OOSE MOD 3

PART-A

1)Define transform mapping? Explain the process with an illustration. What is its strength and weakness?

Transform mapping is a set of design steps that allows a DFD with transform flow characteristics to be mapped into a specific architectural style. In this section transform mapping is described by applying design steps to an example system—a portion of the SafeHome security software.

https://www.1000sourcecodes.com/2012/05/software-engineering-transform-mapping.html (Go through this link for the whole answer)

One **strength** of transform mapping is that it allows data to be transformed and mapped in a flexible and customizable way, allowing for a wide range of data formats and structures to be supported. It also allows for data to be transformed and mapped in real-time, as needed, which can be useful in dynamic or high-volume environments.

One **weakness** of transform mapping is that it can be time-consuming and resource-intensive to set up and maintain, particularly for large or complex data sets. It can also be challenging to ensure the accuracy and integrity of the transformed data, especially if there are errors or inconsistencies in the source data.

Another potential weakness is that transform mapping may not be suitable for all types of data or transformations. It may not be able to support certain data formats or structures, or it may not be able to handle certain types of data transformations, such as those that require more advanced data manipulation or processing.

2)Discuss about frequent item set? Write the Apriori algorithm for frequent item set generation? Explain with an example

A Frequent Itemset is **a subset(s) of an itemset that occurs in a dataset with a particular frequency**. For instance, given a frequency value, perhaps of 0.1 or 0.01%, for a stationery store, all subsets of school items that many customers have bought at different times are called Frequent Itemset.

Frequent Itemset is **an itemset whose support value is greater than a threshold value(support)**. Apriori algorithm uses frequent item sets to generate association rules. It is based on the concept that a subset of a frequent itemset must also be a frequent itemset.

Apriori algorithm(geeks for geeks) **OR**

Apriori algorithm(JavaTpoint)

3)Explain the examples of three data abstractions and the procedural abstractions that can be used to manipulate them(doubt)

In software engineering, data abstraction is the process of exposing only the essential characteristics of data, hiding the implementation details. Procedural abstraction is the process of exposing only the essential features of a procedure, hiding the implementation details.

Here are some examples of data abstractions and procedural abstractions that can be used to manipulate them:

1. **Stacks:** A stack is a data structure that allows elements to be added and removed only from the top. An example of a procedural abstraction for manipulating stacks is the push and pop operations, which allow elements to be added to and removed from the top of the stack, respectively.
2. **Queues:** A queue is a data structure that allows elements to be added only to the end and removed only from the front. An example of a procedural abstraction for

manipulating queues is the enqueue and dequeue operations, which allow elements to be added to the end and removed from the front of the queue, respectively.

3. **Linked lists:** A linked list is a data structure that consists of a series of nodes, each containing a value and a reference to the next node. An example of a procedural abstraction for manipulating linked lists is the insert and delete operations, which allow nodes to be added and removed from the list, respectively.

Overall, data abstractions and procedural abstractions are important concepts in software engineering, as they allow developers to work with complex data structures and algorithms in a simple and intuitive way. By exposing only the essential characteristics and features of data and procedures, abstractions make it easier to understand and modify complex systems.

4)Demonstrate the architecture of a house or building as a metaphor, Draw comparison with software architecture. How are the disciplines of classical architecture and software architecture similar? How do they differ

The architecture of a house or building can be thought of as a metaphor for software architecture. Both involve the design and planning of a structure to meet certain needs and requirements, and both involve the use of various disciplines and principles to achieve this.

One way to draw a comparison between classical architecture and software architecture is to consider the various components and their relationships. In classical architecture, the main components might include the foundation, walls, roof, windows, doors, and so on. These components are arranged and integrated in a specific way to form the overall structure of the house.

In software architecture, the main components might include modules, libraries, frameworks, and other software components. These components are also arranged and integrated in a specific way to form the overall structure of the system.

There are several ways in which classical architecture and software architecture are similar:

1. Both involve the use of a set of disciplines and principles to guide the design and planning process. For example, classical architecture may involve the use of principles such as balance, proportion, and symmetry, while software architecture may involve the use of principles such as modularity, encapsulation, and separation of concerns.
2. Both involve the use of diagrams and models to represent and communicate the design of the structure. For example, classical architecture may use floor plans, elevations, and sections, while software architecture may use class diagrams, component diagrams, and sequence diagrams.
3. Both involve the use of materials and construction techniques to build the structure. For example, classical architecture may involve the use of materials such as wood, concrete, and steel, while software architecture may involve the use of programming languages, libraries, and frameworks.

There are also some ways in which classical architecture and software architecture differ:

1. Classical architecture is concerned with the physical structure of the building, while software architecture is concerned with the logical structure of the software system.
2. Classical architecture involves the use of physical materials and construction techniques, while software architecture involves the use of logical components and design patterns.
3. Classical architecture is typically more constrained by physical limitations, such as the availability of materials and the laws of physics, while software architecture is typically more flexible and can be more easily modified and evolved over time.

While classical architecture and software architecture are similar in many ways, they also have some key differences. Both involve the use of disciplines and principles to guide the design and planning process, but software architecture is concerned with the logical structure of a software system, while classical architecture is concerned with the physical structure of a building.

5)Why are control components necessary in traditional software and generally not required in object-oriented software.

Control components are a type of software component that are responsible for controlling the flow of execution in a system. They are used to coordinate the activities of other components and ensure that they are executed in the correct sequence.

