USN | | | | | | | | | |

22MCA23

Second Semester MCA Degree Examination, June/July 2024
Software Engineering

Time: 3 hrs. Max. Marks: 100

Note: 1. Answer any FIVE full questions, choosing ONE full question from each module.
2. M : Marks , L: Bloom's level , C: Course outcomes.

		Module – 1	M	L	C
Q.1	a.	Explain the essential attributes of good software that are common to all software process.	10	L2	CO1
	b.	Bring out the significance of being ethical and moral responsibility of respected software professional.	10	L1	CO2
		OR			
Q.2	a.	Differentiate between water fall and incremental development process.	10	L2	CO1
	b.	Explain the phases of Rational Unified Process with a neat diagram and example.	10	L1	CO2
		Module – 2			
Q.3	a.	Differentiate between Agile and Plan driven methodologies.	10	L2	CO2
	b.	What are Functional and Non-functional requirements? Explain the different types of Non-functional requirements.	10	L2	CO3
		OR			
Q.4	a.	Discuss requirement Engineering process with a neat diagram.	07	L2	CO2
	b.	Explain requirements elicitation and analysis process.	07	L2	CO3
	c.	What are the requirements validation techniques? Explain briefly	06	L2	CO2
		Module – 3			
Q.5	a.	What is Object Oriented Design? Describe the stages of object oriented methodology used in software development.	10	L2	CO4
	b.	Describe the three models which support for modeling system in different view points.	10	L4	CO5
		OR			
Q.6	a.	Write short notes on: i) Generalization ii) Ordering iii) Bags and sequence iv) Multiplicity v) N-array association	10	L2	CO4
	b.	Draw a class diagram for library management system and explain working process in detail.	10	L4	CO5
		Module – 4			
Q.7	a.	What is use case diagram? Explain the importance of use case modeling.	10	L2	CO4
	b.	Draw a sequence diagram for weather forecasting system and explain the functionality in detail.	10	L4	CO5
		OR			
Q.8	a.	Discuss the importance of Behavioral model with suitable example.	10	L2	CO4
	b.	What is design pattern? Explain four elements of design pattern.	10	L4	CO5
		Module – 5			
Q.9	a.	Write in detail any two black box testing techniques with example.	10	L4	CO4
	b.	Justify when to use verification and validation with suitable example.	10	L4	CO5
		OR			
Q.10	a.	Define Program Evolution dynamics. Discuss Lehman's law for program evolution dynamics.	10	L4	CO4
	b.	Explain the four strategic options of Legacy System Management.	10	L4	CO5

* * * * *

1.a. Explain the essential attributes of good software.
Ans.
The essential attributes of good software are:

Acceptability - Software must be acceptable to the type of users for which it is designed. This means that it must be understandable, usable, and compatible with other systems that they use.

Dependability and security - Software dependability includes a range of characteristics including reliability, security, and safety. Dependable software should not cause physical or economic damage in the event of system failure. Software has to be secure so that malicious users cannot access or damage the system.

Efficiency - Software should not make wasteful use of system resources such as memory and processor cycles. Efficiency therefore includes responsiveness, processing time, resource utilization, etc.

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

b. Bring out the significance of being ethical and moral responsibility of respected software professional

The significance of being ethical and morally responsible as a software professional is immense, given the profound impact that software has on society. Here are key points highlighting this importance:

1. Building Trust and Credibility

- Ethical behavior ensures that clients, users, and stakeholders trust the software professional and their work.
- Trust is essential for fostering long-term relationships and professional growth.

2. Protecting User Data and Privacy

- Software professionals often handle sensitive personal and financial data. Acting ethically means ensuring confidentiality and compliance with data protection regulations.
- Misuse or negligence in handling user data can lead to legal repercussions and harm individuals.

3. Ensuring Quality and Reliability

- Ethical professionals prioritize delivering high-quality, reliable, and safe software.
- This responsibility ensures that software functions as intended and minimizes harm or inconvenience to users.

4. Avoiding Harm

- Ethical decision-making helps prevent creating software that could be misused or cause harm, such as tools for illegal activities or spreading misinformation.
- Professionals must be vigilant about the unintended consequences of their work.

5. Contributing to Social Good

- Software professionals play a crucial role in building systems that improve lives, enhance accessibility, and contribute positively to society.
- Being morally responsible encourages professionals to consider inclusivity and equity in their designs.

6. Upholding Professional Standards

- Following ethical guidelines established by professional bodies (e.g., ACM, IEEE) helps maintain the integrity of the profession.
- It ensures accountability and inspires other professionals to adhere to high standards.

7. Legal Compliance

- Ethical behavior aligns with laws and regulations, reducing the risk of legal issues for both the individual and their organization.
- This includes adherence to intellectual property rights, licensing agreements, and anti-piracy laws.

2. a Differentiate between waterfall and incremental process models.

Waterfall Model:

There are separate identified phases in the waterfall model:

- ☐ Requirements analysis and definition
- ☐ System and software design
- ☐ Implementation and unit testing
- ☐ Integration and system testing

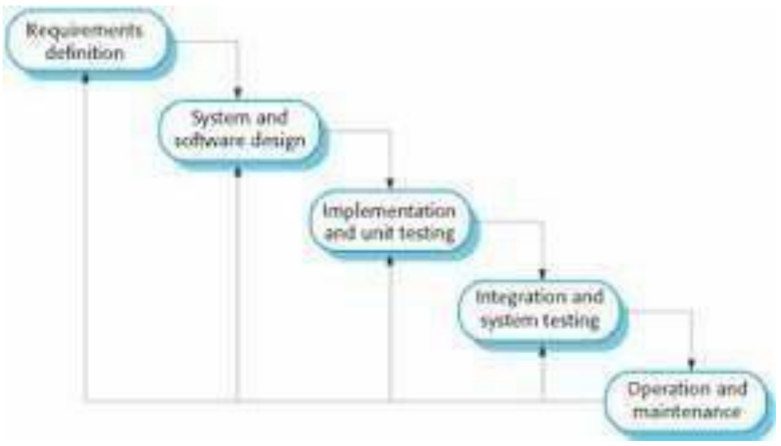
- Operation and maintenance

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

Problems:

- Inflexible partitioning of the project into distinct stages makes it difficult to respond to changing customer requirements.
- Therefore, this model is only appropriate when the requirements are well- understood and changes will be fairly limited during the design process.
- Few business systems have stable requirements.
- The waterfall model is mostly used for large systems engineering projects where a system is developed at several sites.

