, called objects. Think of it as the organized system that makes sure everything works smoothly and efficiently in the software. This layer helps the software perform well and handle a lot of information without slowing down.



The database system layer is a crucial component of a layered software architecture, responsible for managing and storing data persistently. It provides a structured and efficient way to organize, access, and manipulate data, ensuring data integrity, security, and availability.

The operating system (OS) layer is like the boss of a computer, handling all the important tasks and making everything run smoothly. It's the foundation that supports the computer's hardware and allows user applications to work. Think of it as the manager that organizes the computer's memory, decides which tasks to do first, and controls devices like printers and keyboards.

5) Write a note on object-oriented decomposition

Object-oriented decomposition (OOD) is a software design technique that breaks down a complex system into smaller, more manageable objects.

Object-oriented decomposition is a software design technique that involves identifying and defining objects and their relationships within a software

system. It is a fundamental aspect of object-oriented programming (OOP) and is used to create well-structured, maintainable, and reusable software.

1. Define the methods of each object. Methods represent the actions that an object can perform.
2. Define the relationships between objects. Relationships represent how objects are connected to each other.
3. Assign responsibilities to objects. Responsibilities are the tasks that an object is responsible for performing.

Advantage:

1. **Modularity:** Promotes modularity by dividing the system into independent objects, making the code easier to understand, maintain, and extend.
2. **Reusability:** Encourages code reuse by creating reusable objects that can be used in different parts of the system or in other applications.
3. **Maintainability:** Simplifies maintenance by making it easier to identify and modify specific objects without affecting the entire system.
4. **Scalability:** Facilitates scalability by enabling the addition of new objects and relationships without significantly impacting the overall structure.

6) Explain function-oriented pipelining with an example.

Function-oriented pipelining where you decompose a system into functional modules that accept input data and transform it into output data.

Function-oriented pipelining is a software development technique that involves chaining together a series of functions to perform a complex task. Each function in the pipeline takes the output of the previous function as its input and produces an output that is passed to the next function in the pipeline.

- **Modularity:** Function-oriented pipelining promotes modularity by breaking down complex tasks into smaller, independent functions. This makes the code easier to understand, maintain, and extend.
- **Reusability:** Functions in a pipeline can be reused in other contexts, reducing development time and effort.
- **Testability:** Each function in a pipeline can be tested independently, making it easier to identify and fix bugs.
- **Readability:** Function-oriented pipelining makes the code more readable by expressing the data flow explicitly.

Function-oriented pipelining in the kitchen would involve breaking down the meal preparation process into smaller, more manageable steps:

1. Gather ingredients: This involves washing and chopping vegetables, measuring spices, and preparing any other ingredients needed for the meal.
2. Cook the main dish: This could involve grilling meat, roasting vegetables, or simmering a sauce.
3. Prepare side dishes: This could involve boiling rice, steaming vegetables, or making a salad.
4. Assemble and plate the meal: This involves bringing all the components together, plating them attractively, and adding any finishing touches.

7) Write a note on object oriented design process with an example

Object-oriented design (OOD) is a software development process that focuses on creating a model of a software system using objects and their interactions. It aims to create a design that is modular, reusable.

Example: ATM System

Consider designing an ATM system to allow bank customers to withdraw cash, check their account balance, and transfer funds.

1. **Requirements analysis:**

- The ATM should allow customers to withdraw cash from their accounts.
- The ATM should display the account balance of the current customer.
- The ATM should allow customers to transfer funds between their accounts.

2. **Domain analysis:**

- Key concepts: Customer, Account, Transaction
- Entities: ATM, Bank Card

3. **Object identification:**

- Customer object: Represents a bank customer with attributes like name, account number, and PIN.
- Account object: Represents a bank account with attributes like account number, balance, and customer ID.

4. Class definition:

- Customer class: Attributes: name, account number, PIN
- Methods: validatePIN(), withdrawCash(), checkBalance(), transferFunds()
- Account class: Attributes: account number, balance, customer ID
- Methods: deposit(), withdraw(), getBalance()

5. Relationship identification:

- Customer has an Account
- Customer has a Bank Card
- Transaction belongs to an Account

6. Design patterns:

- Use the Factory pattern to create and manage objects
- Use the Strategy pattern to handle different transaction types
- Use the Observer pattern to notify listeners of account balance changes

7. Design documentation:

- Create class diagrams to represent classes and their relationships
- Use UML (Unified Modeling Language) to document the design effectively

8) Explain system context and models of use with a use-case diagram

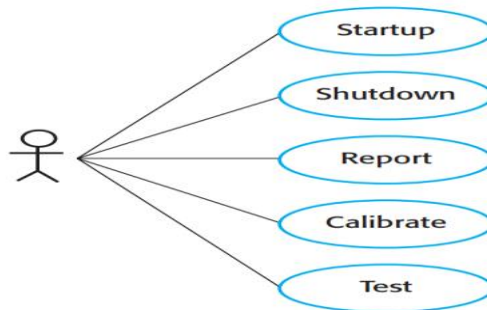
The system context is a static model that describes the other systems in that environment.. system context refers to the environment in which a system operates. It includes all of the external factors that can affect the system, such as users, other systems, and physical environment. System context is important to understand because it can help to identify potential problems or opportunities for the system.

model of the system use is a dynamic model that describes how the system actually interacts with its environment.

