CMR INSTITUTE OF TECHNOLOGY	USN														

Internal Assesment Test – II, August 2022

Sub:	SOFTWARE ENGINEERING							Code:	20MCA24
Date:	30-8-2022	Duration:	90 mins	Max Marks:	50	Sem:	II	Branch:	MCA

Answer **ONE FULL QUESTION** from each part

No.	Question	Marks	OBE CO	OBE RBT
	Part-I			
1	Explain the characteristic features and Structure of a software requirements document.	[10]	CO1	L2
	(OR)			
2	Explain in detail about extreme programming.	[10]	CO1	L2
	Part-II			
3	Describe Requirements Change Management Process and planning	10	CO2	L2
	(OR)			
4	Define: (i) Functional requirements and (ii) Non-functional requirements iii) Domain requirement	[10]	CO2	L2
	Part-III			
5	Explain the different metrics of non-functional requirements.	[10]	CO1	L2
	(OR)			
6	Write Short Notes on: a) Requirement Specification b) View point, c) Scenario techniques, d) Ethnography e). Feasibility study	10	CO2	L2
	Part-IV			
7	What are the various techniques used for requirements elicitationand analysis? Explain any one technique in detail.	[10]	CO2	L2
	(OR)			
8	Define Object Orientation. Explain the three kinds of models used in Object Oriented Design method	[10]	CO1	L2
	Part-V			
9	What are the four aspects of object-oriented approach? Explain	[10]	CO1	L1
	(OR)			
10	What is a class and explain a class diagram with suitable example.	10	CO3	L3

1 Explain the characteristic features and Structure of a software requirements document.

Structure of SRS

The requirements document is devised in a manner that is easier to write, review, and maintain. It is organized into independent sections and each section is organized into modules or units. Note that the level of detail to be included in the SRS depends on the type of the system to be developed and the process model chosen for its development. For example, if a system is to be developed by an external contractor, then critical system specifications need to be precise and detailed. Similarly, when flexibility is required in the requirements and where an in-house development takes place, requirements documents can be less detailed.

Since the requirements document serves as a foundation for subsequent software development phases, it is important to develop the document in the prescribed manner. For this, certain guidelines are followed while preparing SRS. These guidelines are listed below.

1. **Functionality:** It should be separate from implementation.
2. **Analysis model:** It should be developed according to the desired behavior of a system. This should include data and functional response of a system to various inputs given to it.
3. **Cognitive model:** It should be developed independently of design or implementation model. This model expresses a system as perceived by the users.
4. **The content and structure** of the **specification:** It should be flexible enough to accommodate changes.
5. **Specification:** It should be robust. That is, it should be tolerant towards incompleteness and complexity.

The information to be included in SRS depends on a number of factors, for example, the type of software being developed and the approach used in its development. If software is developed using the iterative development process, the requirements document will be less detailed as compared to that of the software developed for critical systems. This is because specifications need to be very detailed and accurate in these systems. A number of standards have been suggested to develop a requirements document. However, the most widely used standard is by IEEE, which acts as a general framework. This general framework can be customized and adapted to meet the needs of a particular organization.

Each SRS fits a certain pattern; thus, it is essential to standardize the structure of the requirements document to make it easier to understand. For this IEEE standard is used for SRS to organize requirements for different projects, which provides different ways of structuring SRS. Note that in all requirements documents, the first two sections are the same.

This document comprises the following sections.

1. **Introduction:** This provides an overview of the entire information described in SRS. This involves purpose and the scope of SRS, which states the functions to be performed by the system. In addition, it describes definitions, abbreviations, and the acronyms used. The references used in SRS provide a list of documents that is referenced in the document.
2. **Overall description:** It determines the factors which affect the requirements of the system. It provides a brief description of the requirements to be defined in the next section called 'specific requirement'. It comprises the following sub-sections.
3. **Product perspective:** It determines whether the product is an independent product or an integral part of the larger product. It determines the interface with hardware, software, system, and communication. It also defines memory constraints and operations utilized by the user.
4. **Product functions:** It provides a summary of the functions to be performed by the software. The functions are organized in a list so that they are easily understandable by the user:
5. **User characteristics:** It determines general characteristics of the users.
6. **Constraints:** It provides the genera1 description of the constraints such as regulatory policies, audit functions, reliability requirements, and so on.
7. **Assumption and dependency:** It provides a list of assumptions and factors that affect the requirements as stated in this document.
8. **Apportioning of requirements:** It determines the requirements that can be delayed until release of future versions of the system.
9. **Specific requirements:** These determine all requirements in detail so that the designers can design the system in accordance with them. The requirements include description of every input and output of the system and functions performed in response to the input provided. It comprises the following subsections.
10. **External interface:** It determines the interface of the software with other systems, which can include interface with operating system and so on. External interface also specifies the interaction of the software with users, hardware, or other software. The characteristics of each user interface of the software product are specified in SRS. For the hardware interface, SRS specifies the logical characteristics of each interface among the software and hardware components. If the software is to be executed on the existing hardware, then characteristics such as memory restrictions are also specified.
11. **Functions:** It determines the functional capabilities of the system. For each functional requirement, the accepting and processing of inputs in order to generate outputs are specified. This includes validity checks on inputs, exact sequence of operations, relationship of inputs to output, and so on.

12. **Performance requirements:** It determines the performance constraints of the software system. Performance requirement is of two types: static requirements and dynamic requirements. **Static requirements** (also known as **capacity requirements)** do not impose constraints on the execution characteristics of the system. These include requirements like number of terminals and users to be supported. **Dynamic requirements** determine the constraints on the execution of the behavior of the

system, which includes response time (the time between the start and ending of an operation under specified conditions) and throughput (total amount of work done in a given time).
13. **Logical database of requirements:** It determines logical requirements to be stored in the database. This includes type of information used, frequency of usage, data entities and relationships among them, and so on.
14. **Design constraint:** It determines all design constraints that are imposed by standards, hardware limitations, and so on. Standard compliance determines requirements for the system, which are in compliance with the specified standards. These standards can include accounting procedures and report format. Hardware limitations implies when the software can operate on existing hardware or some pre-determined hardware. This can impose restrictions while developing the software design. Hardware limitations include hardware configuration of the machine and operating system to be used.
15. **Software system attributes:** It provide attributes such as reliability, availability, maintainability and portability. It is essential to describe all these attributes to verify that they are achieved in the final system.
16. **Organizing Specific Requirements:** It determines the requirements so that they can be properly organized for optimal understanding. The requirements can be organized on the basis of mode of operation, user classes, objects, feature, response, and functional hierarchy.
17. **Change management process:** It determines the change management process in order to identify, evaluate, and update SRS to reflect changes in the project scope and requirements.
18. **Document approvals:** These provide information about the approvers of the SRS document with the details such as approver's name, signature, date, and so on.
19. **Supporting information:** It provides information such as table of contents, index, and so on. This is necessary especially when SRS is prepared for large and complex projects.

2 Explain in detail about extreme programming.

Extreme programming (XP) is a software development methodology which is intended to improve software quality and responsiveness to changing customer requirements.

Extreme Programming – A way to handle the common shortcomings

Software Engineering involves –

- Creativity
- Learning and improving through trials and errors
- Iterations

Extreme Programming builds on these activities and coding. It is the detailed (not the only) design activity with multiple tight feedback loops through effective implementation, testing and refactoring continuously.