In traditional software, control components are often necessary to manage the complexity of the system and ensure that it functions correctly. This is because traditional software is often structured in a procedural or linear fashion, with a clear sequence of steps that must be followed.

Control components are generally not required in object-oriented software, as object-oriented software is structured in a more modular and flexible way. In object-oriented software, the individual components (i.e., objects) are responsible for their own behavior and can interact with each other through well-defined interfaces. This makes it easier to manage the complexity of the system and allows for more flexibility and reuse of the components.

Overall, control components are necessary in traditional software to manage the complexity and ensure the correct execution of the system, but they are generally not required in object-oriented software due to its more modular and flexible structure.

6)Explain the state oriented approaches for representing behavioral specifications of software.(doubt)

State-oriented approaches are a type of software engineering technique that are used to represent the behavioral specifications of a software system. These approaches are based on the idea that the behavior of a system can be modeled as a series of states, transitions, and events.

There are several ways that state-oriented approaches can be used to represent behavioral specifications:

1. **State machines:** A state machine is a model of a system's behavior that consists of a set of states and transitions between them. The system's behavior is determined by the current state and the events that trigger transitions between states.
2. **Statecharts:** A statechart is a graphical representation of a state machine, using symbols such as circles, rectangles, and arrows to represent states, transitions, and events. Statecharts can be used to visualize and communicate the behavior of a system in a clear and concise way.
3. **State diagrams:** A state diagram is a type of diagram that shows the possible states of a system and the transitions between them. State diagrams can be used to represent the behavior of a system in a more abstract way, and can be useful for understanding the overall behavior of the system.

State-oriented approaches are useful for representing behavioral specifications because they provide a clear and structured way to represent the behavior of a system. They can help to ensure that the system's behavior is well-defined and easy to understand, and can serve as a basis for testing and debugging the system.

Overall, state-oriented approaches are an important tool for representing the behavioral specifications of software, and can be used to ensure that the system is reliable, flexible, and easy to understand and modify

7)Construct dynamic model diagram, comprising of state transition diagrams(doubt)

To construct a dynamic model diagram comprising of state transition diagrams, you will need to follow these steps:

1. Identify the states that the system can be in. A state is a unique condition or situation that the system can be in at a given time.
2. Identify the transitions that can occur between states. A transition is a change in state that occurs when certain conditions are met.
3. Determine the triggers that cause transitions to occur. Triggers are events or conditions that initiate a transition.
4. Create a state transition diagram to represent the dynamic behavior of the system. A state transition diagram is a graphical representation of the states and transitions in a system. It

consists of a series of circles or rectangles, each representing a state, and arrows connecting the states to represent transitions.
5. Label the states and transitions with descriptive names or symbols to make the diagram more meaningful.
6. Add additional information to the diagram as needed, such as conditions for transitions, actions that occur during transitions, or time delays.
7. Review and validate the diagram to ensure that it accurately represents the dynamic behavior of the system.

8)Explain in detail about the characteristics and criteria for a good design.

Characteristics for a good design:

For good quality software to be produced, the software design must also be of good quality. Now, the matter of concern is how the quality of good software design is measured? This is done by observing certain factors in software design. These factors are:

1. Correctness
2. Understandability
3. Efficiency
4. Maintainability

1) Correctness

First of all, the design of any software is evaluated for its correctness. The evaluators check the software for every kind of input and action and observe the results that the software will produce according to the proposed design. If the results are correct for every input, the design is accepted and is considered that the software produced according to this design will function correctly.

2) Understandability

The software design should be understandable so that the developers do not find any difficulty to understand it. Good software design should be self- explanatory. This is because there are hundreds and thousands of developers that develop different modules of the software, and it would be very time consuming to explain each design to each developer. So, if the design is easy and self- explanatory, it would be easy for the developers to implement it and build the same software that is represented in the design.

3) Efficiency

The software design must be efficient. The efficiency of the software can be estimated from the design phase itself, because if the design is describing software that is not efficient and useful, then the developed software would also stand on the same level of efficiency. Hence, for efficient and good quality software to be developed, care must be taken in the designing phase itself.

4) Maintainability

The software design must be in such a way that modifications can be easily made in it. This is because every software needs time to time modifications and maintenance. So, the design of the software must also be able to bear such changes. It should not be the case that after making some modifications the other features of the software start misbehaving. Any change made in the software design must not affect the other available features, and if the features are getting affected, then they must be handled properly.

Criteria for a good design:(doubt)

In software engineering, some of the criteria for a good design include:

1. **Modularity:** The design should be modular, with clearly defined and separated components that can be easily maintained and modified independently.
2. **Reusability:** The design should be reusable, with components that can be easily adapted and integrated into other systems or projects.

3. **Extensibility**: The design should be extensible, with the ability to add new features or functionality without requiring significant redesign or rework.
4. **Scalability:** The design should be scalable, able to handle increased workloads or user demands without a decrease in performance.
5. **Portability:** The design should be portable, able to be easily adapted and run on different platforms or environments.
6. **Maintainability:** The design should be maintainable, with a clear and logical structure that makes it easy to understand and modify as needed.
7. **Testability:** The design should be testable, with a clear separation of concerns and well-defined interfaces that make it easy to create and run automated tests.
8. **Security:** The design should be secure, with measures in place to protect against external threats and vulnerabilities.
9. **Performance:** The design should be efficient and have good performance, with fast response times and minimal resource usage.
10. **User experience:** The design should be user-friendly and provide a positive and intuitive experience for users.