Models of use are a way of describing how a system is used. They are typically represented using use-case diagrams. A use-case diagram is a type of diagram that shows the interactions between the system and its actors. Actors are people, other systems, or devices that interact with the system.

- They are easy to understand and use.

- They can help to identify potential problems or opportunities for the system.
- They can be used to communicate requirements to stakeholders the use cases for the weather station in the image are



- **Startup:** This use case describes the steps that are involved in starting up the weather station. This may include turning on the power, connecting the sensors, and calibrating the instruments.
- **Shutdown:** This use case describes the steps that are involved in shutting down the weather station. This may include disconnecting the sensors and turning off the power.
- **Report:** This use case describes the process of generating a report of the weather data that has been collected by the weather station. This report may be sent to a central database, displayed on a local screen, or printed out.
- **Calibrate:** This use case describes the process of calibrating the sensors in the weather station. This ensures that the sensors are providing accurate readings.
- **Test:** This use case describes the process of testing the weather station to ensure that it is working properly. This may include checking the readings from the sensors and running diagnostic tests on the electronics.

9) Briefly explain architectural design with an example.

Architectural design is the process of defining the overall structure and components of a system. It involves making decisions about the high-level design of the system, such as the components that will be included, the interfaces between those components, and the overall organization of the system.

To illustrate this concept, consider an e-commerce platform as an example. The architectural design of such a platform would encompass the following aspects:

Components:

1. **Product Catalog:** Manages product information, including descriptions, images, and prices.
2. **Shopping Cart:** Stores items selected by users for purchase.
3. **User Management:** Handles user registration, authentication, and profile management.
4. **Payment Processing:** Integrates with payment gateways to process transactions securely.
5. **Order Management:** Tracks orders, manages inventory, and handles order fulfillment.

Interactions:

1. Users browse the product catalog and add items to their shopping cart.
2. Users proceed to checkout, providing shipping and payment information.
3. The system validates payment information and processes the transaction.
4. The system confirms the order, updates inventory, and initiates order fulfillment.
5. Users receive order status updates and track their shipments.

The architectural design of an e-commerce platform defines the overall structure and interactions between these components to ensure seamless user experience, secure transactions, and efficient order processing.

10) Explain the various proposals mode to identify object classes

- **Use a grammatical analysis** of a natural language description of the system to be constructed. Objects and attributes are nouns; operations or services are verbs.
 - Objects: customer, book, employee, user
 - Attributes: title, author, publication date, due date
 - Operations: add, remove, update, search, borrow, return
- **Use tangible entities** (things) in the application domain such as aircraft, roles such as manager, events such as request, interactions such as meetings, locations such as offices, organisational units such as companies, and so on. Support this by identifying storage structures (abstract data structures) in the solution domain that might be required to support these objects

- **Use a behavioural approach** where the designer first understands the overall behaviour of the system. The various behaviours are assigned to different parts of the system and that focuses on understanding the behavior of the system before designing its components. This approach is based on the idea that the behavior of a system is more important than its structure.
- **scenario-based analysis** is a software design technique that involves identifying and analyzing various scenarios of system use. This analysis helps designers to understand the system's requirements and to identify the objects, attributes, and operations that the system needs.

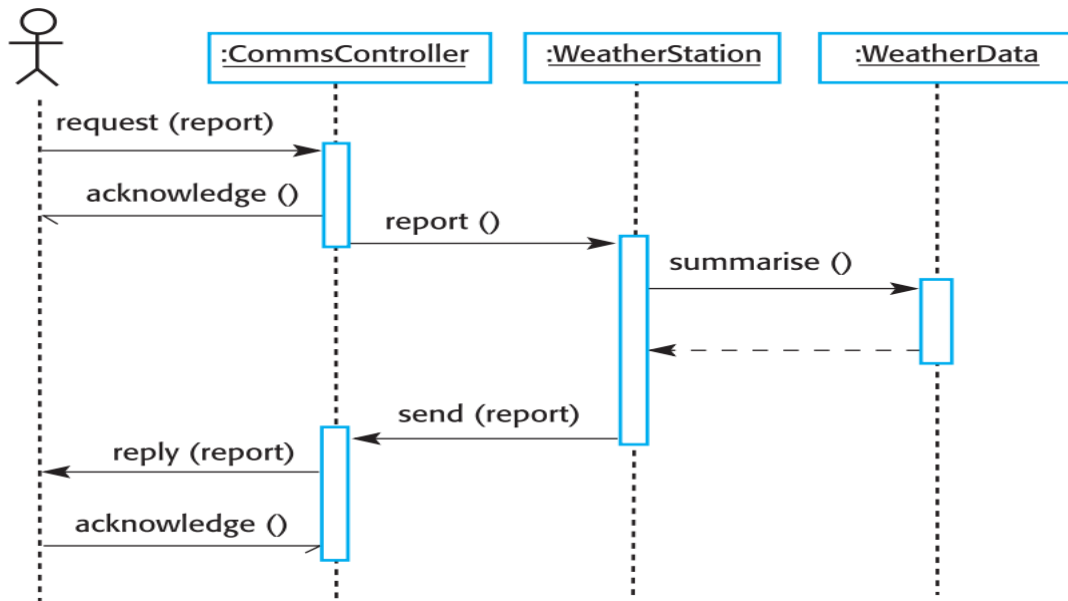
11) Explain sequence model that shows the operations involved in collecting the data from a weather section with a diagram

Sequence models that show the sequence of object interactions. These are represented using a UML sequence or a collaboration diagram. Sequence models are dynamic models.

the weather station gathers weather data from various sensors, such as temperature, humidity, and wind speed. It then sends this data to the weather data server, which processes it and stores it in a database. The weather data server can also be used to generate reports and visualizations of the weather data.