Extreme Programming is based on the following values –

- Communication
- Simplicity
- Feedback
- Courage
- Respect

XP is a lightweight, efficient, low-risk, flexible, predictable, scientific, and fun way to develop a software.

e**X**treme **P**rogramming (XP) was conceived and developed to address the specific needs of software development by small teams in the face of vague and changing requirements.

Extreme Programming is one of the Agile software development methodologies. It provides values and principles to guide the team behavior. The team is expected to self-organize. Extreme Programming provides specific core practices where –

- Each practice is simple and self-complete.
- Combination of practices produces more complex and emergent behavior.

Embrace Change

A key assumption of Extreme Programming is that the cost of changing a program can be held mostly constant over time.

This can be achieved with –

- Emphasis on continuous feedback from the customer
- Short iterations
- Design and redesign
- Coding and testing frequently
- Eliminating defects early, thus reducing costs
- Keeping the customer involved throughout the development
- Delivering working product to the customer

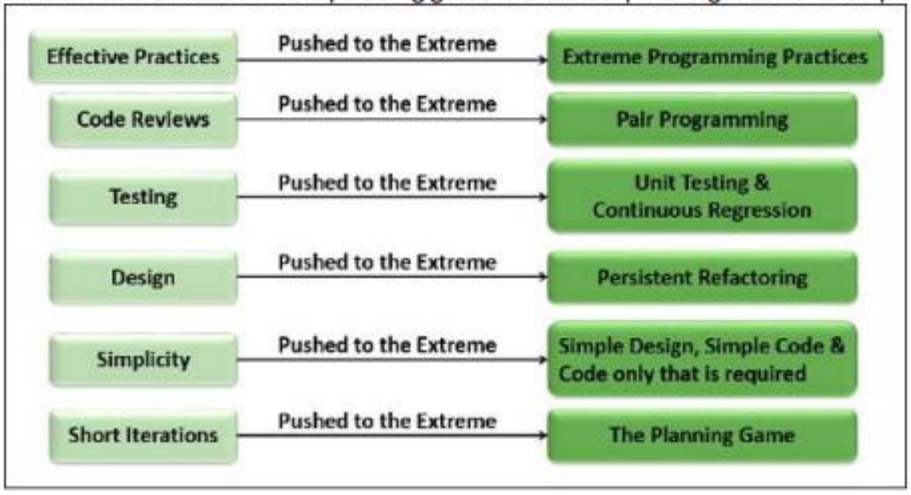
Extreme Programming in a Nutshell

Extreme Programming involves –

- Writing unit tests before programming and keeping all of the tests running at all times. The unit tests are automated and eliminate defects early, thus reducing the costs.
- Starting with a simple design just enough to code the features at hand and redesigning when required.
- Programming in pairs (called pair programming), with two programmers at one screen, taking turns to use the keyboard. While one of them is at the keyboard, the other constantly reviews and provides inputs.
- Integrating and testing the whole system several times a day.
- Putting a minimal working system into the production quickly and upgrading it whenever required.
- Keeping the customer involved all the time and obtaining constant feedback.

Extreme Programming takes the effective principles and practices to extreme levels.

- Code reviews are effective as the code is reviewed all the time.
- Testing is effective as there is continuous regression and testing.
- Design is effective as everybody needs to do refactoring daily.
- Integration testing is important as integrate and test several times a day.
- Short iterations are effective as the planning game for release planning and iteration planning.

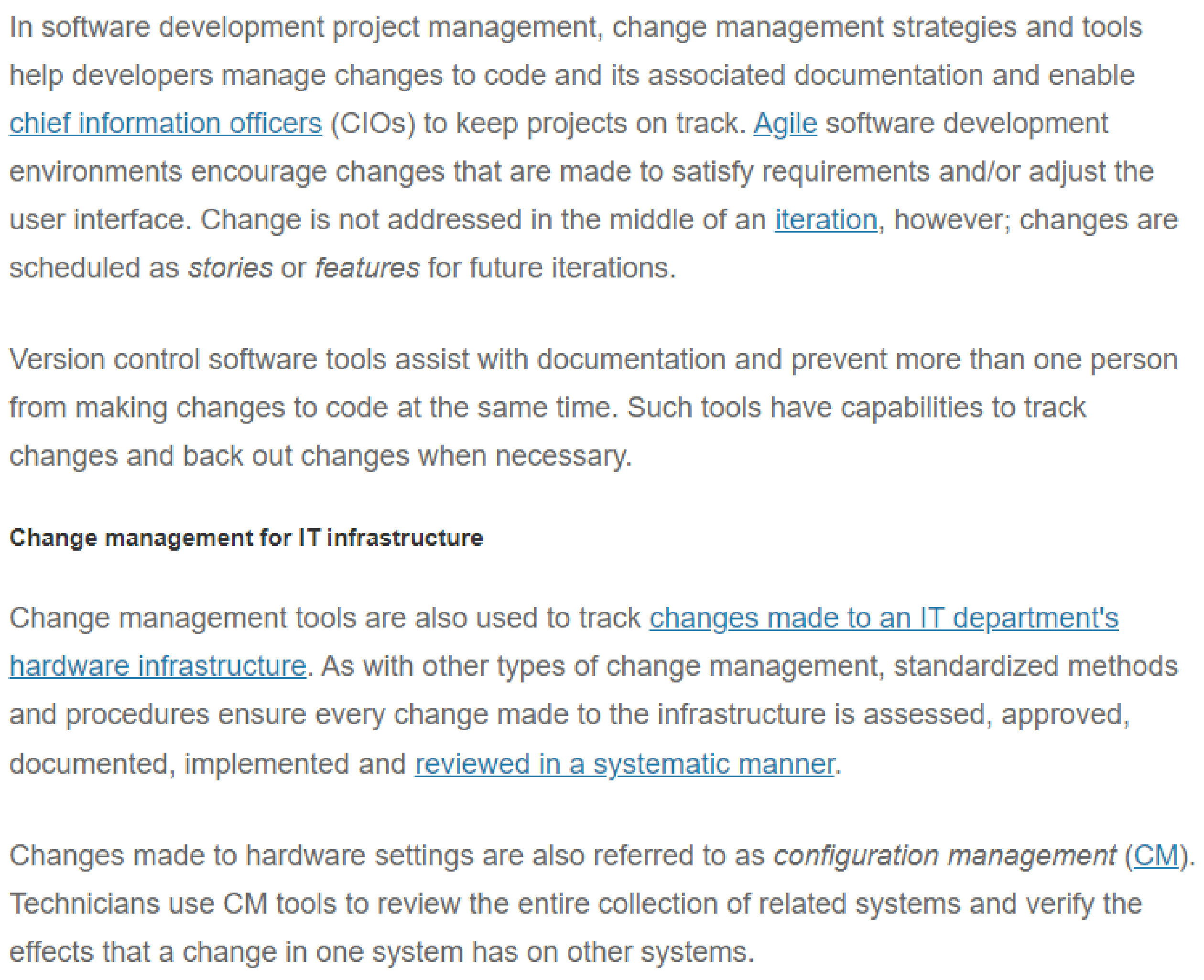


Extreme Programming Advantages

Extreme Programming solves the following problems often faced in the software development projects –

- **Slipped schedules** – and achievable development cycles ensure timely deliveries.
- **Cancelled projects** – Focus on continuous customer involvement ensures transparency with the customer and immediate resolution of any issues.
- **Costs incurred in changes** – Extensive and ongoing testing makes sure the changes do not break the existing functionality. A running working system always ensures sufficient time for accommodating changes such that the current operations are not affected.
- **Production and post-delivery defects: Emphasis is on** – the unit tests to detect and fix the defects early.
- **Misunderstanding the business and/or domain** – Making the customer a part of the team ensures constant communication and clarifications.
- **Business changes** – Changes are considered to be inevitable and are accommodated at any point of time.
- **Staff turnover** – Intensive team collaboration ensures enthusiasm and good will. Cohesion of multi-disciplines fosters the team spirit.