9)Draw the basic structure of analysis model and explain each entity in detail.(doubt)

Analysis modeling

- Analysis model operates as a link between the 'system description' and the 'design model'
- In the analysis model, information, functions and the behaviour of the system is defined and these are translated into the architecture, interface and component level design in the 'design modeling'

Elements of the analysis model

1. Scenario based element

- This type of element represents the system user point of view
- Scenario based elements are use case diagram, user stories

2. Class based elements

- The object of this type of element manipulated by the system
- It defines the object,attributes and relationship
- The collaboration is occurring between the classes
- Class based elements are the class diagram, collaboration diagram

3. Behavioral elements

- Behavioral elements represent state of the system and how it is changed by the external events
- The behavioral elements are sequenced diagram, state diagram

4. Flow oriented elements

- An information flows through a computer-based system it gets transformed
- It shows how the data objects are transformed while they flow between the various system functions
- The flow elements are data flow diagram, control flow diagram

Analysis Rules of Thumb

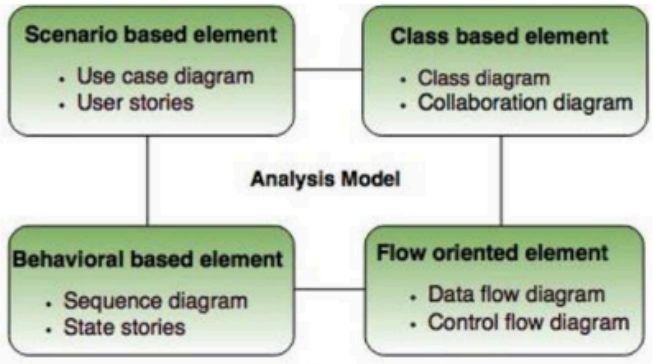


Fig. - Elements of analysis model

Figure 3.1: elements of analysis model

The rules of thumb that must be followed while creating the analysis model

The rules are as follows

- The model focuses on the requirements in the business domain. The level of abstraction

must be high i.e there is no need to give details

- Every element in the model helps in understanding the software requirement and focus on the information, function and behaviour of the system
- The consideration of infrastructure and nonfunctional model delayed in the design

For example, the database is required for a system, but the classes, functions and behavior of the database are not initially required. If these are initially considered then there is a delay in the designing

- Throughout the system minimum coupling is required. The interconnections between the modules is known as 'coupling'
- The analysis model gives value to all the people related to model
- The model should be simple as possible. Because simple model always helps in easy understanding of the requirement

Concepts of data modeling

- Analysis modeling starts with the data modeling
- The software engineer defines all the data object that proceeds within the system and the relationship between data objects are identified. Data objects
- The data object is the representation of composite information
- The composite information means an object has a number of different properties or attribute

For example, Height is a single value so it is not a valid data object, but dimensions contain the height, the width and depth these are defined as an object. Data Attributes

Each of the data object has a set of attributes

Data object has the following characteristics

- Name an instance of the data object
- Describe the instance
- Make reference to another instance in another table ,Relationship

Relationship shows the relationship between data objects and how they are related to each other

10)Describe various prototyping techniques and discuss on analysis sand modeling

[Prototyping techniques]

Sand modeling?

PART-B

1)Write in detail about Object Model and its relationship

The object model is a way of organizing and structuring the data and functionality of a software system. It represents the components of a system, their properties and behaviors, and the relationships between them.

In software engineering, the object model is used to design and implement software systems. It helps developers to understand the structure of a system and how its various components interact with each other. The object model also serves as a blueprint for the implementation of the system, providing a clear and concise representation of its structure and functionality.

The object model is closely related to the concept of object-oriented programming (OOP). OOP is a programming paradigm that focuses on the use of objects and their interactions to design and implement software systems. In OOP, objects are self-contained units that represent real-world entities and encapsulate their properties and behaviors. The object model represents the structure and relationships of these objects in a system.

The object model is an important tool in software engineering because it helps developers to organize and structure the data and functionality of a system in a logical

and intuitive way. It also enables developers to reuse code and design more flexible and maintainable systems.

2)Explain about Design concepts for Modular Design

3)Explain the steps in effective Modular Design

For part B qn 2 and 3:

modular design

– Software is divided into separately named and addressable components, called modules that are integrated to satisfy problem requirements

– Design has to be modularized so that development can be more easily planned, software increments can be defined and delivered, changes can be more easily accommodated, testing and debugging can be conducted more efficiently and long- term maintenance can be conducted without serious side effects

Information Hiding

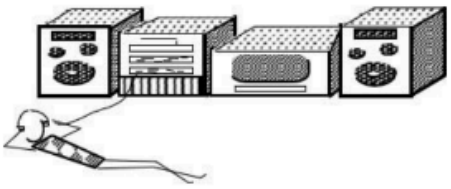


Figure 4.3: modular

- Reduces the likelihood of side effects
- Limits the global impact of local design decisions
- Emphasizes communication through controlled interfaces

- Discourages the use of global data
- Leads to encapsulation—an attribute of high quality design
- Results in higher quality software

Functional Independence

– Functional independence is achieved by developing modules with single minded‘ function and avoids excessive interaction with other modules

– Independent modules are easier to maintain because secondary effects caused by design

or code modification are limited, error propagation is reduced Independence is assessed using two qualitative criteria

- Cohesion which is an indication of the relative functional strength of a module. A cohesive module performs a single task, requiring little interaction with other components in other parts of a program
- Coupling is an indication of the relative interdependence among modules. Coupling depends on the interface complexity between modules, the point at which entry or reference is made to a module, and what data pass across the interface