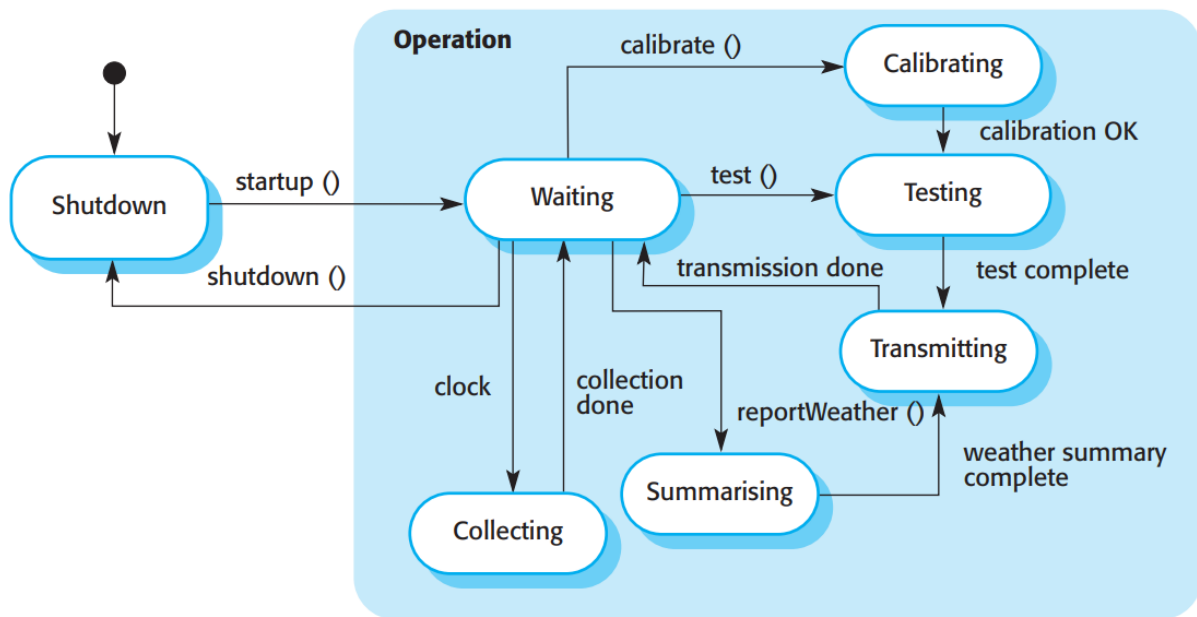
Here's a more detailed explanation of the image:

- **CommsController:** This is the main component that manages the communication between the weather station and the weather data server.
- **WeatherStation:** This component represents the actual weather station hardware. It collects data from the sensors and sends it to the CommsController.



- **WeatherData:** This component represents the weather data that is collected by the weather station. It is stored in a data structure that is optimized for efficient access.
- **request(report):** This message is sent from the CommsController to the WeatherStation to request a weather report.
- **acknowledge():** This message is sent from the WeatherStation to the CommsController to acknowledge the request.
- **report():** This message is sent from the WeatherStation to the CommsController to provide the weather report.
- **summarise(report):** This message is sent from the CommsController to the WeatherData to summarize the weather report.
- **send(report):** This message is sent from the WeatherData to the CommsController to send the summarized weather report to the weather data server.
- **reply(report):** This message is sent from the CommsController to the WeatherData to acknowledge the receipt of the summarized weather report.
- **acknowledge():** This message is sent from the WeatherData to the CommsController to acknowledge the acknowledgement.

12) With a statechart diagram, explain how WeatherStation object responds to requests for various services?



The state diagram you provided shows the different states that a weather station can be in, as well as the events that can trigger transitions between states.

The state diagram consists of the following states:

- **Start** : This is the initial state of the weather station.
- **Calibrating** : This state is used to calibrate the sensors in the weather station.
- **Collecting** : This state is used to collect weather data from the sensors.
- **Reporting** : This state is used to generate and send weather reports.
- **Shutdown** : This state is used to shut down the weather station.

The following events can trigger transitions between states:

- **startup** : This event triggers the transition from the Start state to the Calibrating state.
- **calibrationOK** : This event triggers the transition from the Calibrating state to the Collecting state.
- **collectionDone** : This event triggers the transition from the Collecting state to the Reporting state.
- **reportComplete** : This event triggers the transition from the Reporting state to the Collecting state.
- **shutdown** : This event triggers the transition from any state to the Shutdown state.