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


Requirements definition
System and software design
Implementation and unit testing
Integration and system testing
Operation and maintenance

Incremental Development Model:

The cost of accommodating changing customer requirements is reduced.

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

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

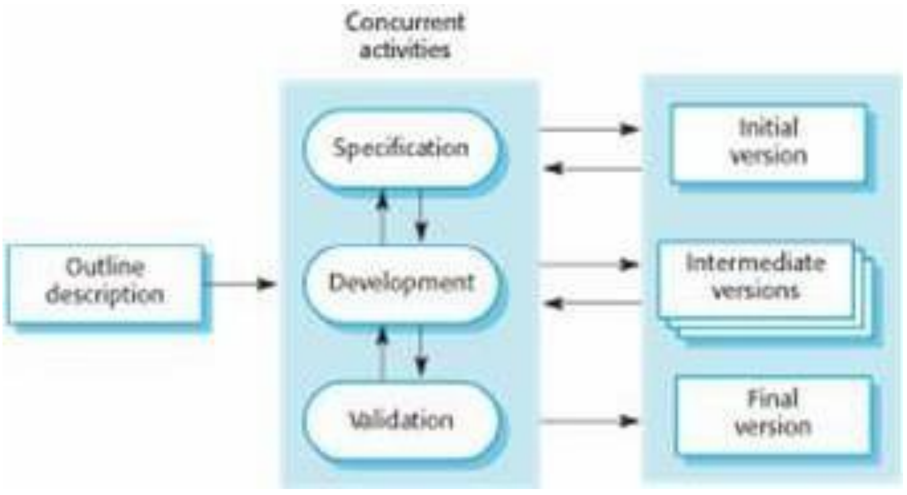
Customers can comment on demonstrations of the software and see how much has been implemented.

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

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

Problems:

- The process is not visible.
- Managers need regular deliverables to measure progress. If systems are developed quickly, it is not cost-effective to produce documents that reflect every version of the system.
- System structure tends to degrade as new increments are added.
- Unless time and money is spent on refactoring to improve the software, regular change tends to corrupt its structure. Incorporating further software changes becomes increasingly difficult and costly.

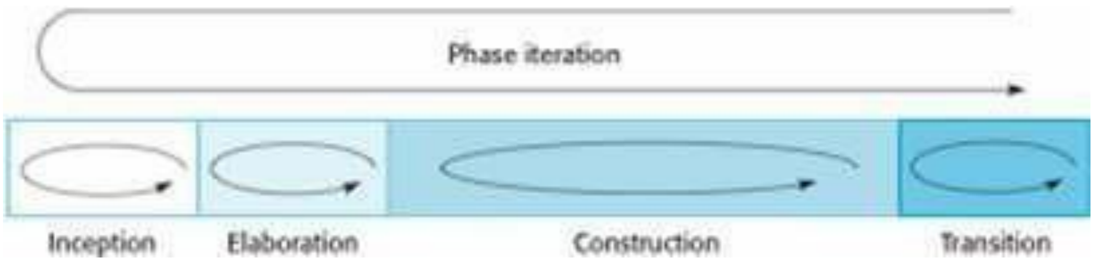


Changes are usually incorporated in documents without following any standard procedure. Thus, verification of all such changes often becomes difficult.
The development ofl1igh-quality and reliable software requires the software to be thoroughly tested. Though thorough testing of software consumes the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

2.b. Explain the phases of RUP with a neat diagram.

The Rational Unified Process:

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



1. Inception Phase

The Inception phase focuses on understanding the project’s scope, feasibility, and primary goals. Key activities include defining the vision, identifying critical requirements, creating an initial business case, and sketching a high-level architecture. This phase ensures that stakeholders align on objectives and provides a foundation for subsequent phases.

2. Elaboration Phase

The Elaboration phase aims to establish a stable architecture to address high-risk elements. During this phase, detailed requirements are gathered, architectural decisions are validated, and prototypes or simulations may be developed. This phase reduces technical risks and ensures the project is on solid ground before entering full-scale development.

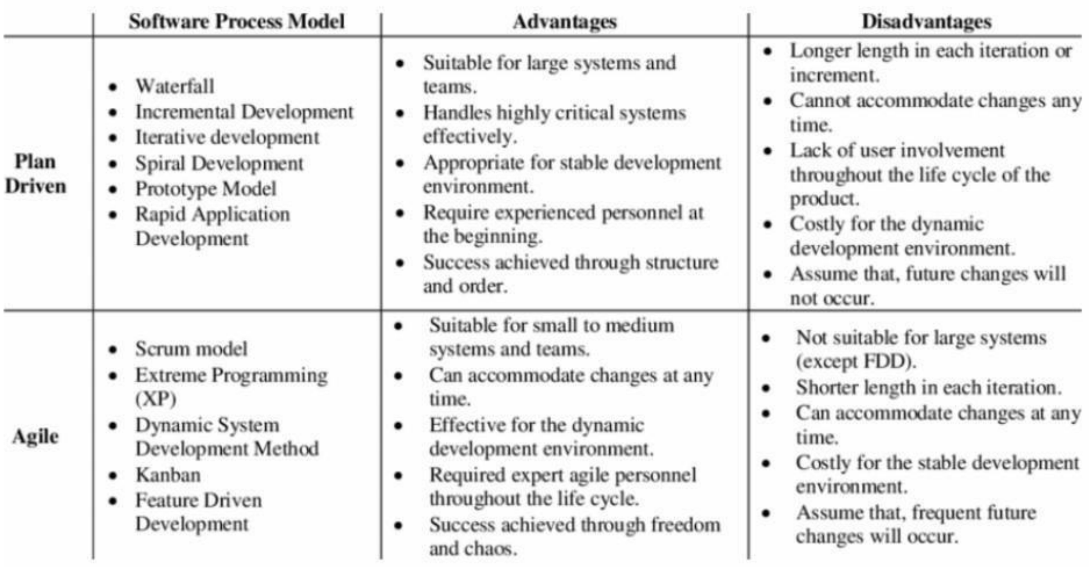
3. Construction Phase

The Construction phase focuses on building the software system based on the elaborated architecture. Development teams create code, integrate components, and perform initial testing. The emphasis is on producing a functional system that meets defined requirements, ready for system testing and deployment.