3 **Describe Requirements Change Management Process and planning**

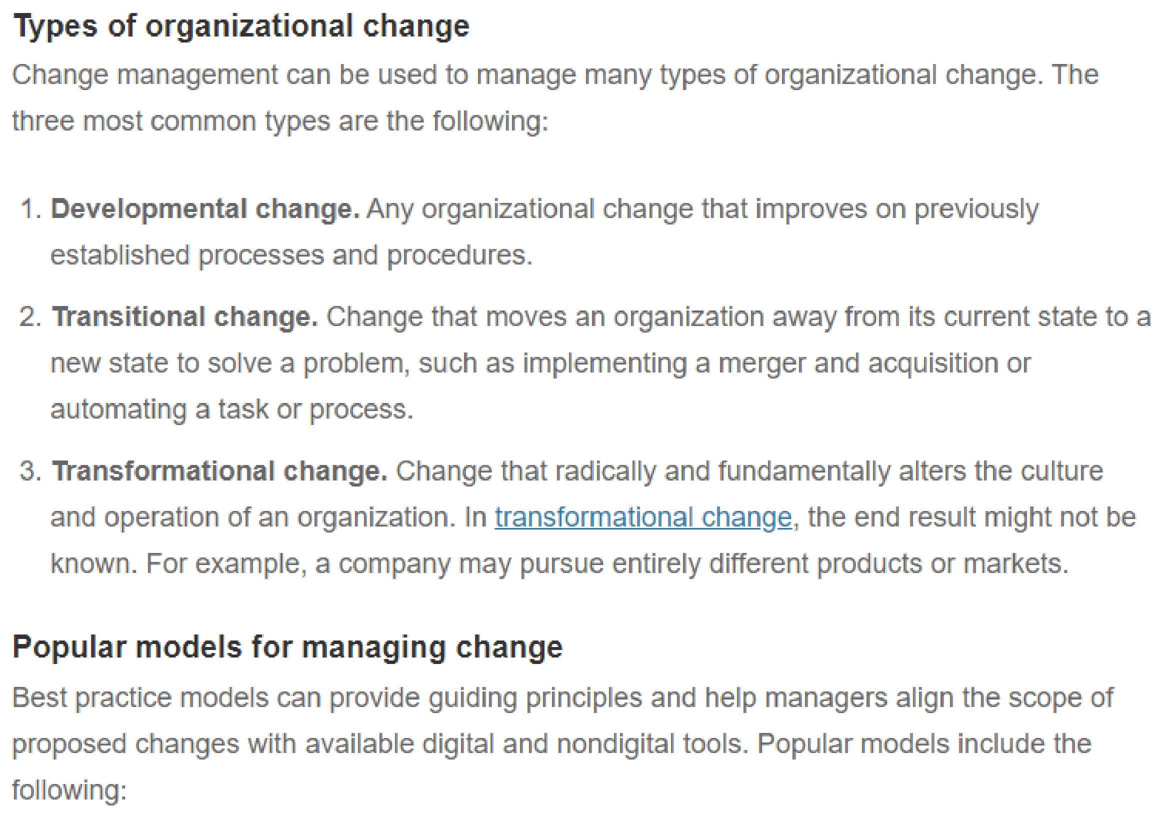

In software development project management, change management strategies and tools help developers manage changes to code and its associated documentation and enable chief information officers (CIOs) to keep projects on track. Agile software development environments encourage changes that are made to satisfy requirements and/or adjust the user interface. Change is not addressed in the middle of an iteration, however; changes are scheduled as *stories* or *features* for future iterations.

Version control software tools assist with documentation and prevent more than one person from making changes to code at the same time. Such tools have capabilities to track changes and back out changes when necessary.

Change management for IT infrastructure

Change management tools are also used to track changes made to an IT department's hardware infrastructure. As with other types of change management, standardized methods and procedures ensure every change made to the infrastructure is assessed, approved, documented, implemented and reviewed in a systematic manner.

Changes made to hardware settings are also referred to as *configuration management* (CM). Technicians use CM tools to review the entire collection of related systems and verify the effects that a change in one system has on other systems.


Types of organizational change

Change management can be used to manage many types of organizational change. The three most common types are the following:

1. **Developmental change.** Any organizational change that improves on previously established processes and procedures.
2. **Transitional change.** Change that moves an organization away from its current state to a new state to solve a problem, such as implementing a merger and acquisition or automating a task or process.
3. **Transformational change.** Change that radically and fundamentally alters the culture and operation of an organization. In transformational change, the end result might not be known. For example, a company may pursue entirely different products or markets.

Popular models for managing change

Best practice models can provide guiding principles and help managers align the scope of proposed changes with available digital and nondigital tools. Popular models include the following:

4. **Define: (i) Functional requirements and (ii) Non-functional requirements iii) Domain requirement**

Functional Requirements: These are the requirements that the end user specifically demands as basic facilities that the system should offer. All these functionalities need to be necessarily incorporated into the system as a part of the contract. These are represented or stated in the form of input to be given to the system, the operation performed and the output expected. They are basically the requirements stated by the user which one can see directly in the final product, unlike the non-functional requirements.
For example, in a hospital management system, a doctor should be able to retrieve the information of his patients. Each high-level functional requirement may involve several interactions or dialogues between the system and the outside world. In order to accurately describe the functional requirements, all scenarios must be enumerated.

There are many ways of expressing functional requirements e.g., natural language, a structured or formatted language with no rigorous syntax and formal specification language with proper syntax.