Here is a brief explanation of the state diagram:

1. *The weather station starts in the Start state.*
2. *The weather station then transitions to the Calibrating state to calibrate its sensors.*
3. *Once the sensors are calibrated, the weather station transitions to the Collecting state.*
4. *The weather station collects weather data from its sensors until the collection is done.*
5. *Once the collection is done, the weather station transitions to the Reporting state to generate and send a weather report.*
6. *Once the weather report is complete, the weather station transitions back to the Collecting state to collect more weather data.*
7. *The weather station can also transition to the Shutdown state .*

13) Write a note on object interface specification

An object interface specification defines the behavior of an object and the interactions that other objects can have with it. The purpose of an object interface specification is to ensure that different objects can communicate and interact with each other in a consistent and predictable way.

Key Components of an Object Interface Specification

- **Methods:** Define the actions that an object can perform.
- **Properties:** Define the attributes or characteristics of an object.
- **Events:** Define the notifications that an object can send to other objects.
- **Preconditions and Postconditions:** Define the conditions that must be met before and after calling a method.

Benefits of Object Interface Specifications

- **Encapsulation:** Enforces data hiding and prevents unauthorized access to an object's internal state.
- **Reusability:** Allows objects to be reused in different parts of an application without modifying their internal implementation.
- **Maintainability:** Makes it easier to understand and modify the behavior of objects.
- **Testability:** Facilitates the testing of objects and their interaction

14) Write a note on design patterns.

design patterns are reusable solutions to commonly recurring problems in software design. They serve as blueprints for addressing frequently encountered challenges and provide a standardized approach to creating robust, flexible, and maintainable software systems.

1. **Reusability:** Design patterns promote code reuse, reducing development time and effort.
2. **Maintainability:** Well-structured patterns enhance code readability and maintainability, simplifying modifications and updates.
3. **Flexibility:** Design patterns provide adaptability to changing requirements, making software more resilient and adaptable.
4. **Communication:** Design patterns serve as a common language among developers, fostering better understanding and collaboration.

Unit 4

2 mark:

1) What is test planning?

Test planning is the process of creating a blueprint for how software will be tested. It includes defining the scope of testing, identifying test objectives, and outlining the resources and schedule for testing activities. A well-defined test plan is essential for ensuring that software is thoroughly tested and released with minimal defects.

2) What is system testing?

System testing is a crucial stage in the software development process, especially in an iterative development environment. It involves integrating two or more components that implement specific system functions or features and then testing this integrated system as a whole. The focus is on testing the integrated system increment that is ready to be delivered to the customer.

3) What are two phases in system testing?

Integration testing: where the test team have access to the source code of the system. When a problem is discovered, the integration team tries to find the source of the problem and identify the components that must be debugged. Integration testing is mostly concerned with finding defects in the system.

Release testing: Release testing is usually 'black-box' testing where the test team is simply concerned with demonstrating that the system does or does not work properly. Problems are reported to the development team whose job is to debug the program. Where customers already involved in release testing, this is sometimes called acceptance testing. If the release is good enough, the customer may then accept it for use.

4) What is Integration testing?

Refer above

5) What is top-down integration and down-up integration?

The overall skeleton of the system is developed first, and components are added to it. This is called top-down integration. In Top Down Integration testing approach the main module is designed at first then the submodules are called from it.

In Bottom Up Integration testing approach different modules are created first then these modules are integrated with the main function. System integration begins with lowest(top) level modules.

6) What is release testing?

Refer 2m 3rd

7) What is black Box Testing and white box testing?

1. **Black Box Testing** is a software testing method in which the internal structure/design/implementation of the item being tested is not known to the tester. Only the external design and structure are tested. It is also called closed testing.
 - Identify interface-level bugs
 - Determine if software is easy to use
2. **White Box Testing** is a software testing method in which the internal structure/design/implementation of the item being tested is known to the tester. Implementation and impact of the code are tested. It is also called as clear box testing.
 - Detect internal defects
 - Ensure that code is efficient and maintainable

8) What is glass Box or clear Box testing?

Also known as whitebox testing

9) What is Component Testing?

Component testing (sometimes called unit testing) is the process of testing individual components in the system. This is a defect testing process so its goal is to expose faults in these components.

10) What is software testing workbench?

A software testing workbench is a comprehensive set of tools designed to facilitate the testing process. **It goes beyond just providing frameworks for automated test execution and offers a range of features to facilitate various aspects of testing.**

11) Describe Junit.

JUnit is a set of Java classes that the user extends to create an automated testing environment. Each individual test is implemented as an object and a test runner runs all of the tests. Test automation frameworks such as JUnit are examples of test managers.

12) What is Interface testing?

Interface Testing is a type of software testing type that checks the proper communication between two different software systems. Interface is the connection that integrates two components. The interface could be anything like APIs, web services etc.

13) What is Unit testing and regression testing?

Regression Testing means to confirm that a recent program or code change has not adversely affected existing features. and that the new code interacts as expected with the existing code.

unit tests verify that individual units of code work properly in isolation. Unit testing involves the testing of each unit or an individual component of the software application.

14) What is alpha testing and beta testing?

Alpha Testing is a type of software testing performed to identify bugs before releasing the product to real users or to the public. performed by testers who are usually internal employees (black and white)

Beta Testing is performed by real users of the software application in a real environment. Beta testing is performed by clients (black)

15) What is Smoke testing?