4. Transition Phase

The Transition phase involves deploying the system to users and ensuring it meets their needs in the operational environment. Activities include final testing, addressing bugs, user training, and preparing for maintenance. The goal is to deliver a fully functional product to stakeholders.

3.a. Differentiate between Agile and Plan driven methodologies.



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

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

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

Ans. Functional requirements

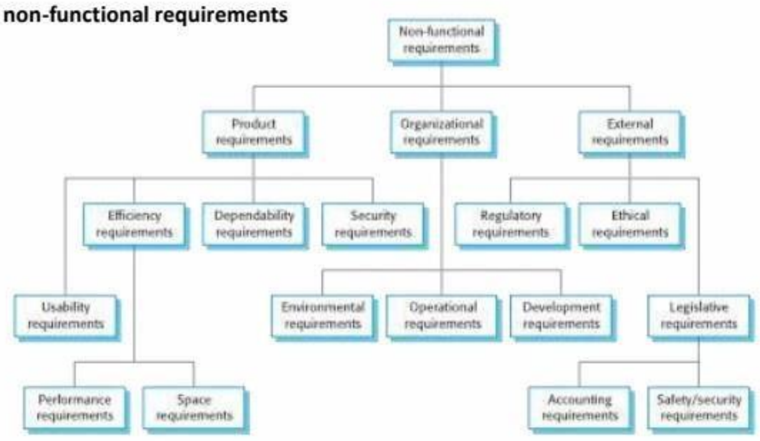
a. Statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations.
b. May state what the system should not do.

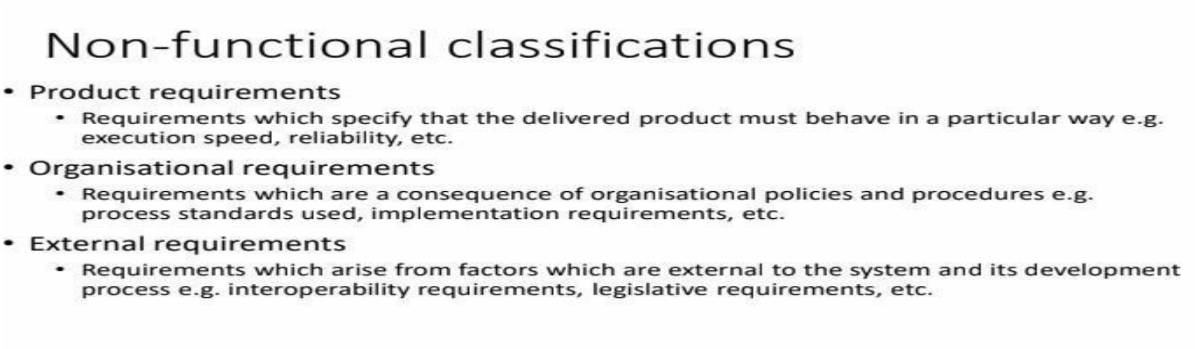
Non-functional requirements

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

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

Types of non-functional requirements



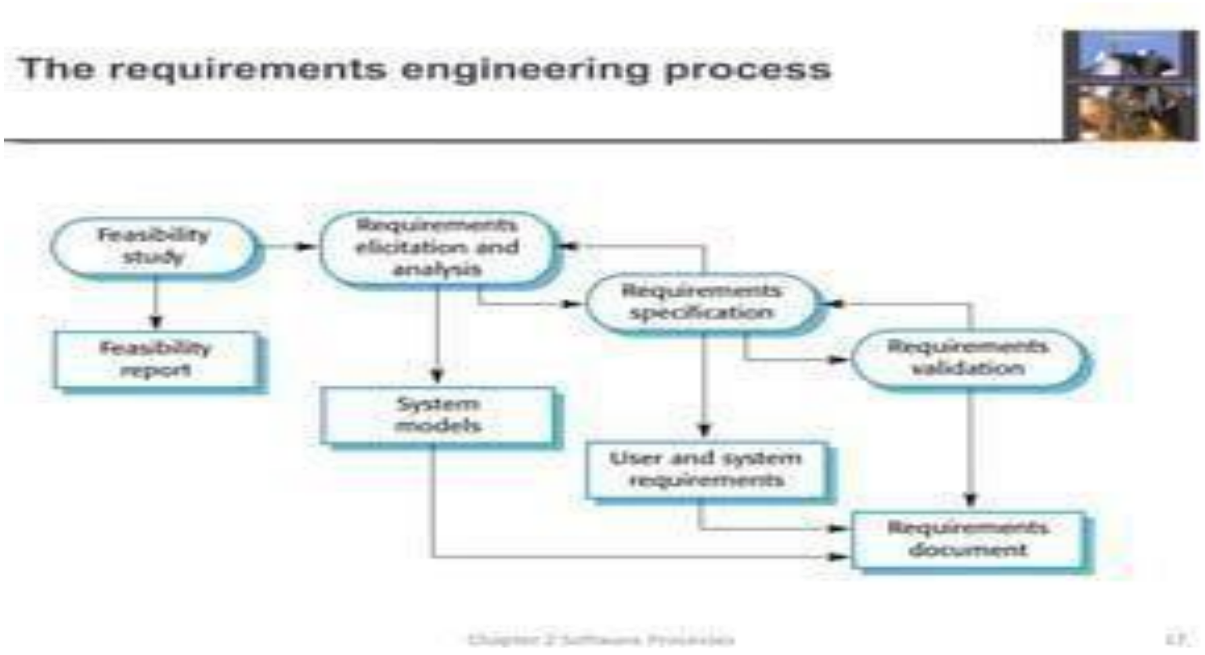


Non-functional classifications

- Product requirements
  - Requirements which specify that the delivered product must behave in a particular way e.g. execution speed, reliability, etc.
- Organisational requirements
  - Requirements which are a consequence of organisational policies and procedures e.g. process standards used, implementation requirements, etc.
- External requirements
  - Requirements which arise from factors which are external to the system and its development process e.g. interoperability requirements, legislative requirements, etc.