Refinement

– Refinement is a process of elaboration. Refinement causes the designer to elaborate on the original statement, providing more and more detail as each successive refinement occurs

– Refinement helps the designer to reveal low-level details as design progresses

Refactoring

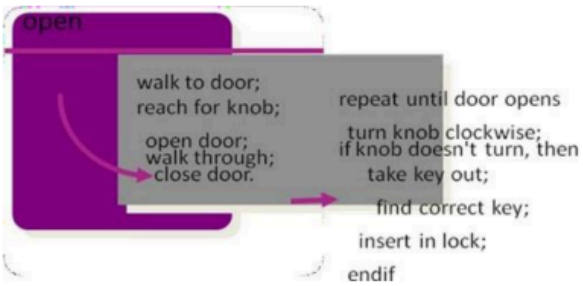


Figure 4.4: refinement

– Refactoring is the process of changing a software system in such a way that it does not alter the external behaviour of the code yet improves its internal structure

– When software is refactored, the existing design is examined for redundancy, unused design elements, inefficient or unnecessary algorithms, poorly constructed or inappropriate data structures, or any other design failure that can be corrected to yield a better design

Aspects

- As requirements analysis occurs, a set of concerns is uncovered. These concerns

include requirements, use cases, features, data structures, quality-of-service issues, variants, intellectual property boundaries, collaborations, patterns and contracts

- Ideally, a requirements model can be organized in a way that allows you to isolate each concern (requirement) so that it can be considered independently
- In practice, however, some of these concerns span the entire system and cannot be easily compartmentalized. As design begins, requirements are refined into a modular design representation
- Consider two requirements, A and B. Requirement A crosscuts requirement B if a software decomposition [refinement] has been chosen in which B cannot be satisfied without taking A into account

Refactoring

- An important design activity suggested for many agile methods, refactoring is a reorganization technique that simplifies the design (or code) of a component without changing its function or behavior
- When software is refactored, the existing design is examined for redundancy, unused design elements, inefficient or unnecessary algorithms, poorly constructed or inappropriate data structures, or any other design failure that can be corrected to yield a better design

4)Explain in detail about Flow-oriented Modeling and Behavioral Modeling

Flow-oriented Modeling

Flow-oriented modeling is a software design technique that involves representing the flow of data and control through a system as a series of interconnected nodes. Each node represents a discrete step in the process, and the flow of data and control is represented by the connections between the nodes.

Flow-oriented modeling is often used to design and implement systems that involve a high degree of data processing or transformation, such as data pipelines, ETL (extract, transform, load) systems, or data integration systems. It is particularly well-suited to systems that involve a high degree of parallelism or concurrency, as it allows developers to easily visualize and understand the flow of data and control through the system.

There are several advantages to using flow-oriented modeling in software engineering:

1. It provides a clear and intuitive representation of the flow of data and control through a system.
2. It makes it easy to identify and model parallelism and concurrency in a system.
3. It allows developers to easily visualize and understand the dependencies between different parts of a system.
4. It enables developers to identify and optimize bottlenecks and other performance issues in a system.
5. It allows for the easy modification and evolution of a system, as changes can be made by modifying individual nodes rather than the entire system.

Overall, flow-oriented modeling is a powerful tool for designing and implementing complex systems that involve the processing or transformation of data.

Behavioral Modeling

Behavioral modeling is a software design technique that involves representing the behavior of a system as a series of interactions between objects or components. It is

used to design and implement systems that involve complex or dynamic behavior, such as real-time systems, event-driven systems, or systems with a high degree of concurrency.

In behavioral modeling, the behavior of a system is represented as a set of interactions between objects or components, which are represented as nodes in a model. The interactions between objects are represented as edges between the nodes, and the flow of control through the system is represented by the sequence of interactions.

There are several advantages to using behavioral modeling in software engineering:

1. It provides a clear and intuitive representation of the behavior of a system.
2. It allows developers to easily understand and model complex or dynamic behavior.
3. It enables developers to identify and optimize performance issues in a system.
4. It allows for the easy modification and evolution of a system, as changes can be made by modifying the interactions between objects or components.
5. It helps to ensure that the behavior of a system is consistent with the requirements and design of the system.

Overall, behavioral modeling is a powerful tool for designing and implementing complex systems with dynamic or concurrent behavior.

5)Explain in detail about Design Concepts and Principles

Software Design Concepts:

The design concepts provide the software designer with a foundation from which more sophisticated methods can be applied. A set of fundamental design concepts has evolved. They are:

1)Abstraction - Abstraction is the process or result of generalization by reducing the information content of a concept or an observable phenomenon, typically in order to retain only information which is relevant for a particular purpose.

2)Refinement - It is the process of elaboration. A hierarchy is developed by decomposing a macroscopic statement of function in a step-wise fashion until programming language statements are reached. In each step, one or several instructions of a given program are decomposed into more detailed instructions. Abstraction and Refinement are complementary concepts.

3)Modularity - Software architecture is divided into components called modules.

4)Software Architecture - It refers to the overall structure of the software and the ways in which that structure provides conceptual integrity for a system. A good software architecture will yield a good return on investment with respect to the desired outcome of the project, e.g. in terms of performance, quality, schedule and cost.

5)Control Hierarchy - A program structure that represents the organization of a program component and implies a hierarchy of control.