Smoke testing, also called build verification testing or confidence testing, is a software testing method that is used to determine if a new software build is ready for the next testing phase. purpose of smoke testing is to determine whether the build software is testable or not.

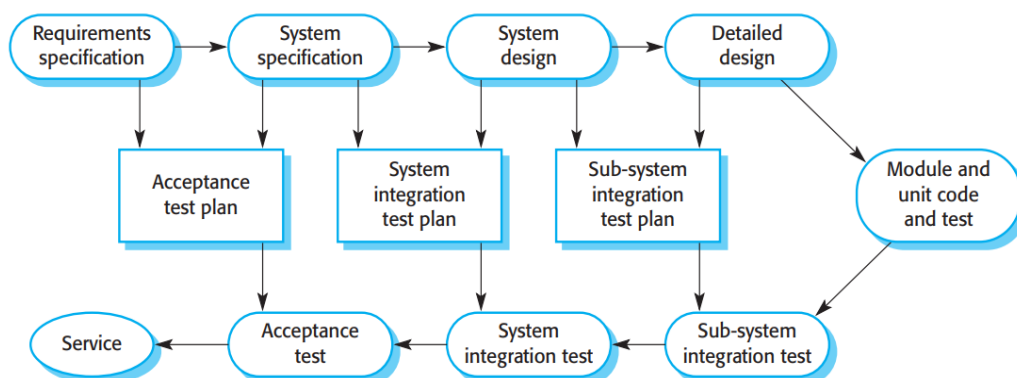
16) What is test case design?

Test case design is a document that defines the scope and strategy for testing. It involves planning and creating a set of instructions that will be used to verify whether a software application meets its requirements and functions as intended. This process helps to ensure the quality and reliability of the software product.

4 mark:

1) Explain V model (planning Verification and Validation) with diagram.

The V-model is a type of [SDLC model](#) where the process executes in a sequential manner in a V-shape. It is also known as the Verification and Validation model. It is based on the association of a testing phase for each corresponding development stage. The development of each step is directly associated with the testing phase. The next phase starts only after completion of the previous phase i.e., for each development activity, there is a testing activity corresponding to it.



- **Requirements specification:** This document defines the requirements that the system must meet.
- **System specification:** This document describes the architecture, components, and interfaces of the system.
- **System design:** This document describes the detailed design of the system, including its modules, classes, and functions.
- **Detailed design:** This document describes the detailed design of each module of the system.
- **Acceptance test plan:** This document outlines the tests that will be performed to verify that the system meets the requirements of the users and stakeholders.
- **System integration test plan:** This document outlines the tests that will be performed to verify that the different modules of the system interact correctly.
- **Sub-system integration test plan:** This document outlines the tests that will be performed to verify that the different sub-systems of the system interact correctly.
- **Module and unit code and test plan:** This document outlines the tests that will be performed to verify the functionality and correctness of the individual modules and units of code in the system.
- **Service acceptance test:** This test verifies that the system meets the requirements of the users and stakeholders.
- **System integration test:** This test verifies that the different modules of the system interact correctly.
- **Sub-system integration test:** This test verifies that the different sub-systems of the system interact correctly.

2) Explain the structure of software test plan.

A Test Plan is a detailed document that catalogs the test strategies, objectives, schedule, estimations, deadlines, and resources required to complete that project.

The testing process : A description of the major phases of the testing process. Each serving a specific purpose in ensuring the quality and reliability of a software system

Requirements traceability: Users are most interested in the system meeting its requirements and testing should be planned so that all requirements are individually tested.

Tested items: The products of the software process that are to be tested should be specified.

Testing schedule: An overall testing schedule and resource allocation for this schedule is, obviously, linked to the more general project development schedule.

Test recording procedures: It is not enough simply to run tests; the results of the tests must be systematically recorded. It must be possible to audit the testing process to check that it has been carried out correctly.

Hardware and software requirements: This section should set out the software tools required and estimated hardware utilisation

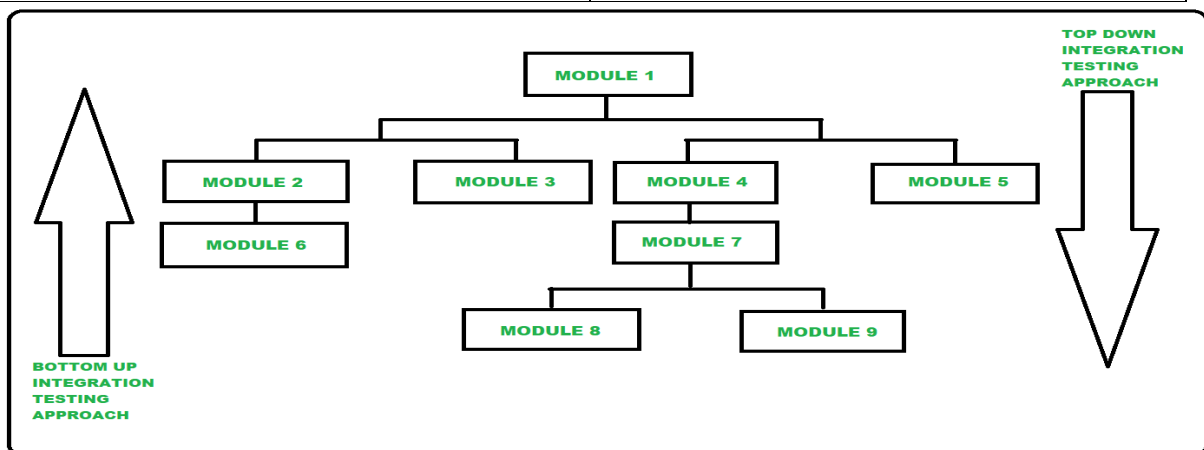
Constraints affecting the testing process such as *staff shortages* should be anticipated in this section.