1. 4. a **Define Requirements Engineering. Explain the activities of requirement engineering process.** Requirement Engineering is the process of defining, documenting and maintaining the requirements. The requirements engineering process is an iterative process that involves several steps, including:

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


The requirements engineering process
Feasibility study
Requirements elicitation and analysis
Requirements specification
Requirements validation
Feasibility report
System models
User and system requirements
Requirements document
Chapter 2 Software Processes

4. b. 4.b Explain the different Techniques for Requirement elicitation and analysis?

Requirement elicitation and analysis are crucial phases in software development, where the needs and expectations of stakeholders are gathered and understood. Different techniques are used to ensure that the requirements are comprehensive, clear, and aligned with the project's objectives. Here's an overview of some common techniques:

1. **Interviews**

   One-on-one or group discussions with stakeholders to gather detailed information about their needs, expectations, and constraints. **Types:** Structured (predefined set of questions), Unstructured (open-ended), and Semi-structured (a mix of both).
   - **Advantages:** Direct interaction helps in understanding complex requirements and clarifying doubts immediately.
   - **Challenges:** Time-consuming and may lead to incomplete or biased information if not conducted properly.
2. **Surveys/Questionnaires**
   - A set of written questions distributed to stakeholders to collect their requirements, preferences, and opinions.
   - **Advantages:** Can reach a large number of stakeholders quickly and is cost-effective.
   - **Challenges:** Limited depth of information, and the quality of responses can vary.
3. **Workshops**
   - Collaborative sessions where stakeholders and developers work together to identify and discuss requirements.
   - **Advantages:** Facilitates brainstorming, encourages collaboration, and helps in building a consensus.
   - **Challenges:** Requires careful planning and skilled facilitation to be effective.
4. **Focus Groups**
   - A guided discussion with a selected group of stakeholders to gain insights into their needs and attitudes.
   - **Advantages:** Allows for in-depth exploration of specific topics and helps in identifying common requirements and concerns.
   - **Challenges:** Group dynamics can influence the discussion, and it may not represent the views of all stakeholders.
5. **Observation**
   - Observing end-users in their working environment to understand their tasks, challenges, and interactions with existing systems.
   - **Techniques:** Passive observation (no interaction) or Active observation (with interaction).
   - **Advantages:** Provides real-world insights into how users interact with systems and can uncover hidden requirements.
   - **Challenges:** Time-consuming and may not capture all possible scenarios.
6. **Prototyping**
   - Developing a preliminary version of the system to visualize and validate requirements with stakeholders.
   - **Types:** Low-fidelity (paper sketches) and High-fidelity (interactive models).
   - **Advantages:** Helps in identifying requirements early, reduces misunderstandings, and allows stakeholders to provide feedback.
   - **Challenges:** Can be resource-intensive and might lead to unrealistic expectations if not managed properly.
7. **Document Analysis**
   - Reviewing existing documentation, such as business plans, project charters, or legacy systems, to extract relevant requirements.
   - **Advantages:** Provides a historical perspective and can help in identifying regulatory, compliance, or operational requirements.
   - **Challenges:** Documents may be outdated or incomplete, leading to gaps in understanding.
8. **Brainstorming**
   - A group activity focused on generating a wide range of ideas or solutions related to the requirements.
   - **Advantages:** Encourages creative thinking and can help in discovering new or innovative requirements.
   - **Challenges:** Can result in too many ideas, some of which may be impractical or irrelevant.
9. **Use Case/Scenario Analysis**

- Creating detailed narratives or scenarios that describe how the system will be used in various situations.
- **Advantages:** Helps in identifying functional requirements and understanding user interactions with the system.
- **Challenges:** Requires thorough knowledge of the business processes and can be complex to develop.

10. **Mind Mapping**
    - A visual technique that involves creating a diagram to represent ideas, tasks, or requirements.
    - **Advantages:** Helps in organizing and visualizing complex information and relationships between requirements.
    - **Challenges:** May oversimplify complex requirements or miss important details.

12. **Ethnographic Studies**
    - A deep immersion into the users' environment to understand their cultural and social contexts.
    - **Advantages:** Provides deep insights into user behavior and needs in a natural setting.
    - **Challenges:** Time-consuming and may require specialized skills to interpret the findings.

4. c. **What are different Requirement Validation Techniques.**

1. Requirements Reviews

Requirements reviews involve a systematic examination of documented requirements by stakeholders, developers, and testers. These reviews ensure that requirements are clear, complete, and aligned with the project's goals. They can be formal, such as structured meetings with predefined agendas, or informal, like peer reviews. This technique is effective in identifying inconsistencies, ambiguities, and gaps in the requirements.

2. Prototyping

Prototyping involves developing a simplified model or version of the system or specific features to validate requirements. Stakeholders interact with the prototype to verify if it aligns with their expectations and suggest improvements. This technique is particularly useful for identifying unclear, incomplete, or misunderstood requirements early in the development process.

3. Model Validation

Model validation uses diagrams or visual representations, such as use case diagrams, data flow diagrams, or state diagrams, to review requirements. These models provide a clear and structured way to represent requirements, making it easier to identify inconsistencies, omissions, or ambiguities. This approach is helpful in validating both functional and non-functional requirements.

4. Walkthroughs

Walkthroughs are collaborative sessions where stakeholders and team members go through the requirements step by step. These sessions focus on understanding the requirements, clarifying doubts, and ensuring alignment among all parties. Walkthroughs are informal but structured discussions that help uncover errors or gaps in the requirements.

5.a What is Object Oriented Design? Describe the stages of Object Oriented methodology in software development.

Object-Oriented Design (OOD) is a software development approach that focuses on designing a system using objects, which represent real-world entities. These objects encapsulate both data (attributes) and behavior (methods), enabling modular, reusable, and maintainable software. OOD leverages principles like abstraction, encapsulation, inheritance, and polymorphism to create systems that are easier to understand, modify, and extend.