6)Structural Partitioning -The program structure can be divided both horizontally and vertically. Horizontal partitions define separate branches of modular hierarchy for each major program function. Vertical partitioning suggests that control and work should be distributed top down in the program structure.

7)Data Structure-It is a representation of the logical relationship among individual elements of data.

8)Software Procedure - It focuses on the processing of each modules individually.

9)Information Hiding - Modules should be specified and designed so that information contained within a module is inaccessible to other modules that have no need for such information.

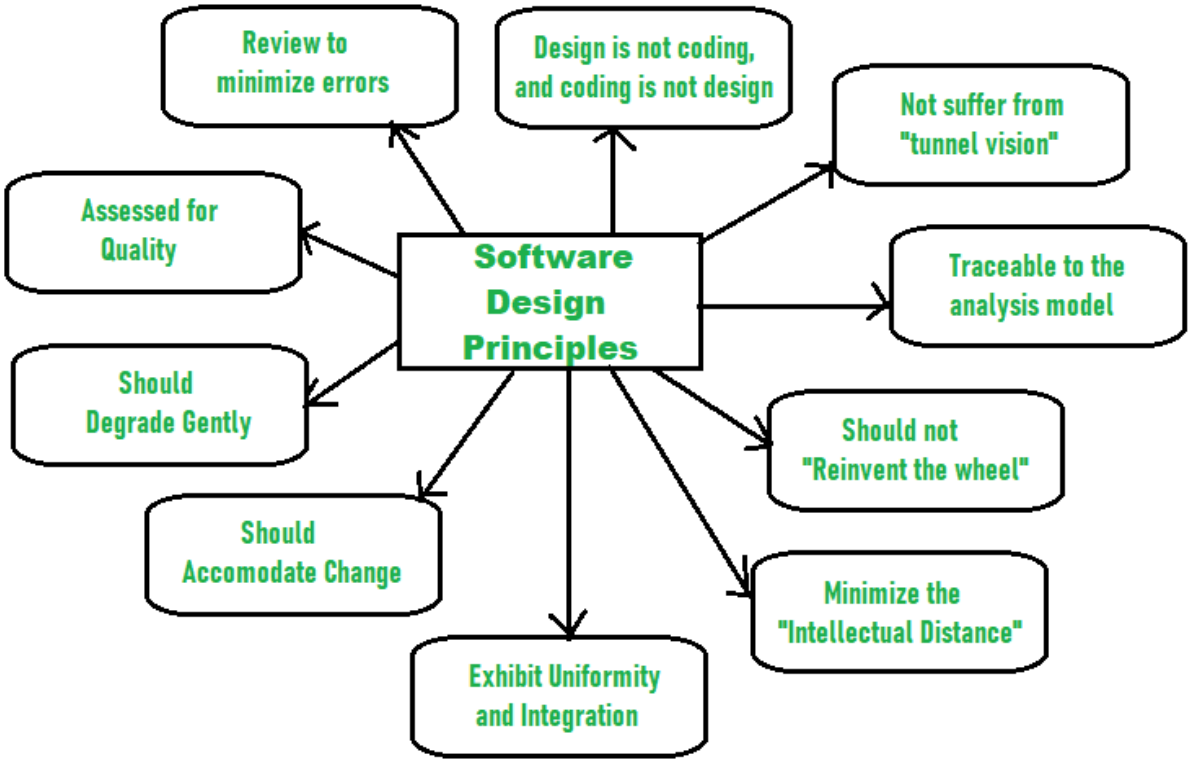
Design concepts taken from here

Principles Of Software Design :

1. **Should not suffer from “Tunnel Vision” –**
   While designing the process, it should not suffer from “tunnel vision” which means that is should not only focus on completing or achieving the aim but on other effects also.
2. **Traceable to analysis model –**
   The design process should be traceable to the analysis model which means it should satisfy all the requirements that software requires to develop a high-quality product.
3. **Should not “Reinvent The Wheel” –**
   The design process should not reinvent the wheel that means it should not waste time or effort in creating things that already exist. Due to this, the overall development will get increased.
4. **Minimize Intellectual distance –**
   The design process should reduce the gap between real-world problems and software solutions for that problem meaning it should simply minimize intellectual distance.
5. **Exhibit uniformity and integration –**
   The design should display uniformity which means it should be uniform throughout the process without any change. Integration means it should mix or combine all parts of software i.e. subsystems into one system.
6. **Accommodate change –**
   The software should be designed in such a way that it accommodates the change implying that the software should

adjust to the change that is required to be done as per the user's need.

7. **Degrade gently –**
   The software should be designed in such a way that it degrades gracefully which means it should work properly even if an error occurs during the execution.
8. **Assessed or quality –**
   The design should be assessed or evaluated for the quality meaning that during the evaluation, the quality of the design needs to be checked and focused on.
9. **Review to discover errors –**
   The design should be reviewed which means that the overall evaluation should be done to check if there is any error present or if it can be minimized.
10. **Design is not coding and coding is not design –**
    Design means describing the logic of the program to solve any problem and coding is a type of language that is used for the implementation of a design.



6)Explain the techniques in Domain Analysis

Refer here

7)Brief about Structured Analysis vs Object Oriented Analysis

Structured analysis is a traditional approach to software design that focuses on breaking a system down into smaller, well-defined units of functionality called modules. These modules are then connected and coordinated through a series of control and data flows. Structured analysis is based on the principles of modular design and structured programming, and is well-suited to systems with a high degree of procedural or algorithmic processing.