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

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

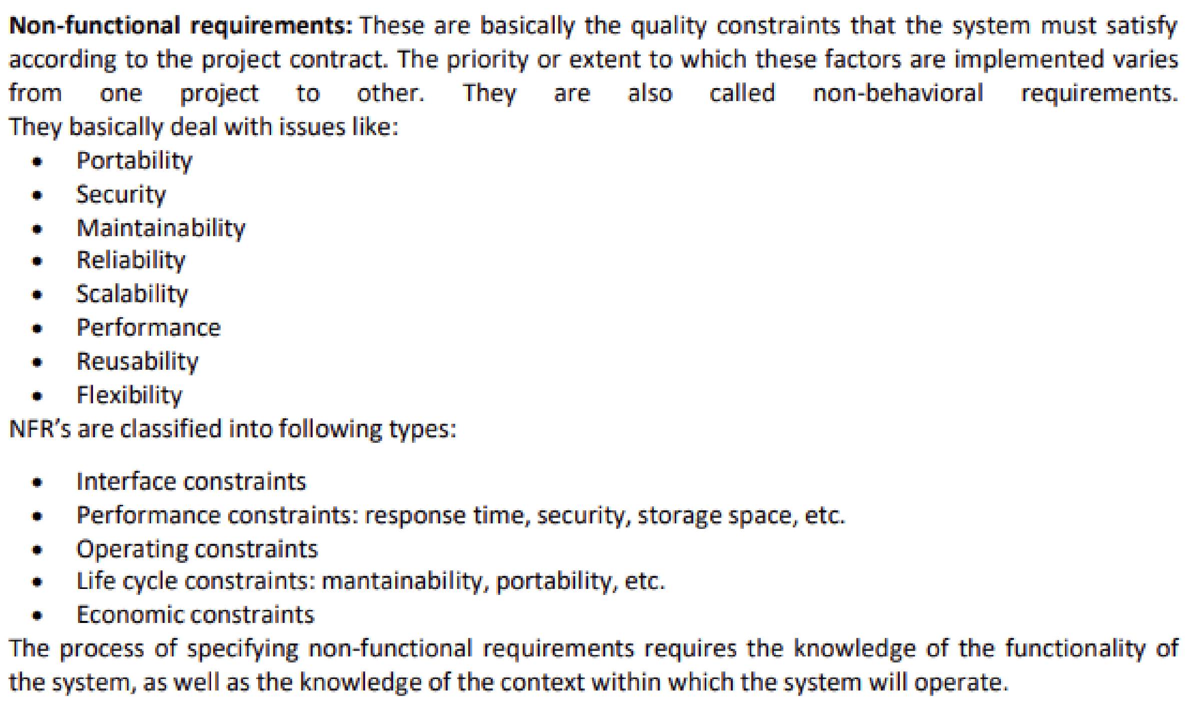
NFR's are classified into following types:

- Interface constraints
- Performance constraints: response time, security, storage space, etc.
- Operating constraints
- Life cycle constraints: mantainability, portability, etc.
- Economic constraints

The process of specifying non-functional requirements requires the knowledge of the functionality of the system, as well as the knowledge of the context within which the system will operate.

Domain requirements: Domain requirements are the requirements which are characteristic of a particular category or domain of projects. The basic functions that a system of a specific domain must necessarily exhibit come under this category. For instance, in academic software that maintains records of a school or college, the functionality of being able to access the list of faculty and list of students of each grade is a domain requirement. These requirements are therefore identified from that domain model and are not user specific.

5. Explain the different metrics of non-functional requirements



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

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

NFR's are classified into following types:

- Interface constraints
- Performance constraints: response time, security, storage space, etc.
- Operating constraints
- Life cycle constraints: mantainability, portability, etc.
- Economic constraints

The process of specifying non-functional requirements requires the knowledge of the functionality of the system, as well as the knowledge of the context within which the system will operate.

Explain everything in detail

6 **Write Short Notes on: a) Requirement Specification b) View point, c) Scenario techniques, d) Ethnography e). Feasibility study**

a. A **viewpoint** is a way of organising the **requirements** for a **software** system, based on some perspective such as an end-user perspective or a manager's perspective. ... Interactor **viewpoints** that represent people or other systems that interact directly with the system.

b. **Stakeholder** A person, group, or organization that is actively involved in a project, is affected by its outcome, or can influence its outcome. The term **stakeholder** is used to refer to any person or group who will be affected by the system, directly or indirectly.

Project Stakeholders

- Top Management. Top management may include the president of the company, vice-presidents, directors, division managers, the corporate operating committee, and others. ...
- The Project Team. ...
- Your Manager. ...
- Peers. ...
- Resource Managers. ...
- Internal Customers. ...
- External customer. ...
- Government

c. **Scenario** planning, also called **scenario** thinking or **scenario** analysis is a strategic planning method that some organizations use to make flexible long-term plans. It is in large part an adaptation and generalization of classic methods used by military intelligence.

d. ***Ethnography:*** Ethnography is the systematic study of people and cultures. It is designed to explore cultural phenomena where the researcher observes society from the point of view of the subject of the study. Ethnography is a means to represent graphically and in writing the culture of a group.

e. ***Use-case*** In software and systems engineering, a use case is a list of actions or event steps typically defining the interactions between a role and a system to achieve a goal. The actor can be a human or other external system.

f. ***Sequence diagram*** A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario

g. ***Feasibility study:*** Feasibility Study is an assessment of the practicality of a proposed project or system.

h. ***Volatile requirements:*** Stable **requirements** are concerned with the essence of a system and its application domain. They change more slowly than **volatile requirements**. **Volatile requirements** are specific to the instantiation of the system in a particular environment and for a particular customer.

i. ***Require change management:*** Requirements management is the process of documenting, analyzing, tracing, prioritizing and agreeing on requirements and then controlling change and communicating to relevant stakeholders. It is a continuous process throughout a project. A requirement is a capability to which a project outcome should conform.

7 What are the various techniques used for requirements elicitationand analysis? Explain any one technique in detail.

Requirements Elicitation & Analysis

It's a process of interacting with customers and end-users to find out about the domain requirements, what services the system should provide, and the other constrains.

Domain requirements reflect the environment in which the system operates so, when we talk about an application domain we mean environments such as train operation, medical records, e-commerce etc.

It may also involve a different kinds of stockholders; end-users, managers, system engineers, test engineers, maintenance engineers, etc.

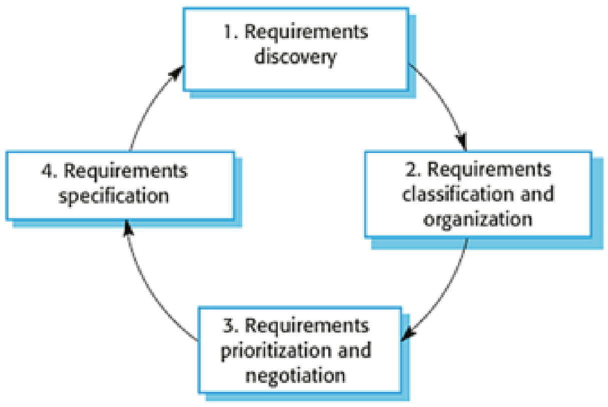
A stakeholder is anyone who has direct or indirect influence on the requirements.

The requirements elicitation and analysis has 4 main process

We typically start by gathering the requirements, this could be done through a general discussion or interviews with your stakeholders, also it may involve some graphical notation.

Then you organize the related requirements into sub components and prioritize them, and finally, you refine them by removing any ambiguous requirements that may raise from some conflicts.

Here are the 4 main process of requirements elicitation and analysis.


1. Requirements discovery
2. Requirements classification and organization
3. Requirements prioritization and negotiation
4. Requirements specification

The process of requirements elicitation and analysis

It shows that it's an iterative process with a feedback from each activity to another. The process cycle starts with requirements discovery and ends with the requirements document. The cycle ends when the requirements document is complete.

1. Requirements Discovery

It's the process of interacting with, and gathering the requirements from, the stakeholders about the required system and the existing system (if exist).

It can be done using some techniques, like interviews, scenarios, prototypes, etc, which help the stockholders to understand what the system will be like.

Gathering and understanding the requirements is a difficult process

That's because stakeholders may not know what exactly they want the software to do, or they may give un-realistic requirements.

They may give different requirements, which will result in conflict between the requirements, so we have to discover and resolve these conflicts.
Also there might be some factors that influence on the stakeholder's decision, like for example, managers at a company or professors at the university want to take a full control over the management system.