Stages of Object-Oriented Methodology in Software Development

Object-Oriented software development involves distinct stages that align with the software engineering lifecycle, emphasizing the use of objects throughout. The key stages are:

---

1. Object-Oriented Analysis (OOA)

- Focuses on understanding and modeling the problem domain.
- Identifies the key objects and their relationships based on the system's requirements.
- Produces artifacts like use case models, class diagrams, and object diagrams.
- Tools such as Unified Modeling Language (UML) are often used for visualization.

---

2. Object-Oriented Design (OOD)

- Transforms the analysis model into a design model that serves as a blueprint for implementation.
- Defines system architecture, specifying how objects interact to achieve system functionality.
- Includes class hierarchy design, defining attributes and methods, and outlining interfaces for interaction between components.
- Results in design artifacts like sequence diagrams, state diagrams, and detailed class diagrams.

---

3. Object-Oriented Programming (OOP)

- The implementation phase where the design model is translated into executable code.
- Focuses on creating and manipulating objects in an object-oriented programming language (e.g., Java, Python, C++).
- Ensures that the code adheres to the design principles and maintains object encapsulation and abstraction.

---

4. Testing and Maintenance

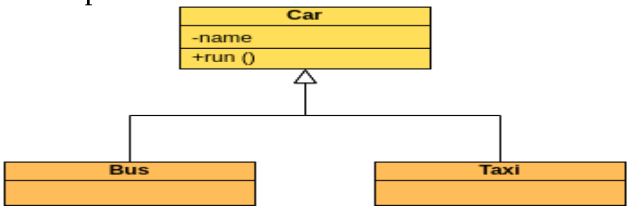
- Objects and their interactions are tested to verify the system's correctness and functionality.
- Maintenance ensures the system remains operational and adaptable to future requirements, leveraging the modularity and reusability of object-oriented components.
- Updates are often localized, thanks to the encapsulation and low coupling of objects.

1. 5.b **Describe the three models which support for modeling system in different viewpoints**.

Ans. A ***model*** is an abstraction of something for the purpose of understanding it before building it.

Different Types of Models:
The different types of modeling techniques are:
i) Class Model: It describes the structure of objects in a system – their identity, their relationships to other objects, their attributes and their operations. The goal of constructing the class model is to capture those concepts from the real world that are important to an application. Class diagram express the class model.
ii) State Model: It describes those aspects of objects concerned with time and the sequencing of operations – events that mark changes, state that define the context for events, and the organization of events and states. State diagram expresses the state model.
iii) Interaction Model: It describes interactions between objects – How individual objects collaborate to achieve the behavior of the system as a whole. Use case, sequence diagram and activity diagram documents the interaction model.
6. a. Write short notes on : i) Generalization ii)Ordering iii) Bags and sequence iv) Multiplicity v) N-ary Association
Ans

Generalization: Inheritance is also called **generalization** and is used to describe the relationship between parent and child classes. A parent class is also called a base class, and a subclass is also called a derived class. In the inheritance relationship, the subclass inherits all the functions of the parent class, and the parent class has all the attributes, methods, and subclasses. Subclasses contain additional information in addition to the same information as the parent class.



ii) Ordering :

Ordering ensures that the elements in a collection follow a specific arrangement, which can be based on criteria like numerical order, alphabetical order, or insertion order. This property is particularly useful in systems where the sequence of elements is significant, such as maintaining a sorted list of customers or tasks.

iii)Bags and Sequence

Bags

A bag, also known as a multiset, is a collection that allows duplicate elements but does not enforce any specific order. Bags are suitable for scenarios where the frequency of occurrence matters more than the order, such as tracking the count of items sold in a store.

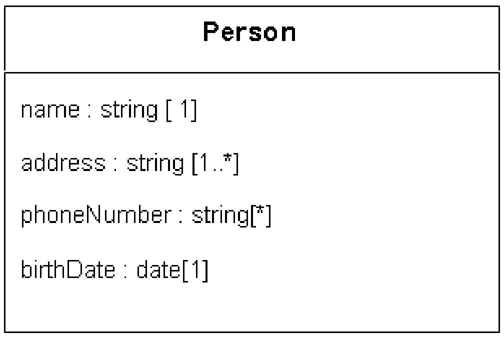
Sequence

A sequence is an ordered collection that preserves the order of insertion of its elements. Unlike a bag, it considers both order and repetition. Sequences are often used when processing items in a specific order is crucial, such as executing a series of commands or managing a playlist.

These structures provide flexibility in managing collections based on the requirements of the application.

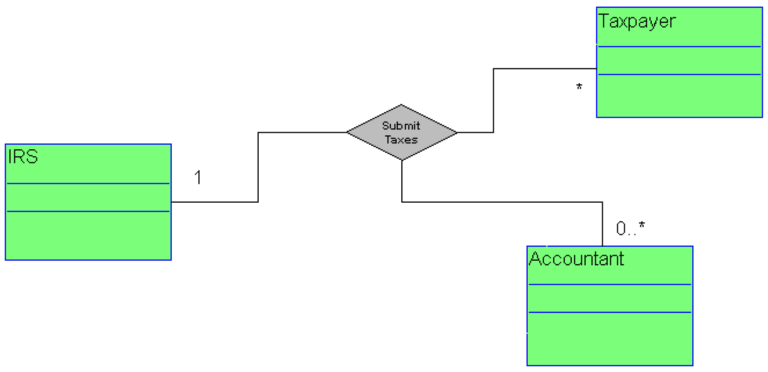
iv) Multiplicity

- Multiplicity is a collection on the cardinality of a set, also applied to attributes (database application).
- Multiplicity of an attribute specifies the number of possible values for each instantiation of an attribute. i.e., whether an attribute is mandatory ([1]) or an optional value ([0..1] or * i.e., null value for database attributes) .
- Multiplicity also indicates whether an attribute is single valued or can be a collection.