Object-oriented analysis (OOA) is a more modern approach to software design that is based on the principles of object-oriented programming (OOP). OOA involves modeling a system as a set of interacting objects, each of which represents a real-world entity or

concept and encapsulates its properties and behaviors. OOA is well-suited to systems with a high degree of complexity or reuse, and allows developers to easily model and manipulate complex data structures and relationships.

Difference Between Structured and Object-oriented analysis :

Structured Analysis	Object-Oriented Analysis
The main focus is on the process and procedures of the system.	The main focus is on data structure and real-world objects that are important.
It uses System Development Life Cycle (SDLC) methodology for different purposes like planning, analyzing, designing, implementing, and supporting an information system.	It uses Incremental or Iterative methodology to refine and extend our design.
It is suitable for well-defined projects with stable user requirements.	It is suitable for large projects with changing user requirements.
Risk while using this analysis technique is high and reusability is also low.	Risk while using this analysis technique is low and reusability is also high.

Structuring requirements include DFDs (Data Flow Diagram), Structured Analysis, ER (Entity Relationship) diagram, CFD (Control Flow Diagram), Data Dictionary, Decision table/tree, and the State transition diagram.	Requirement engineering includes the Use case model (find Use cases, Flow of events, Activity Diagram), the Object model (find Classes and class relations, Object interaction, Object to ER mapping), Statechart Diagram, and deployment diagram.
This technique is old and is not preferred usually.	This technique is new and is mostly preferred.

8)Explain in detail about Object Design Process(doubt)

Object design is the process of designing the structure and behavior of objects in an object-oriented software system. It involves identifying the objects that are needed to represent the entities and concepts in the system, and defining their properties and behaviors.

The object design process typically involves the following steps:

1. **Identify the objects in the system:** The first step in object design is to identify the objects that are needed to represent the entities and concepts in the system. This may involve creating a domain model to represent the relationships and concepts in the problem domain.
2. **Define the properties and behaviors of the objects:** Once the objects have been identified, the next step is to define their properties and behaviors. This may involve specifying the data that the objects need to store and the operations that they need to perform.
3. **Define the relationships between objects:** The objects in a system will often have relationships with each other, such as inheritance, aggregation, or association. It is important to identify and define these relationships in the object design.

4. **Define the interfaces of the objects:** The interfaces of the objects define how they can be used and interacted with by other parts of the system. It is important to define these interfaces carefully, as they will determine the flexibility and maintainability of the system.
5. **Implement the objects:** Once the object design is complete, the next step is to implement the objects according to the design. This may involve writing code to define the properties and behaviors of the objects and to implement the interfaces.

By following these steps, developers can create well-designed objects that are flexible, maintainable, and easy to understand and modify.

9)Explain in detail about Design Patterns

Design patterns are reusable solutions to common design problems in software engineering. They provide a way to structure and organize code in a way that is effective, flexible, and maintainable.

There are several types of design patterns, including creational, structural, and behavioral patterns. Creational patterns focus on the creation and initialization of objects, structural patterns focus on the composition of objects and classes, and behavioral patterns focus on the communication and interaction between objects.

Some examples of popular design patterns include:

1. **Singleton pattern:** This pattern ensures that a class has only one instance, and provides a global point of access to it.
2. **Factory pattern:** This pattern defines an interface for creating an object, but allows subclasses to alter the type of objects that will be created.
3. **Observer pattern:** This pattern defines a one-to-many dependency between objects, such that when one object changes state, all of its dependents are notified and updated automatically.
4. **Adapter pattern:** This pattern allows two incompatible interfaces to work together by adapting one interface to the other.

5. **Decorator pattern:** This pattern allows new behavior to be added to an existing object dynamically, by wrapping the object in a decorator object that adds the new behavior.

Design patterns are useful because they provide a common language and set of best practices for designing and structuring code. They can also help to improve the reuse and maintainability of code, as they provide proven solutions to common design problems.

10)Write about importance of data dictionary in classical analysis

A data dictionary is a collection of information about the data used in a software system. In classical analysis, a data dictionary is an important tool for documenting and organizing the data requirements of a system.

There are several reasons why a data dictionary is important in classical analysis:

1. It provides a central repository for information about the data used in a system, including the names, definitions, and relationships of the data elements.
2. It helps to ensure that the data used in a system is consistent and accurate, as it provides a single source of truth for the definitions and meanings of the data.
3. It helps to improve communication between members of the development team, as it provides a common understanding of the data used in the system.
4. It facilitates the maintenance and evolution of the system, as it provides a clear and concise record of the data requirements and relationships.

Overall, a data dictionary is an essential tool for classical analysis, as it helps to ensure that the data used in a system is well-organized, consistent, and accurate.

11)Why are control components necessary in traditional software and generally not required in object-oriented software

Control components, such as loops and conditional statements, are necessary in traditional software to control the flow of execution through a program. They are used to specify the order in which different parts of the program should be executed, and to make decisions based on the values of variables or other conditions.

In object-oriented software, control components are generally not required because the behavior of the system is determined by the interactions between objects. Objects communicate with each other through messages, and the flow of control is determined by the way in which these messages are passed between objects.

There are several reasons why control components are generally not required in object-oriented software:

1. Objects are self-contained units of functionality that encapsulate their own data and behavior. This means that the flow of control within an object is determined by the methods and operations that are defined within the object.
2. The relationships between objects are dynamic and flexible, and can be changed at runtime. This allows objects to be composed and reconfigured in different ways to achieve different behaviors.
3. Objects are designed to be reusable, and can be easily integrated into different contexts and systems. This reduces the need for control components, as the behavior of the system is determined by the interactions between objects rather than by the flow of control through a program.