3) What is system Testing? Explain two distinct phases of System Testing.

Refer 2mark

4) Explain the top-down and bottom-up integration testing?

Top Down Integration testing is one of the approach of Integration testing in which integration testing takes place from top to bottom means system integration begins with top level modules.	Bottom Up Integration testing is one of the approach of Integration testing in which integration testing takes place from bottom to top means system integration begins with lowest level modules
In this testing the higher level modules are tested first then the lower level modules are tested and then the modules are integrated accordingly.	In this testing the lower level modules are tested first then the higher level modules are tested and then the modules are integrated accordingly.
Top Down testing is more expensive	Bottom Up testing is less expensive
The complexity of this testing is simple. It works on big to small components.	The complexity of this testing is complex.



5) What are the different components that are tested in unit testing?

Component testing (sometimes called unit testing) is the process of testing individual components in the system. This is a defect testing process so its goal is to expose faults in these components.

There are different types of components that may be tested at this stage:

- Individual functions or methods within an object.
- Object classes that have several attributes and method.
- Composite components made up of several different objects or functions. These composite components have a defined interface that is used to access their functionality.

WeatherStation
identifier
reportWeather () calibrate (instruments) test () startup (instruments) shutdown (instruments)

- **WeatherStation:** The WeatherStation class is the main class in the system. It is responsible for collecting weather data from the instruments and reporting it to the user.
- **Instruments:** The Instruments class represents the different instruments that are used to collect weather data, such as a thermometer, barometer, and anemometer.
- **Calibration:** The Calibration class is used to calibrate the instruments so that they provide accurate readings.
- **Startup:** The Startup class is responsible for initializing the WeatherStation and its instruments.
- **Shutdown:** The Shutdown class is responsible for shutting down the WeatherStation and its instruments.

6) What are the different types of interface errors in interface testing?

parameter interfaces are interfaces through which data or, at times, references to functions are transferred from one software component to another. This type of interface plays a pivotal role in ensuring smooth communication and interaction between different components of a software system

shared memory interfaces, components of a system use a common block of memory to share information. One part puts data into this shared memory, and other parts retrieve that data from the same memory location. Imagine it as a shared whiteboard: one part of the system writes data on it, and other parts read that information.

Procedural interfaces involve one component providing a set of procedures that other components can call. This type of interface is commonly found in objects and reusable components. In simpler terms, it's like having a set of predefined actions that different parts of a system can use or interact with.

Message passing interfaces work by one component asking another for a service by sending a message. When the service is executed, the results are sent back in a

return message. This type of interface is common in object-oriented systems and client-server setups. In simpler terms, it's like sending a request to another part of a system, and when that part completes the task, it sends back the results.

7) What are interface errors? Mention different classes of interface errors.

Interface errors refer to issues or problems that arise in the communication and interaction between different components or systems. These errors can occur when there are inconsistencies, miscommunications, or failures in how interfaces are defined or utilized.

Interface misuse occurs when a component makes an error in using the interface of another component. This often happens with parameter interfaces, where mistakes can be made regarding the type, order, or number of parameters passed. In simpler terms, it's like one part of a system trying to communicate with another, but there are mistakes in how they exchange information – for example, sending the wrong kind of data, putting it in the wrong order, or sending too much or too little information.

Interface misunderstanding occurs when a component misinterprets or misunderstands the specifications of the interface of another component. In simpler terms, it's like one part of a system expecting something specific from another part, but due to a misunderstanding, the second part behaves differently than anticipated. For instance, calling a binary search routine with an unordered array can lead to unexpected behavior because the calling component assumes the array is ordered, which is not the case. This misunderstanding can result in errors or failures in the system.

Timing errors in real-time systems with shared memory or message-passing interfaces occur when the producer and consumer of data operate at different speeds. Without careful interface design, the consumer might access outdated information because the producer hasn't updated the shared interface information. In simpler terms, it's like a mismatch in the timing between two components – one is creating information, but the other might try to use it before it's properly updated, leading to potential inaccuracies or errors.

8) Write the guidelines of interface testing?

A further problem may arise because of interactions between faults in different modules or objects. Faults in one object may only be detected when some other object behaves in an unexpected way.

Some general guidelines for interface testing are:

1. Examine the code to be tested and explicitly list each call to an external component. Design a set of tests where the values of the parameters to the external components are at the extreme ends of their ranges. These extreme values are most likely to reveal interface inconsistencies.
2. Where pointers are passed across an interface, always test the interface with null pointer parameters.