Interviews

In Interviews, requirements engineering teams put the questions to the stakeholder about the system that's currently used, and the system to be developed, and hence they can gather the requirements from the answers.
The questions fall under two categories:

1. *Closed-Ended* questions: A pre-defined set of question.
2. *Open-Ended* questions: There is no a pre-defined expected answer, they are more of generic questions. It's used to explore issues that's not clear in a less structured way.

In practice, interviews usually use mixture of both. Usually, start with the open-ended questions, and then use the closed-ended questions to be more specific about some requirements that aren't clear yet.
Interviews are good to get an overall understanding of what stakeholders need, how they might interact with the new system, and the difficulties they face with the current system.

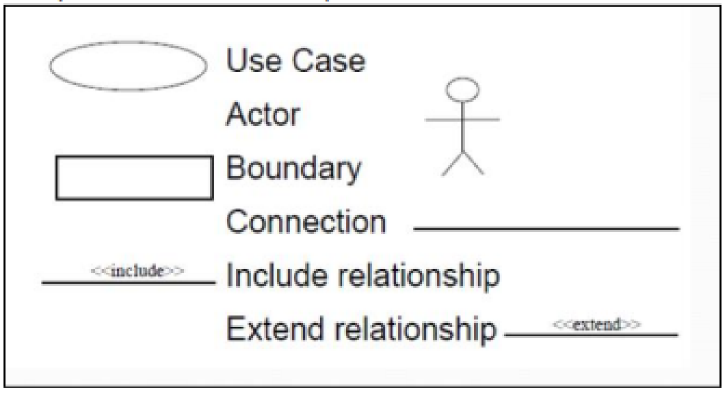
However, interviews aren't so helpful in understanding the domain requirements. This is for two reasons:

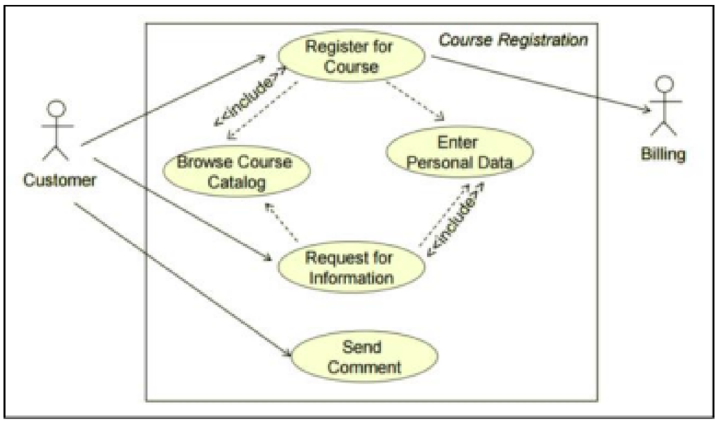
1. Domain requirements may be expressed using special domain terminologies, and software engineers often find it difficult to understand and it's easy for them to misunderstand.

2. Sometimes stakeholders won't tell you some requirements because they assume it's so fundamental and it doesn't worth mentioning, or they find it difficult to explain, which won't be taken into consideration in the requirements.

Use Cases & Scenarios

The use cases and scenarios are two different techniques, but, usually they are used together.
Use cases identify interactions between the system and it's users or even other external systems (using graphical notations), while a scenario is a textual description of one or more of these interactions.
Use case involves some symbols to describe the system:





Use case diagram symbols and an example

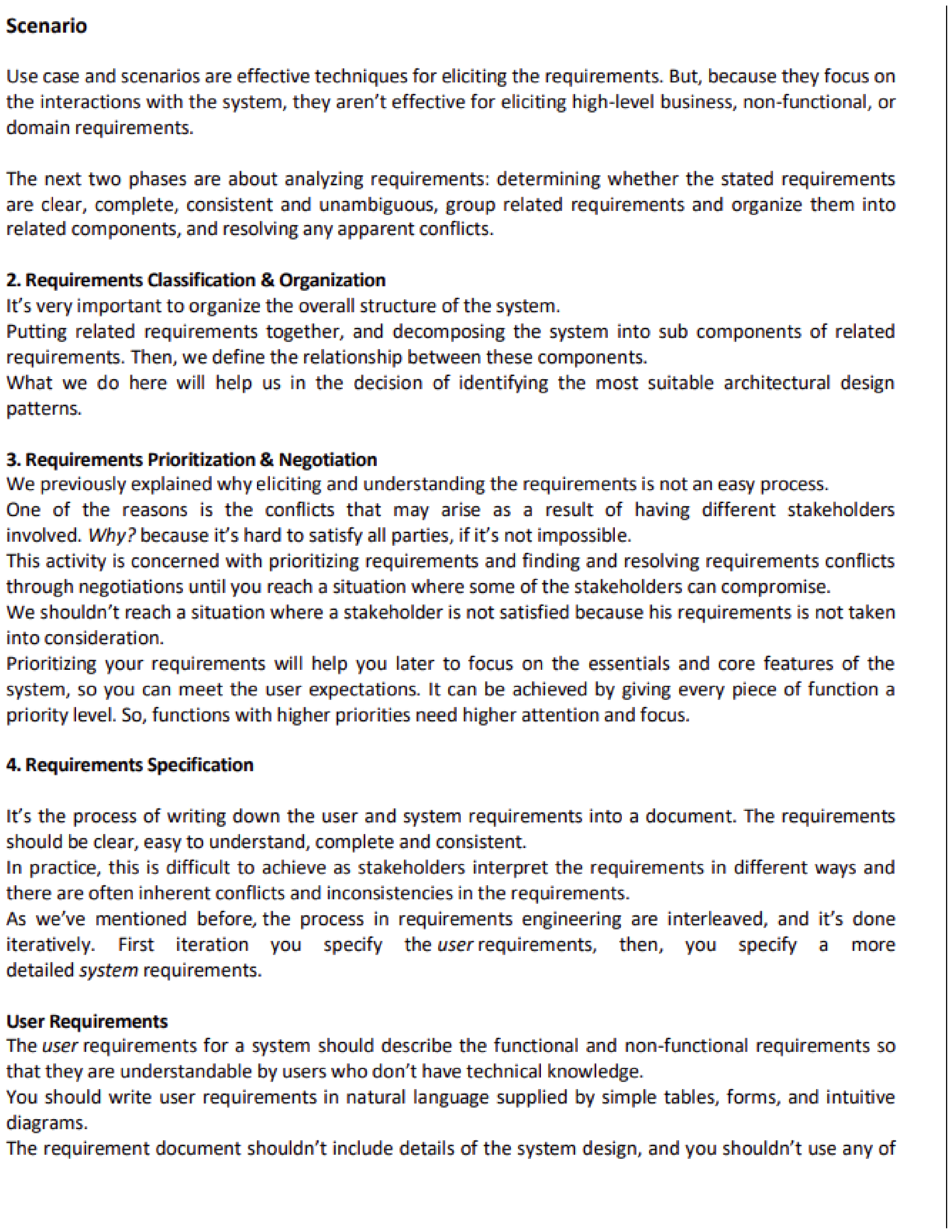
1. **Actors**: Are those who interact with the system; human or other systems
2. **Interaction (Use Case)**: It denotes the name of the interaction (verb). It's represented as a named ellipse.
3. **Connection**: Lines that links between the actors and the interactions.
4. **Include Relationship**: It denotes a connection between two interactions when an interaction is invoked by another. As an example, splitting a large interaction into several interactions.
5. **Exclude Relationship**: It denotes a connection between two interactions when you want to extend an interaction by adding an optional behavior, but you can use the main interaction on it's own without the extending interaction.