Overall, control components are necessary in traditional software to control the flow of execution through a program, but are generally not required in object-oriented software because the behavior of the system is determined by the interactions between objects.

12)Disadvantages of Object Oriented Analysis

In lecture notes:(just given about Object Oriented Analysis but no info about adv and disadv)

Object Oriented Analysis

the system analysis or object-oriented analysis phase of software development, the system requirements are determined, the classes are identified and the relationships among classes are identified

The three analysis techniques that are used in conjunction with each other for object-oriented analysis are object modelling, dynamic modelling, and functional modelling

Object Modelling Object modelling develops the static structure of the software system in terms of objects. It identifies the objects, the classes into which the objects can be grouped into and the relationships between the objects. It also identifies the main attributes and operations that characterize each class

The process of object modelling can be visualized in the following steps

- Identify objects and group into classes
- Identify the relationships among classes
- Create user object model diagram
- Define user object attributes
- Define the operations that should be performed on the classes
- Review glossary

Object-oriented analysis (OOA) is a widely-used approach to software design that is based on the principles of object-oriented programming (OOP). While OOA has many advantages, there are also some potential disadvantages to consider:

1. **Complexity:** OOA can be more complex than other approaches to software design, as it involves modeling a system as a set of interacting objects with complex relationships and behaviors. This can make it more difficult to understand and modify the system, especially for developers who are new to OOP.

2. **Rigidity:** Once the objects in a system have been defined, it can be difficult to make changes to their properties and behaviors. This can make the system less flexible and adaptable, and can make it harder to modify and evolve the system over time.
3. **Overhead:** OOA involves a lot of upfront design and modeling work, which can be time-consuming and may not be necessary for simple or straightforward systems. This can add overhead to the development process, and may not be cost-effective for smaller or simpler projects.
4. **Limited reuse:** While OOA can facilitate the reuse of objects within a system, it can be more difficult to reuse objects across different systems or contexts. This can limit the benefits of object-oriented design, especially in environments where reuse is a key concern.

Overall, while OOA has many advantages, it is important to carefully consider the potential disadvantages and decide whether it is the right approach for a given project.

13)Advantages of Object Oriented Analysis

Object-oriented analysis (OOA) is a widely-used approach to software design that is based on the principles of object-oriented programming (OOP). OOA has several advantages over other approaches to software design:

1. **Modularity:** OOA allows developers to create modular, self-contained objects that encapsulate their own data and behavior. This makes it easier to understand and modify the system, as the behavior of each object is clearly defined and isolated from the rest of the system.
2. **Reusability:** OOA facilitates the reuse of objects within a system, as well as across different systems. This can save development time and improve the maintainability of the system.
3. **Extensibility:** OOA allows developers to easily extend and modify the system by adding or modifying the objects in the system. This can make it easier to evolve the system over time to meet changing requirements.
4. **Maintainability:** OOA helps to improve the maintainability of a system by making it easier to understand and modify the system. The modular structure of objects

and the clear separation of responsibilities can make it easier to fix bugs and implement new features.

5. **Flexibility:** OOA allows developers to model real-world entities and concepts in a more flexible and intuitive way. This can make it easier to understand and work with complex systems, and can improve the usability of the system.

Overall, OOA is a powerful approach to software design that offers many advantages, including modularity, reusability, extensibility, maintainability, and flexibility.

14)Disadvantages of Structured Analysis

In lecture notes:(just given about Structured Analysis but no info about adv and disadv)

Structured Analysis

Structured Analysis is a development method that allows the analyst to understand the system and its activities in a logical way. It is a systematic approach, which uses graphical tools that analyze and refine the objectives of an existing system and develop a new system specification which can be easily understandable by user

It has following attributes

- It is graphic which specifies the presentation of application
- It divides the processes so that it gives a clear picture of system flow. • It is logical rather than physical i.e., the elements of system do not depend on vendor or hardware
- It is an approach that works from high-level overviews to lower-level details

Structured Analysis Tools

During Structured Analysis, various tools and techniques are used for system development. They are

- Data Flow Diagrams
- Data Dictionary
- Decision Trees
- Decision Tables
- Structured English
- Pseudocode

15)Advantages of Structured Analysis

1)Greater Efficiency

Much like with building information modelling, structural analysis improves project efficiency. Following the steps means there's little room to miss the important things. You cover every base, from the issues with an existing model through to the alternatives to the model you come up with.

You have a plan in place and establish feasibility at an early stage. Furthermore, you create a structure that all stakeholders can work within.

Everybody understands their role within your projects and works as per the system's design.

2) It Takes Client Needs into Account

In most cases, your clients have come to you with a problem. They may need you to redesign an existing model. Or, they have certain specifications your model must meet.

One of the benefits of using structural analysis is that the technique takes the client's needs into account from the beginning. The process of analysing and modelling the current system helps. It allows you to ask questions of the client, and gives them time to tell you about specific issues.

This continues throughout the process. Modelling the physical environment offers a real-world interpretation of your model. You also consider the alternatives, which gives your client something else to think about.

3)Better Risk Analysis

One of the benefits of using structural analysis is that it challenges you to look at all aspects of a project. As a result, you analyse risks constantly. The data points out areas of contention, so you can adapt the model to them.

