S.E MIDS-2 IMP

SAQ:

1. List out Data Modelling examples

- Entity-Relationship Diagrams (ERD)
- Unified Modelling Language (UML) Class Diagrams
- Hierarchical Data Model
- Network Data Model
- Relational Data Model

2. List out the Golden Rules for user interface design

Theo Mandel coins three golden rules as:

1. Place the user in control.
2. Reduce the user's memory load.
3. Make the interface consistent.

These golden rules actually form the basis for a set of user interface design principles that guide this important aspect of software design.

3. Define Design Evaluation

Design Evaluation refers to the process of assessing and analysing the effectiveness, efficiency, and quality of a software system's design. It involves reviewing and critiquing the design artifacts to ensure they meet the specified requirements, adhere to design principles, and align with the project's objectives. The evaluation helps identify design flaws, potential improvements, and areas where the design might need further refinement.

4. What is Debugging?

Debugging is the process of identifying, analysing, and resolving defects or errors that occur in a software program. When a program does not behave as expected or produces incorrect results, developers use debugging techniques and tools to trace the root cause of the issue and fix it. The goal of debugging is to eliminate the bugs and ensure the software functions correctly.

5. Compare Testing and Debugging

Testing and debugging are both essential activities in software development, but they serve different purposes:

- Testing involves systematically evaluating a software system to identify defects, inconsistencies, or deviations from requirements. It aims to verify that the software meets specified criteria.
- Debugging, on the other hand, focuses on the specific process of identifying and correcting defects or bugs found during testing or development.

6. Compare white box testing and black box testing

- White box testing is a testing technique where the tester has access to the internal code and structure of the software. Test cases are designed based on the knowledge of the program's internal logic.
- Black box testing, on the other hand, is a testing technique where the tester evaluates the software's functionality without knowledge of its internal code. Test cases are designed based on the software's specifications and requirements.

7. What is Black box testing?

Black box testing is a software testing technique where the tester assesses the functionality of a software system without having access to its internal code or structure. Test cases are designed based on the software's specifications and requirements. The tester focuses on input-output behaviour and examines whether the software produces the expected outputs for given inputs, regardless of how the internal processing occurs.

8. What are objectives of software design?

- Translating system requirements into a detailed and well-organized design.
- Creating a blueprint that guides the development process.
- Making the software modular, maintainable, and scalable.
- Ensuring the design adheres to established architectural patterns and design principles.
- Minimizing complexity and enhancing the software's understandability.
- Considering potential constraints like budget, time, and resources.

9. What is CMMI?

CMMI stands for Capability Maturity Model Integration. It is a process improvement approach used to help organizations enhance their software development and management processes. CMMI provides a set of best practices and guidelines that help organizations improve their capabilities and achieve higher levels of maturity in their software development processes. It is widely used for assessing and benchmarking process maturity in various industries.

10. Define user interface design and list out its types

User Interface (UI) design refers to the process of creating visually appealing and user-friendly interfaces for software applications or systems. It focuses on the presentation and interaction layers of the software, ensuring that users can interact with the system efficiently and intuitively. Types of user interface design include:

- Graphical User Interface (GUI): This involves the use of graphical elements like icons, buttons, and menus to interact with the software.
- Command-Line Interface (CLI): This interface requires users to input text-based commands to interact with the software.

LAQ:

1. Explain about object oriented analysis and design and list out the advantages and disadvantages

Object-oriented analysis is a software engineering technique that focuses on identifying, modelling, and understanding the objects (or entities) within a system and their interactions. It is the first step in the object-oriented software development process and plays a crucial role in designing high-quality, maintainable, and extensible software systems. Some key concepts associated with object-oriented analysis are:

1. **Objects:** An object is a self-contained entity that encapsulates both data (attributes or properties) and behaviour (methods or functions). Objects represent real-world entities, concepts, or abstract elements in the software system.

2. **Classes:** A class is a blueprint or template that defines the structure, behaviour, and attributes of objects. It serves as a reusable pattern for creating objects.

3. **Abstraction:** Abstraction is the process of simplifying complex systems by focusing on essential properties while ignoring irrelevant details. It allows us to model and represent real-world entities in a software system using classes and objects.

4. **Relationships:** Object-oriented analysis involves identifying relationships between objects, such as associations (linkages between objects), dependencies (one object depends on another), and inheritances (a class inherits properties and behaviour from another class).

5. **Use Cases:** Use cases describe the interactions between the system and its users or external entities. They define the system's functionalities from a user's perspective and help in understanding the system's requirements.

6. **Analysis Models:** Analysis models, such as class diagrams, interaction diagrams (e.g., sequence diagrams and collaboration diagrams), and state transition diagrams, are used to represent the system's structure, behaviour, and dynamics during the analysis phase.