Now, we are going to use scenarios to describe the interactions in each use case textually. They should have a format and include the following:

1. A description of the initial situation.
2. A description of the flow of the events or interactions with the system
3. A description of the exceptions, or in other words, what can go wrong, and how it can be handled.
4. Any concurrent activities should be mentioned
5. A description of the final state.

Here is the example for a scenario for the use case example above.

Name	Course Registration
Actors	Student and University System
Description	It shows how a student can register for a course and view personal info
Pre-condition	The student is logged in
Post-condition	The student registered his/her course list for the semester.
Actions(Main Scenario)	1. Student will press on "Course Registration" from Home Page. 2. Select desired courses for the next semester. 3. Enter personal info. 4. Press on "Register". 5. Confirmation message upon success. 6. Student can view his/her personal info.
Exceptions	#3 User entered invalid input, thus, an error message will be displayed



Scenario

Use case and scenarios are effective techniques for eliciting the requirements. But, because they focus on the interactions with the system, they aren't effective for eliciting high-level business, non-functional, or domain requirements.

The next two phases are about analyzing requirements: determining whether the stated requirements are clear, complete, consistent and unambiguous, group related requirements and organize them into related components, and resolving any apparent conflicts.

2. Requirements Classification & Organization
It's very important to organize the overall structure of the system.
Putting related requirements together, and decomposing the system into sub components of related requirements. Then, we define the relationship between these components.
What we do here will help us in the decision of identifying the most suitable architectural design patterns.

3. Requirements Prioritization & Negotiation
We previously explained why eliciting and understanding the requirements is not an easy process.
One of the reasons is the conflicts that may arise as a result of having different stakeholders involved. *Why?* because it's hard to satisfy all parties, if it's not impossible.
This activity is concerned with prioritizing requirements and finding and resolving requirements conflicts through negotiations until you reach a situation where some of the stakeholders can compromise.
We shouldn't reach a situation where a stakeholder is not satisfied because his requirements is not taken into consideration.
Prioritizing your requirements will help you later to focus on the essentials and core features of the system, so you can meet the user expectations. It can be achieved by giving every piece of function a priority level. So, functions with higher priorities need higher attention and focus.

4. Requirements Specification

It's the process of writing down the user and system requirements into a document. The requirements should be clear, easy to understand, complete and consistent.
In practice, this is difficult to achieve as stakeholders interpret the requirements in different ways and there are often inherent conflicts and inconsistencies in the requirements.
As we've mentioned before, the process in requirements engineering are interleaved, and it's done iteratively. First iteration you specify the *user* requirements, then, you specify a more detailed *system* requirements.

User Requirements
The *user* requirements for a system should describe the functional and non-functional requirements so that they are understandable by users who don't have technical knowledge.
You should write user requirements in natural language supplied by simple tables, forms, and intuitive diagrams.
The requirement document shouldn't include details of the system design, and you shouldn't use any of

8. Define Object Orientation. Explain the three kinds of models used in Object Oriented Design method

In the object-oriented approach, the focus is on capturing the structure and behavior of information systems into small modules that combines both data and process. The main aim of Object Oriented Design (OOD) is to improve the quality and productivity of system analysis and design by making it more usable.

In analysis phase, OO models are used to fill the gap between problem and solution. It performs well in situation where systems are undergoing continuous design, adaption, and maintenance. It identifies the objects in problem domain, classifying them in terms of data and behavior.

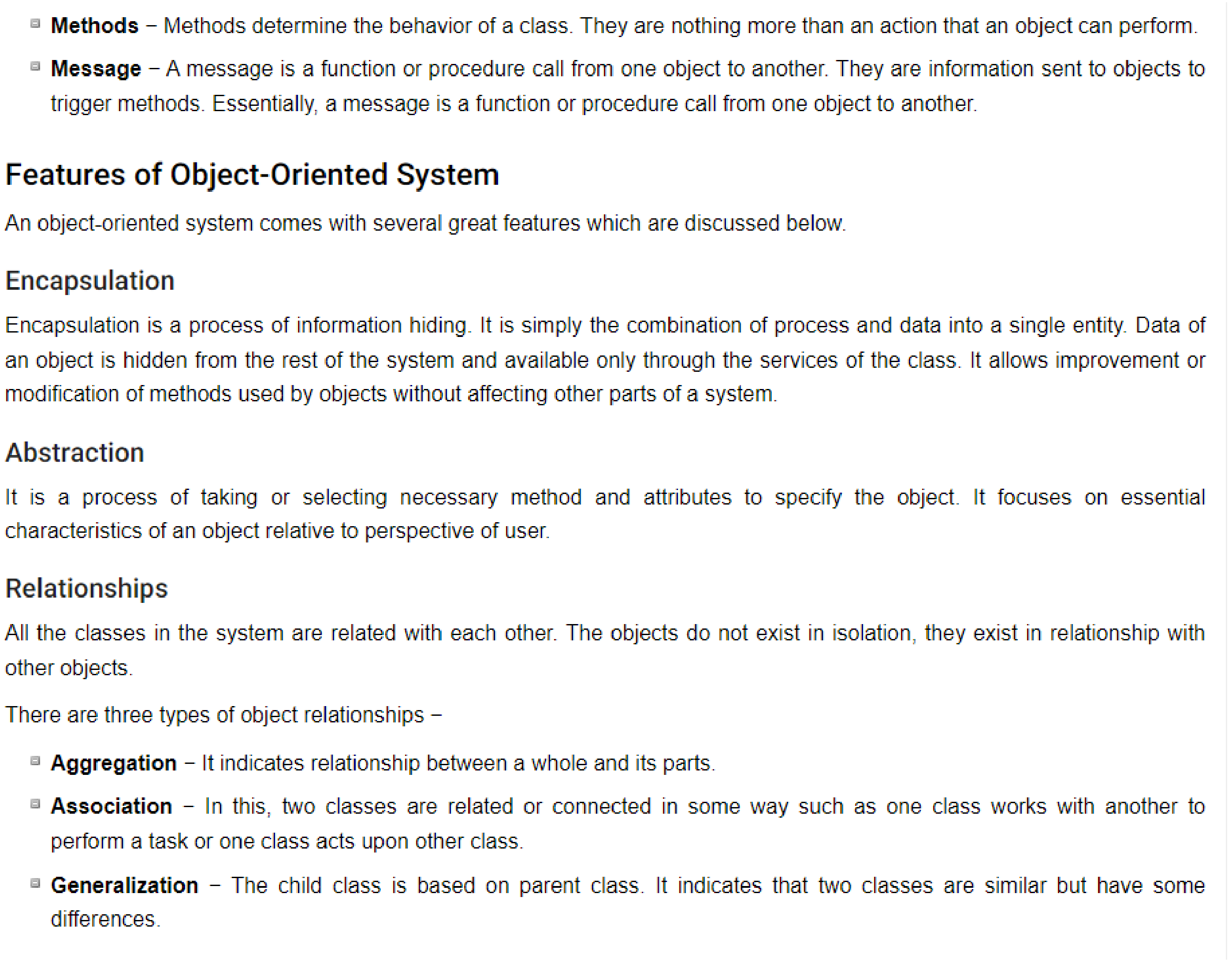
The OO model is beneficial in the following ways −

- It facilitates changes in the system at low cost.
- It promotes the reuse of components.
- It simplifies the problem of integrating components to configure large system.
- It simplifies the design of distributed systems.