3. Where a component is called through a procedural interface, design tests that should cause the component to fail. Differing failure assumptions are one of the most common specification misunderstandings.
4. Use stress testing, as discussed in the previous section, in message-passing system. Design tests that generate many more messages than are likely to occur in practice. Timing problems may be revealed in this way.
5. Where several components interact through shared memory, design tests that Vary the order in which these components: are activated. These tests may reveal implicit assumptions made by the programmer about the order in which the shared data is produced and consumed.

9) What is test case design? Explain the various approaches involved in test case design

Test case design is a part of system and component testing where you design the test cases (inputs and predicted outputs) that test the system. The goal of the test case design process is to create a set of test cases that are effective in discovering program defects and showing that the system meets its requirements.

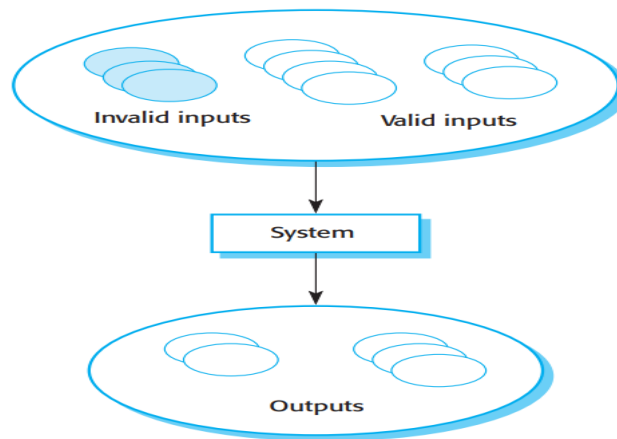
Requirements-based testing (RBT) is a testing method that evaluates if a system meets the functional and non-functional requirements of the customer. The goal of RBT is to determine if the software product meets its intended objectives

Partition testing which input data is divided into partitions of valid and invalid values, and it is mandatory that all partitions must exhibit the same behavior. If a condition of one partition is true, then the condition of another equal partition must also be true, and if a condition of one partition is false, then the condition of another equal partition must also be false. The principle of equivalence partitioning is, test cases should be designed to cover each partition at least once. Each value of every equal partition must exhibit the same behavior as other.(otp)

Structural testing where you use knowledge of the program's structure to design tests that exercise all parts of the program. Essentially, when testing a program, you should try to execute each statement at least once. Structural testing helps identify test cases that can make this possible.

10) Explain Equivalence Partitions with a diagram?

Equivalence partitioning is a technique of software testing in which input data is divided into partitions of valid and invalid values, and it is mandatory that all partitions must exhibit the same behavior. If a condition of one partition is true, then the condition of another equal partition must also be true, and if a condition of one partition is false, then the condition of another equal partition must also be false. The principle of equivalence partitioning is, test cases should be designed to cover each partition at least once. Each value of every equal partition must exhibit the same behavior as other.



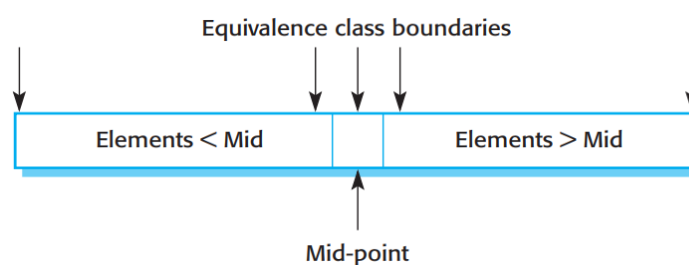
1. Inputs where the key element is a member of the sequence (Found = true)
2. Inputs where the key element is not a sequence member (Found = false)

Assume that there is a function of a software application that accepts a particular number of digits, not greater and less than that particular number. For example, an OTP number which contains only six digits, less or more than six digits will not be accepted, and the application will redirect the user to the error page.

11) Explain structural testing?

Structural testing where you use knowledge of the program's structure to design tests that exercise all parts of the program. Essentially, when testing a program, you should try to execute each statement at least once. Structural testing helps identify test cases that can make this possible.

Structural testing is an approach to test case design where the tests are derived from knowledge of the software's structure and implementation. This approach is sometimes called 'white-box', 'glass-box' testing, or "clear-box' testing .



12) Explain path testing?

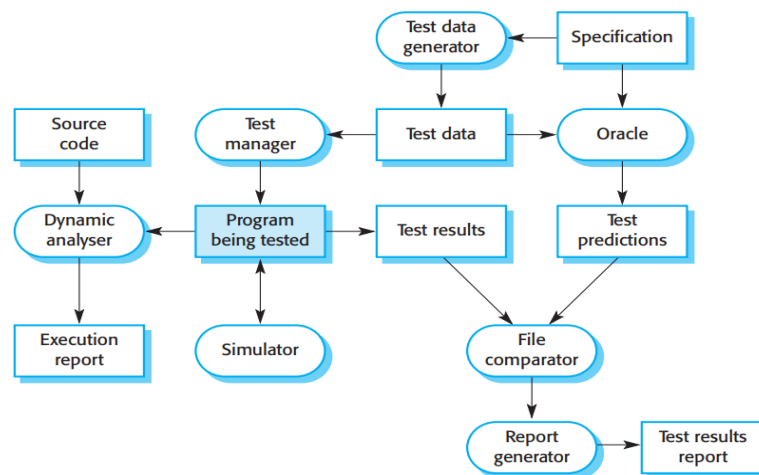
path testing is a structural testing method that involves using the source code of a program in order to find every possible executable path. It helps to determine all faults lying within a piece of code. This method is designed to execute all or selected path through a computer program.