7. **Iterative and Incremental Approach**: Object-oriented analysis is typically performed iteratively and incrementally, allowing for continuous refinement and improvement of the analysis models based on feedback and changing requirements.

8. **Inheritance:** Inheritance is a fundamental concept in object-oriented programming that allows a class to inherit properties and behaviour from another class. In OOA, inheritance helps in organizing and reusing common attributes and methods. A subclass can inherit from a superclass, acquiring its characteristics while also adding or modifying its own unique features.

9. **Messages:** In OOA, communication between objects occurs through messages. A message represents a request from one object to another object to perform a specific action. Messages encapsulate the intent or the method invocation and are used to trigger behaviour within objects.

10. **Polymorphism:** Polymorphism refers to the ability of objects of different classes to respond to the same message in different ways. It allows objects to exhibit different behaviours based on their specific implementations of methods. Polymorphism enhances flexibility and extensibility in the software system by enabling interchangeable and dynamic behaviour.

11. **Design Classes:** During OOA, design classes are identified to represent the objects in the system. These classes capture the attributes (data) and behaviour (methods) of the objects. Design classes serve as the blueprints for creating object instances during the software design and implementation phases.

Advantages of OOAD:

- Reusability: Objects can be reused in different contexts, leading to more efficient development.
- Modularity: Objects encapsulate data and behaviour, enabling easier maintenance and troubleshooting.
- Flexibility: Changes to the system are localized to specific objects, reducing the impact on other parts of the system.
- Scalability: OOAD allows for a scalable design, making it easier to handle large and complex systems.
- Abstraction: OOAD uses abstraction to hide unnecessary details and focus on essential aspects of the system.

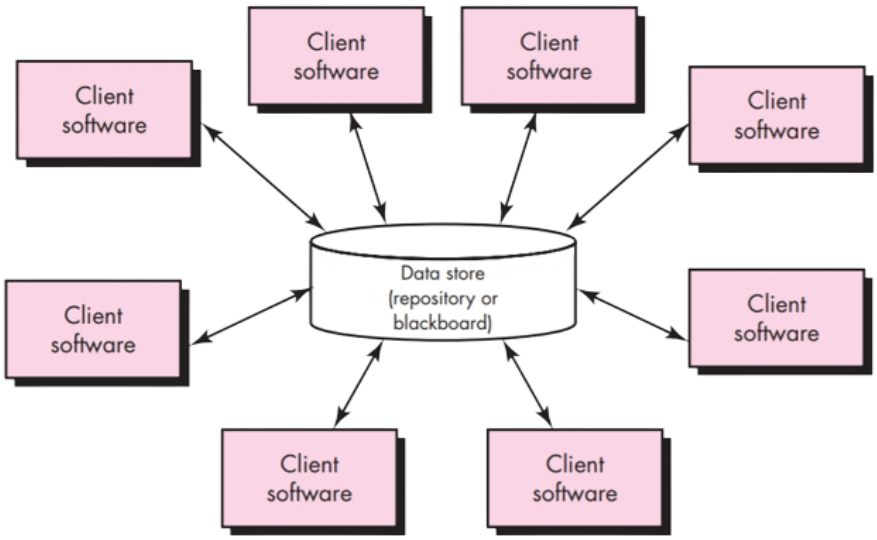
Disadvantages of OOAD:

- Learning Curve: OOAD can have a steeper learning curve for developers transitioning from procedural to object-oriented paradigms.
- Overhead: Implementing object-oriented systems can sometimes lead to increased memory and processing overhead.
- Complexity: In some cases, the object-oriented approach can introduce additional complexity, especially for smaller projects.
- Design Overhead: OOAD requires careful design planning and modelling, which can be time-consuming.

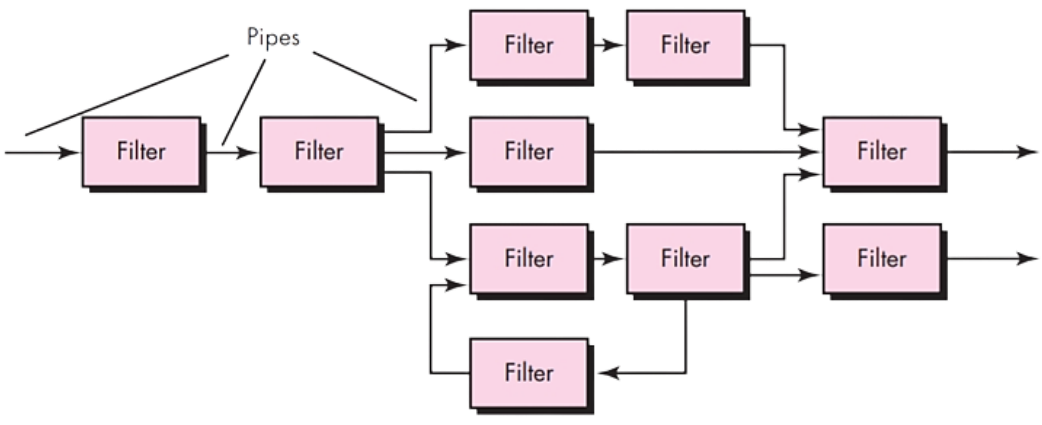
2. Explain the Taxonomy of Architectural Styles