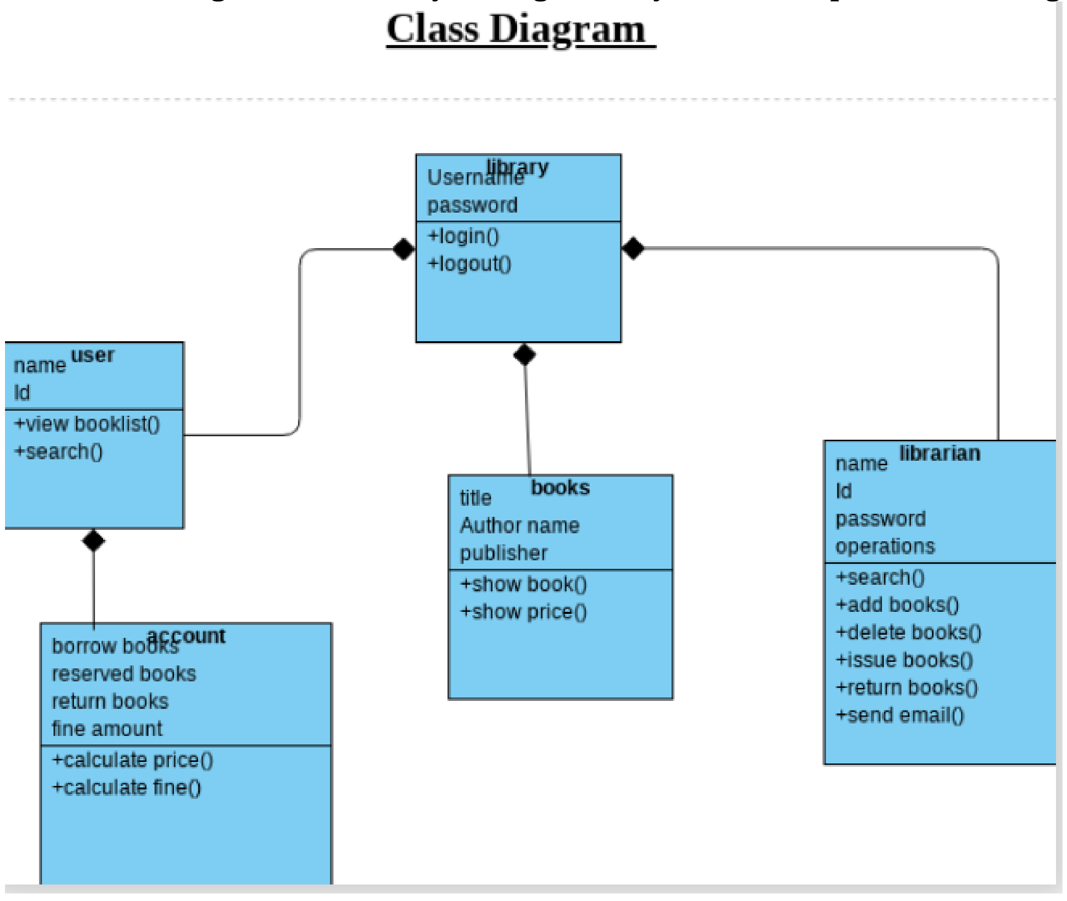


v) **N-ary association** in class modeling refers to a relationship between three or more classes. It is used to represent a situation where multiple entities are related in a meaningful way, and the relationship itself involves more than two classes. Unlike binary associations (which connect only two classes), n-ary associations involve multiple classes simultaneously.

Example of an N-ary Association



6.b Draw a class diagram for Library management system and explain its working.



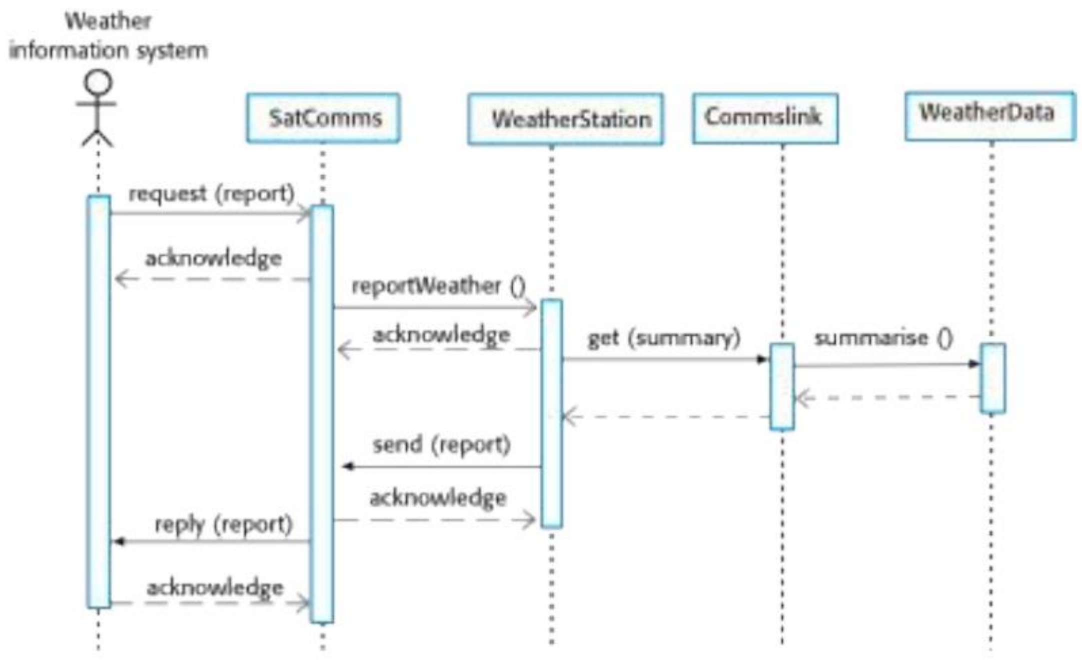
7.a What is a use case diagram? Explain the importance of use case modeling.

A **Use Case Diagram** is a visual representation in Unified Modeling Language (UML) that depicts the functional requirements of a system from the user's perspective. It shows how **actors** (users or external systems) interact with the system through **use cases** (specific functionalities or tasks). These diagrams help define the scope and boundary of the system, highlighting what the system will do without delving into implementation details.