This benefit also comes back to the client-focus. You have a better understanding of what the client wants. As a result, you take on less risk when designing the model. In some cases, you may even get feedback from the clients' customers. For example, you could expect more feedback when

replacing an older model. The clients' customers may have raised complaints, which you can use to shape your own efforts.

4)It Makes Projects More Manageable

You confront the issue of feasibility during the early stages of the process. You collect data, consider the problem, and come up with a solution. As a result, the project grows from a solid foundation. You understand how you'll manage every stage of the project, and who you must involve.

Using structural analysis when developing models helps you to determine the workload your team takes on. You can assign tasks confidently, and ensure you have the correct people in place.

It comes down to the planning. The data you have available shows you exactly what you need to do. Your models build on this information. As the process continues, you identify problem areas and tasks. For example, you'll know when to invite contributions from electricians. Plus, you can tackle potential structural issues early in the process.

Structural analysis gives you control of the project from start to finish.

5)Create Updateable Models

Structural analysis doesn't just take the client's present needs into account. It also considers future issues. It allows for the creation of models you can modify and update as time goes on.

This is particularly useful during the pre-construction stage. Your models may face last minute changes. An uncaught design flaw could cause issues, or the client may change their requirements.

Each demands modification of the existing model. Sometimes, you may need to add to your model based on the client's new requirements. Combine the structural analysis approach with the right building information modelling software. This helps in several ways.

For example, you can code new data sets into the model to reflect the changes. The building information modelling structure also allows you to share these changes with your team. Everybody can contribute to the ongoing maintenance and updating of the model during development. As a result, new client demands don't set the project back too far. Your model can adapt to them, in preparation for the construction phase.

16)Advantages of object relational model

Advantages of Object Relational model

The advantages of the Object Relational model are

Inheritance

The Object Relational data model allows its users to inherit objects, tables etc. so that they can extend their functionality. Inherited objects contains new attributes as well as the attributes that were inherited

Complex Data Types

Complex data types can be formed using existing data types. This is useful in Object relational data model as complex data types allow better manipulation of the data

Extensibility

The functionality of the system can be extended in Object relational data model. This can be achieved using complex data types as well as advanced concepts of object oriented model such as inheritance

17)Disadvantages of object relational model

Disadvantages of Object Relational model

The object relational data model can get quite complicated and difficult to handle at times as it is a combination of the Object oriented data model and Relational data model and utilizes the functionalities of both of them

18)Identifying Events with the Use Case(doubt)

In software engineering, a use case is a description of a system's behavior as it responds to a request from a user or another system. Identifying events is an important part of the use case process, as it helps to define the trigger points for the system's behavior.

There are several ways to identify events in the use case process:

1. **Identify the external triggers:** Events are often triggered by external actions or requests, such as a user input or a request from another system. It is important

to identify these external triggers and describe how the system responds to them.
2. **Identify the internal triggers:** Events can also be triggered by internal conditions or states, such as the completion of a task or the expiration of a timer. It is important to identify these internal triggers and describe how the system responds to them.
3. **Identify the event flow:** The sequence of events in a use case is called the event flow. It is important to identify the event flow and describe the sequence of actions and decisions that the system takes in response to each event.
4. **Identify the exception conditions:** It is also important to identify any exception conditions that may arise during the event flow, and describe how the system handles these conditions.

By identifying the events in a use case, developers can better understand the behavior of the system and how it responds to different requests and conditions. This can help to ensure that the system is reliable, flexible, and easy to understand and modify.

19)What are system requirements? Explain in detail.

In software engineering, system requirements are the specifications that define the capabilities, constraints, and characteristics of a software system. System requirements are used to define the functionality and performance of a system, and to guide the design and development process.

There are several types of system requirements that may be defined for a software system, including:

1. **Functional requirements:** These are the requirements that define what the system is supposed to do. They may include specific features or capabilities that the system must have, as well as the requirements for input, output, and processing.
2. **Non-functional requirements:** These are the requirements that define how the system is supposed to behave. They may include requirements for performance, reliability, security, usability, and other qualities of the system.

3. **Constraints:** These are the requirements that define the boundaries or limits within which the system must operate. They may include requirements for hardware, software, or other resources that the system must be compatible with.
4. **Assumptions:** These are the requirements that define the conditions that are assumed to be true when the system is being designed or developed. They may include assumptions about the environment in which the system will be used, the users of the system, or other factors.

It is important to define and document the system requirements carefully, as they will serve as the foundation for the design and development of the system. Clearly defined system requirements can help to ensure that the system is reliable, flexible, and easy to understand and modify.

20)What is requirement? Explain about user requirements with an example

In software engineering, a requirement is a specification that defines a capability, constraint, or characteristic that a software system must have. Requirements are used to define the functionality and performance of a system, and to guide the design and development process.

User requirements are the requirements that are defined by the users of a software system. They represent the needs and expectations of the users, and they are used to guide the design and development of the system.

For example, consider a software system that is being developed for a bank. The user requirements for this system might include:

1. **The ability to create and manage customer accounts:** Users (e.g., bank employees) must be able to create new customer accounts, view and update account information, and perform other account-related tasks.
2. **The ability to process transactions:** Users must be able to process deposits, withdrawals, and other transactions on customer accounts.
3. **The ability to generate reports:** Users must be able to generate reports on customer accounts and transactions, such as account balance reports and transaction history reports.

4. **Security:** The system must be secure, with measures in place to prevent unauthorized access and protect sensitive customer data.