Elements of Object-Oriented System

Let us go through the characteristics of OO System −

- **Objects** − An object is something that is exists within problem domain and can be identified by data (attribute) or behavior. All tangible entities (student, patient) and some intangible entities (bank account) are modeled as object.
- **Attributes** − They describe information about the object.
- **Behavior** − It specifies what the object can do. It defines the operation performed on objects.
- **Class** − A class encapsulates the data and its behavior. Objects with similar meaning and purpose grouped together as class.
- **Methods** − Methods determine the behavior of a class. They are nothing more than an action that an object can perform.



- **Methods** – Methods determine the behavior of a class. They are nothing more than an action that an object can perform.
- **Message** – A message is a function or procedure call from one object to another. They are information sent to objects to trigger methods. Essentially, a message is a function or procedure call from one object to another.

Features of Object-Oriented System

An object-oriented system comes with several great features which are discussed below.

Encapsulation

Encapsulation is a process of information hiding. It is simply the combination of process and data into a single entity. Data of an object is hidden from the rest of the system and available only through the services of the class. It allows improvement or modification of methods used by objects without affecting other parts of a system.

Abstraction

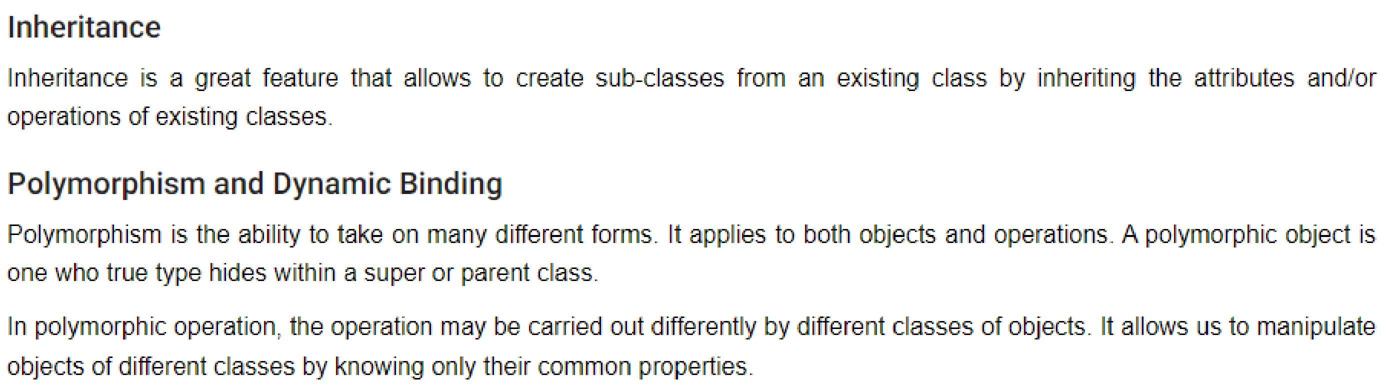
It is a process of taking or selecting necessary method and attributes to specify the object. It focuses on essential characteristics of an object relative to perspective of user.

Relationships

All the classes in the system are related with each other. The objects do not exist in isolation, they exist in relationship with other objects.

There are three types of object relationships –

- **Aggregation** – It indicates relationship between a whole and its parts.
- **Association** – In this, two classes are related or connected in some way such as one class works with another to perform a task or one class acts upon other class.
- **Generalization** – The child class is based on parent class. It indicates that two classes are similar but have some differences.



Inheritance

Inheritance is a great feature that allows to create sub-classes from an existing class by inheriting the attributes and/or operations of existing classes.

Polymorphism and Dynamic Binding

Polymorphism is the ability to take on many different forms. It applies to both objects and operations. A polymorphic object is one who true type hides within a super or parent class.

In polymorphic operation, the operation may be carried out differently by different classes of objects. It allows us to manipulate objects of different classes by knowing only their common properties.

9. What are the four aspects of object-oriented approach? Explain

The Four Principles of Object-Oriented-Programming (OOP):

Encapsulation

Encapsulation is accomplished when each object maintains a private state, inside a class. Other objects can not access this state directly, instead, they can only invoke a list of public functions. The object manages its own state via these functions and no other class can alter it unless explicitly allowed. In order to communicate with the object, you will need to utilize the methods provided. One way I like to think of encapsulation is by using the example of people and their dogs. If we want to apply encapsulation, we do so by encapsulating all "dog" logic into a Dog class. The "state" of the dog is in the private variables playful, hungry and energy and each of these variables has their respective fields.

There is also a private method: bark(). The dog class can call this whenever it wants, and the other classes can not tell the dog when to bark. There are also public methods such as sleep(), play() and eat() that are available to other classes. Each of these functions modifies the internal state of the Dog class and may invoke bark(), when this happens the private state and public methods are bonded.

Abstraction

Abstraction is an extension of encapsulation. It is the process of selecting data from a larger pool to show only the relevant details to the object. Suppose you want to create a dating application and you are asked to collect all the information about your users. You might receive the following data from your user: Full name, address, phone number, favorite food, favorite movie, hobbies, tax information, social security number, credit score. This amount of data is great however not all of it is required to create a dating profile. You only need to select the information that is pertinent to your dating application from that pool. Data like Full name, favorite food, favorite movie, and hobbies make sense for a dating application. The process of fetching/removing/selecting the user information from a larger pool is referred to as Abstraction. One of the advantages of Abstraction is being able to apply the same information you used for the dating application to other applications with little or no modification.