Importance of Use Case Modeling

1. **Captures Functional Requirements**:
   Use case modeling focuses on the system's functionality, ensuring that all requirements are captured and documented clearly.
2. **Facilitates Communication**:
   By providing a visual, user-friendly representation, use case diagrams bridge the gap between technical teams and stakeholders, ensuring everyone understands the system's goals.
3. **Defines System Scope**:
   The diagrams outline the system boundaries, specifying which functionalities are part of the system and which are external, helping to avoid scope creep.
4. **Supports System Design**:
   Use case modeling serves as a foundation for system design by identifying key interactions, which guides the creation of class diagrams, sequence diagrams, and other design artifacts.
5. **Improves Test Case Development**:
   Each use case provides a basis for creating test cases, ensuring thorough testing of functionalities and meeting user expectations.
6. **Focus on User Perspective**:
   By centering on actors and their interactions, use case modeling ensures that the system aligns with user needs and improves user satisfaction.

   7.b. Draw a sequence diagram for Weather forecast system and explain the functionality.



8.a Discuss the importance of Behavioral Model.

Behavioral models are essential tools in software engineering that represent how a system behaves in response to internal and external stimuli. These models focus on the dynamic aspects of a system, such as processes, events, and interactions among components, making them crucial for understanding and designing complex systems.

Key Importance of Behavioral Models

1. **Understanding System Dynamics**:
   Behavioral models help in visualizing and analyzing how a system reacts to different inputs or events over time. This dynamic perspective complements the static structure provided by class diagrams or other static models.
2. **Defining Functional Requirements**:
   These models capture functional requirements by illustrating how the system interacts with users or other systems, ensuring that all interactions are clearly understood and documented.
3. **Improving Communication**:
   Behavioral models, such as state diagrams and sequence diagrams, provide an intuitive way for stakeholders, developers, and testers to understand system behavior, enhancing collaboration and reducing misunderstandings.
4. **Guiding System Design**:
   By modeling interactions and states, these diagrams provide a blueprint for designing system processes, ensuring that all workflows are accounted for and optimized.
5. **Supporting Testing and Validation**:
   Behavioral models form the basis for creating test cases that validate system functionality. For example, state diagrams can help testers understand all possible states and transitions to identify edge cases.
6. **Handling Complex Interactions**:
   They are particularly useful in systems with intricate logic or multiple interacting components, providing clarity and ensuring that every scenario is addressed.

Examples of Behavioral Models

- **State Diagrams**: Show how the system transitions between different states based on events.
- **Activity Diagrams**: Depict workflows or processes within the system.

8.b **What is design pattern? Explain four elements of design pattern**

The design pattern is a description of the problem and the essence of its solution, so that the solution may be reused in different settings.

The pattern is not a detailed specification.

Patterns and Pattern Languages are ways to describe best practices, good designs, and capture experience in a way that it is possible for others to reuse this experience.

Design patterns are usually associated with object-oriented design.

The general principle of encapsulating experience in a pattern is one that is equally applicable to any kind of software design

The four essential elements of design patterns were defined by the 'Gang of Four' in their patterns book:

▪ 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. This is not a concrete design description. It is a template for a design solution that can be instantiated in different ways. This isoften expressed graphically and shows the relationships between the objects and object classes in the solution.

9. a Explain in detail any two Black box testing techniques.

Black box testing focuses on evaluating a system's functionality without considering its internal code or structure. Testers provide inputs and examine outputs to ensure the system meets specified requirements. Two commonly used techniques in black box testing are **Equivalence Partitioning** and **Boundary Value Analysis**.

---

1. Equivalence Partitioning (EP)

- **Description**:
  Equivalence partitioning involves dividing input data into groups or partitions that are expected to exhibit similar behavior. Each partition is treated as a single test case, reducing the number of test cases needed while ensuring comprehensive coverage.
- **Purpose**:
  To identify representative inputs and ensure that the system handles different categories of input data correctly.
- **Example**:
  For a field accepting numbers from 1 to 100, the input can be divided into three partitions:
  - Valid range (e.g., 1 to 100)
  - Below valid range (e.g., <1)
  - Above valid range (e.g., >100)

---

2. Boundary Value Analysis (BVA)

- **Description**:
  Boundary value analysis focuses on testing the boundaries of input ranges, as errors are more likely to occur at these points. Test cases are created for values at, just below, and just above the boundaries.
- **Purpose**:
  To detect errors at the edges of input ranges where systems often fail.
- **Example**:
  For a system accepting inputs between 1 and 100:
  - Test cases: 0 (below boundary), 1 (on boundary), 100 (on boundary), and 101 (above boundary).

9.b Justify when to use verification and validation.

Verification and **Validation** are complementary processes in software engineering, ensuring that a product is both built correctly and meets user needs.

When to Use Verification?

Verification is conducted during the development process to ensure the product conforms to its specifications and design. It focuses on "building the product right."

- **Purpose**: To check if the software meets the predefined requirements and is free of defects.
- **Stage**: Early in the development lifecycle, during requirement gathering, design, and coding.
- **Example**:
    - During the **requirement phase**, a team reviews the Software Requirement Specification (SRS) document to ensure all customer requirements are documented accurately.
    - During **coding**, static testing is performed to check for syntax errors and adherence to coding standards without running the software.

When to Use Validation?

Validation is conducted after development to ensure the product meets user needs and expectations. It focuses on "building the right product."