Any software program includes, multiple entry and exit points. Testing each of these points is a challenging as well as time-consuming. In order to reduce the redundant tests and to achieve maximum test coverage, basis path testing is used.

1. Path testing method reduces the redundant tests.
2. Path testing focuses on the logic of the programs.
3. Path testing is used in test case design.

13) Explain Test automation with diagram

Test automation is the process of using software tools to automate the execution of software tests. This allows testers to execute tests faster, more reliably, and with less human effort.



1. **Test data generator:** The test data generator is used to generate the test data that will be used to test the application. The test data should be representative of the real-world data that the application will be used with.
2. **Test manager:** The test manager is responsible for overseeing the test cycle. They work with the test team to develop the test plan, execute the test cases, and report the results.
3. **Oracle:** The oracle is a known good result that is used to compare the actual test results to.
4. **File comparator:** The file comparator is a tool that is used to compare two files. This can be used to compare the actual test results to the expected results.
5. **Report generator:** The report generator is a tool that is used to generate the test report. The test report should include information about the test cases that were executed, the results of those tests, and any defects that were found.
6. **Dynamic analyser:** The dynamic analyser is a tool that is used to analyze the behavior of the application while it is running. It can be used to identify defects in the application and to measure its performance.
7. **Simulator:** The simulator is a tool that is used to simulate the environment in which the application will be used. This can be used to test the application in a variety of conditions.

14) Explain activities involved in smoke testing.

1. Software components that have been translated into code are integrated into a build. A build includes all data files, libraries, reusable modules, and engineered components that are required to implement one or more product functions.

2. A series of tests is designed to expose errors that will keep the build from properly performing its function. The intent should be to uncover “showstopper” errors that have the highest likelihood of throwing the software project behind schedule.

3. The build is integrated with other builds, and the entire product (in its current form) is smoke tested daily. The integration approach may be top down or bottom up

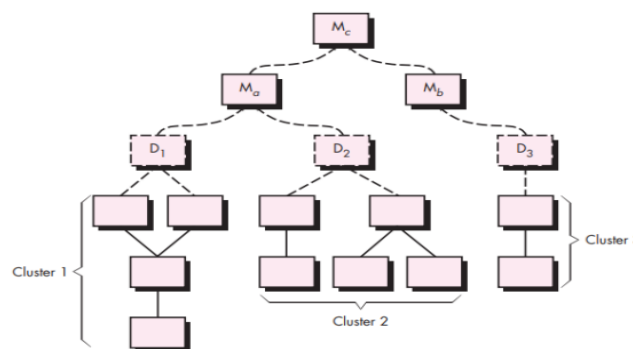
15) Explain steps involved in bottom-up integration testing.

1. Low-level components are combined into clusters (sometimes called builds) that perform a specific software subfunction.

2. A driver (a control program for testing) is written to coordinate test case input and output.

3. The cluster is tested.

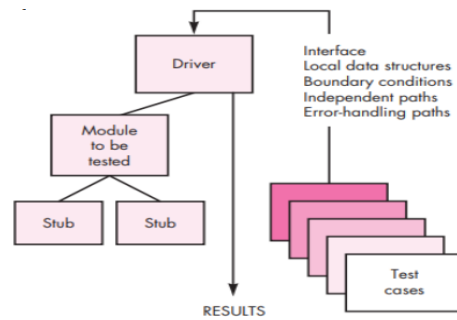
4. Drivers are removed and clusters are combined moving upward in the program structure



Integration follows the pattern illustrated above. Components are combined to form clusters 1, 2, and 3. Each of the clusters is tested using a driver (shown as a dashed block). Components in clusters 1 and 2 are subordinate to M_a . Drivers D_1 and D_2 are removed, and the clusters are interfaced directly to M_a . Similarly, driver D_3 for cluster 3 is removed prior to integration with module M_b . Both M_a and M_b will ultimately be integrated with component M_c , and so forth.

16) Explain Unit test procedures with diagram

A unit test is a type of software testing that tests individual units of code, such as functions, methods, or classes. Unit tests are typically written by the developers of the code being tested, and they are run frequently to ensure that the code is working as intended.



The diagram shows the following components of a unit test:

- **Driver:** The driver is a piece of code that is used to call the unit under test and to provide it with input data.
 - **Module to be tested:** The module to be tested is the unit of code that is being tested.
 - **Stubs:** Stubs are pieces of code that simulate the behavior of other units of code that the module under test depends on.
 - **Test cases:** Test cases are scenarios that are used to test the module under test.
 - **Results:** The results of the unit test are compared to the expected results to determine whether the test passed or failed.
-
- ✚ **Interface:** The interface of the module to be tested.
 - ✚ **Local data structures:** The local data structures that are used by the module to be tested.
 - ✚ **Boundary conditions:** The boundary conditions of the module to be tested.
 - ✚ **Independent paths:** The independent paths through the code of the module to be tested.
 - ✚ **Error-handling paths:** The error-handling paths of the module to be tested.