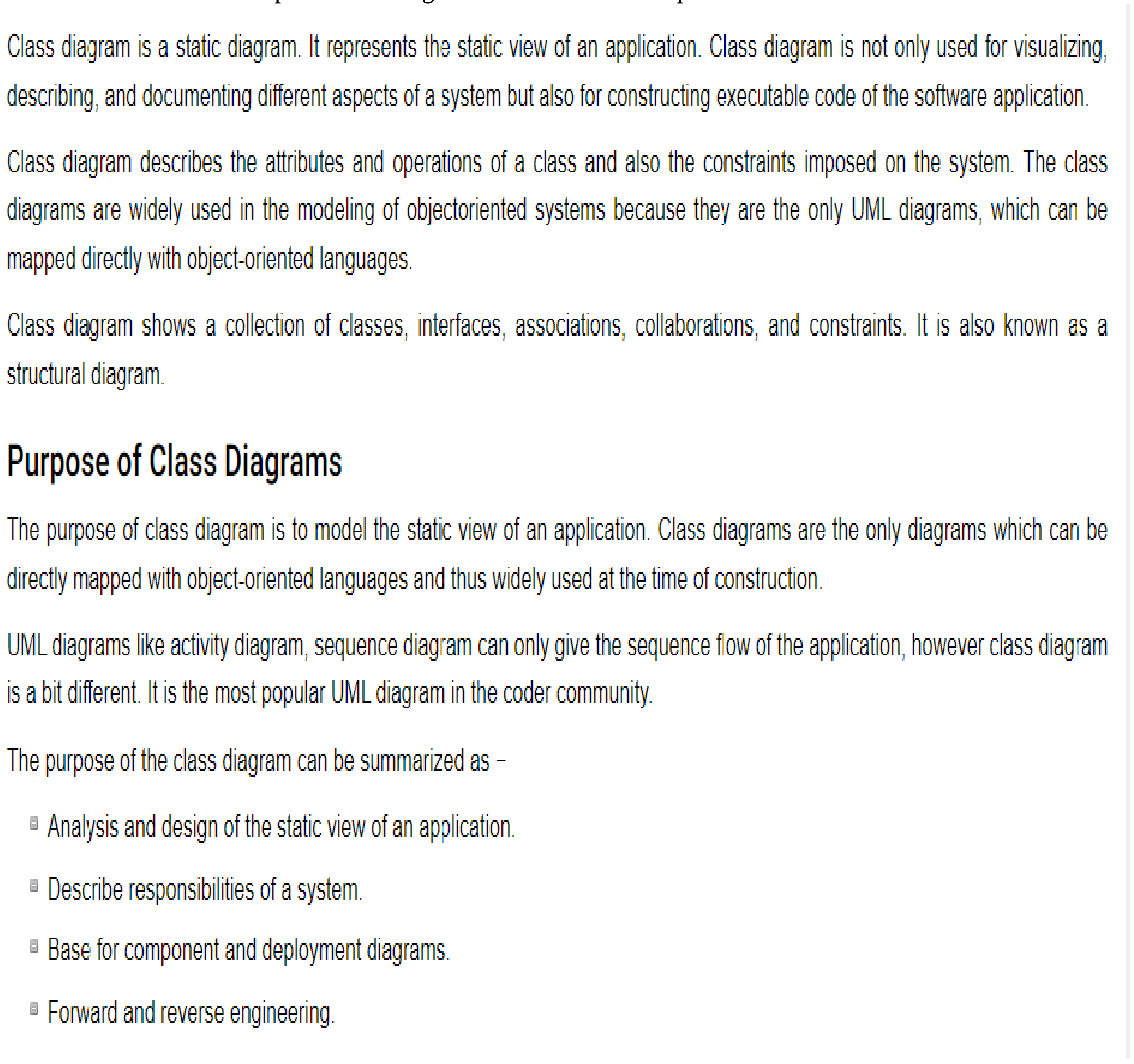
Inheritance

Inheritance is the ability of one object to acquire some/all properties of another object. For example, a child inherits the traits of his/her parents. With inheritance, reusability is a major advantage. You can reuse the fields and methods of the existing class. In Java, there are various types of inheritances: single, multiple, multilevel, hierarchical, and hybrid. For example, Apple is a fruit so assume that we have a class Fruit and a subclass of it named Apple. Our Apple acquires the properties of the Fruit class. Other classifications could be grape, pear, and mango, etc. Fruit defines a class of foods that are mature ovaries of a plant, fleshy, contains a large seed within or numerous tiny seeds. Apple the sub-class acquires these properties from Fruit and has some unique properties, which are different from other sub-classes of Fruit such as red, round, depression at the top.

Polymorphism

Polymorphism gives us a way to use a class exactly like its parent so there is no confusion with mixing types. This being said, each child sub-class keeps its own functions/methods as they are. If we had a superclass called Mammal that has a method called mammalSound(). The sub-classes of Mammals could be Dogs, whales, elephants, and horses. Each of these would have their own iteration of a mammal sound (dog-barks, whale-clicks).

10. What is a class and explain class diagram with suitable example.



Class diagram is a static diagram. It represents the static view of an application. Class diagram is not only used for visualizing, describing, and documenting different aspects of a system but also for constructing executable code of the software application.

Class diagram describes the attributes and operations of a class and also the constraints imposed on the system. The class diagrams are widely used in the modeling of objectoriented systems because they are the only UML diagrams, which can be mapped directly with object-oriented languages.

Class diagram shows a collection of classes, interfaces, associations, collaborations, and constraints. It is also known as a structural diagram.

Purpose of Class Diagrams

The purpose of class diagram is to model the static view of an application. Class diagrams are the only diagrams which can be directly mapped with object-oriented languages and thus widely used at the time of construction.

UML diagrams like activity diagram, sequence diagram can only give the sequence flow of the application, however class diagram is a bit different. It is the most popular UML diagram in the coder community.

The purpose of the class diagram can be summarized as −

- Analysis and design of the static view of an application.
- Describe responsibilities of a system.
- Base for component and deployment diagrams.
- Forward and reverse engineering.

How to Draw a Class Diagram?

Class diagrams are the most popular UML diagrams used for construction of software applications. It is very important to learn the drawing procedure of class diagram.

Class diagrams have a lot of properties to consider while drawing but here the diagram will be considered from a top level view.

Class diagram is basically a graphical representation of the static view of the system and represents different aspects of the application. A collection of class diagrams represent the whole system.

The following points should be remembered while drawing a class diagram −

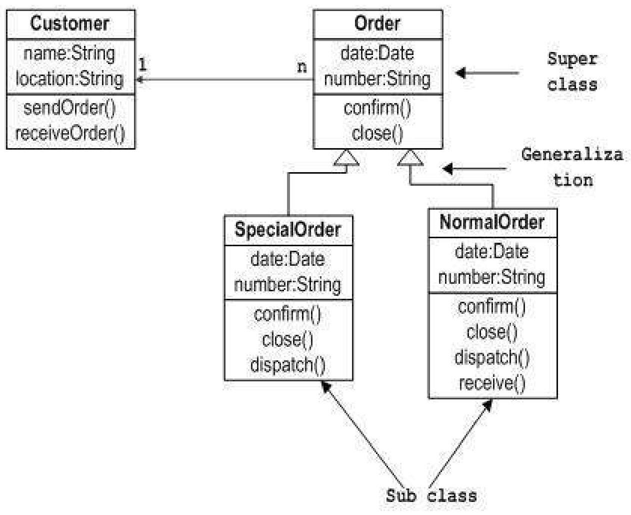
- The name of the class diagram should be meaningful to describe the aspect of the system.
- Each element and their relationships should be identified in advance.
- Responsibility (attributes and methods) of each class should be clearly identified
- For each class, minimum number of properties should be specified, as unnecessary properties will make the diagram complicated.
- Use notes whenever required to describe some aspect of the diagram. At the end of the drawing it should be understandable to the developer/coder.
- Finally, before making the final version, the diagram should be drawn on plain paper and reworked as many times as possible to make it correct.

The following diagram is an example of an Order System of an application. It describes a particular aspect of the entire application.

- First of all, Order and Customer are identified as the two elements of the system. They have a one-to-many relationship because a customer can have multiple orders.
- Order class is an abstract class and it has two concrete classes (inheritance relationship) SpecialOrder and NormalOrder.
- The two inherited classes have all the properties as the Order class. In addition, they have additional functions like dispatch () and receive ().

The following class diagram has been drawn considering all the points mentioned above.

Sample Class Diagram


Customer
name:String
location:String
sendOrder()
receiveOrder()
1
n
Order
date:Date
number:String
confirm()
close()
Super class
Generalization
SpecialOrder
date:Date
number:String
confirm()
close()
dispatch()
NormalOrder
date:Date
number:String
confirm()
close()
dispatch()
receive()
Sub class