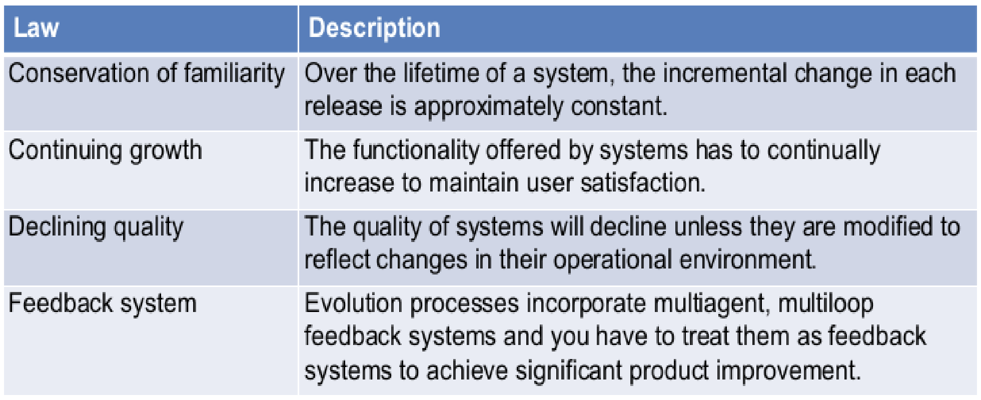
- **Purpose**: To check if the software fulfills its intended use in the real-world environment.
- **Stage**: After development, during testing phases, and at deployment.
- **Example**:
    - During **user acceptance testing (UAT)**, end users test the system to confirm it solves their problem and performs as expected in their work environment.
    - For a mobile app, validation would involve testing its responsiveness, usability, and compatibility across devices.

Aspect	Verification	Validation
Objective	Ensure product conforms to requirements.	Ensure product meets user needs.
Focus	Process-oriented ("building it right").	Product-oriented ("building the right one").
Timing	During development phases.	After development or during deployment.
Example	Code reviews, static testing.	User acceptance testing, usability testing.

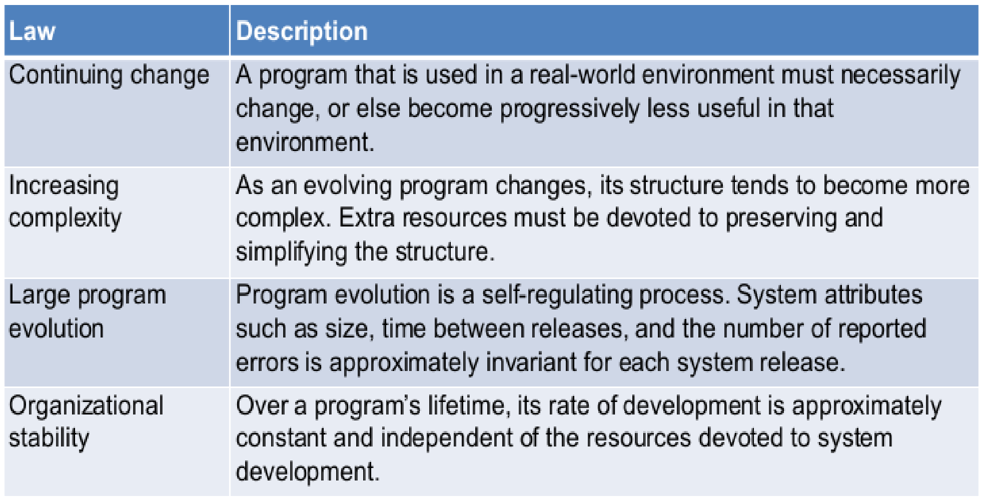
Both are essential to deliver high-quality software that is both technically accurate and user-friendly.

10.a Define 'program evolution dynamics. Discuss Lehman's Law for program evolution dynamics.

Program evolution dynamics is the study of the processes of system change.After several major empirical studies, Lehman and Belady proposed that there were a number of 'laws' which applied to all systems as they evolved.

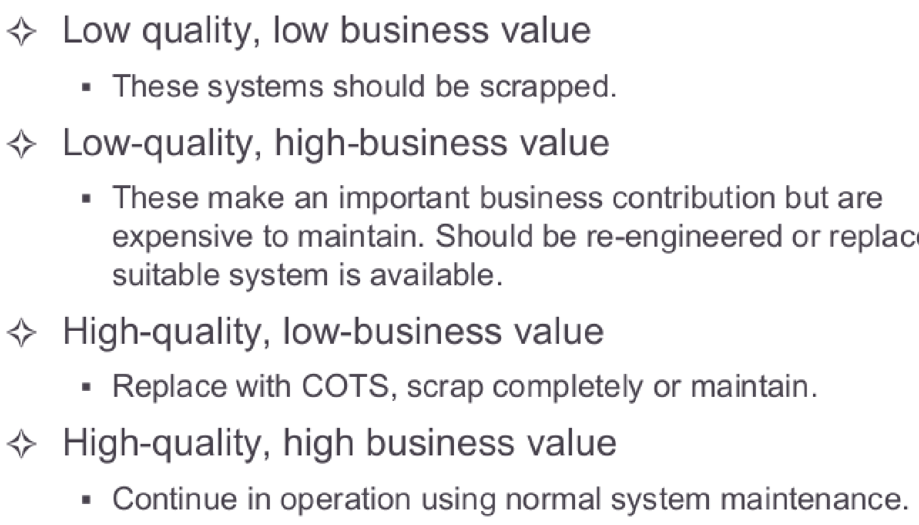


Law	Description
Conservation of familiarity	Over the lifetime of a system, the incremental change in each release is approximately constant.
Continuing growth	The functionality offered by systems has to continually increase to maintain user satisfaction.
Declining quality	The quality of systems will decline unless they are modified to reflect changes in their operational environment.
Feedback system	Evolution processes incorporate multiagent, multiloop feedback systems and you have to treat them as feedback systems to achieve significant product improvement.



Law	Description
Continuing change	A program that is used in a real-world environment must necessarily change, or else become progressively less useful in that environment.
Increasing complexity	As an evolving program changes, its structure tends to become more complex. Extra resources must be devoted to preserving and simplifying the structure.
Large program evolution	Program evolution is a self-regulating process. System attributes such as size, time between releases, and the number of reported errors is approximately invariant for each system release.
Organizational stability	Over a program's lifetime, its rate of development is approximately constant and independent of the resources devoted to system development.

10.b Explain the four strategic options of Legacy System Management.



- Low quality, low business value
  - These systems should be scrapped.
- Low-quality, high-business value
  - These make an important business contribution but are expensive to maintain. Should be re-engineered or replaced if a suitable system is available.
- High-quality, low-business value
  - Replace with COTS, scrap completely or maintain.
- High-quality, high business value
  - Continue in operation using normal system maintenance.