(a) Data-centred architectures: A data store (e.g., a file or database) resides at the centre of this architecture and is accessed frequently by other components that update, add, delete, or otherwise modify data within the store. Client software accesses a central repository. In some cases, the data repository is passive. That is, client software accesses the data independent of any changes to the data or the actions of other client software. A variation on this approach transforms the repository into a “blackboard” that sends

notifications to client software when data of interest to the client changes. Data-centred architectures promote integrability. That is, existing components can be changed and new client components added to the architecture without concern about other clients. In addition, data can be passed among clients using the blackboard mechanism. Client components independently execute processes.



(b) Data-flow architectures: This architecture is applied when input data are to be transformed through a series of computational or manipulative components into output data. A pipe-and-filter pattern has a set of components, called filters, connected by pipes that transmit data from one component to the next. Each filter works independently of those components upstream and downstream, is designed to expect data input of a certain form, and produces data output (to the next filter) of a specified form. However, the filter does not require knowledge of the workings of its neighbouring filters. If the data flow degenerates into a single line of transforms, it is termed batch sequential. This structure accepts a batch of data and then applies a series of sequential components (filters) to transform it.

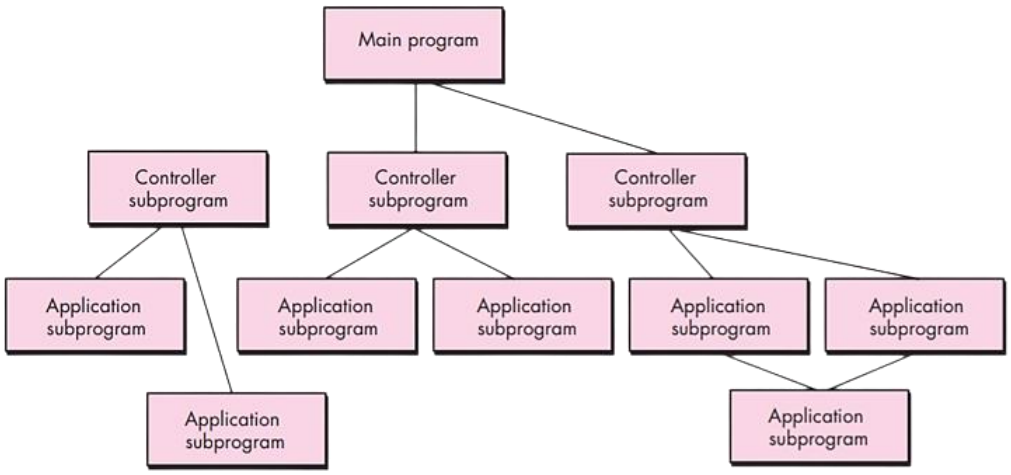


Pipes and filters

(c) Call and return architectures: This architectural style enables to achieve a program structure that is relatively easy to modify and scale. A number of sub-styles exist within this category:

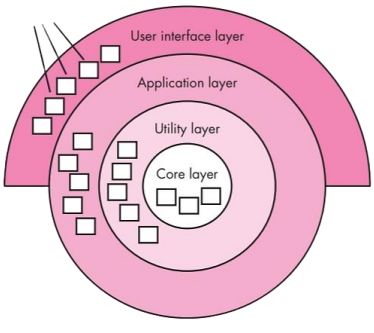
- ***Main program/subprogram architectures*-** This classic program structure decomposes function into a control hierarchy where a "main" program invokes a number of program components that in turn may invoke still other components.

- ***Remote procedure calls architectures*-** The components of main program/subprogram architecture is distributed across multiple computers on a network.



(d) Object-oriented architectures: The components of a system encapsulate data and the operations that must be applied to manipulate the data. Communication and coordination between components are accomplished via message passing.

(e) **Layered architectures:** In this a number of different layers are defined, each accomplishing operations that progressively become closer to the machine instruction set. At the outer layer, components service user interface operations. At the inner layer, components perform operating system interfacing. Intermediate layers provide utility services and application software functions.



3. Distinguish between Alpha and Beta Testing

Alpha Testing:

1. Timing: Alpha testing is the first phase of testing and is conducted within the development environment before the software is released to external users.

2. Participants: Alpha testing is carried out by the internal development team or a select group of trusted testers who are part of the development organization.

3. Environment: Alpha testing occurs in a controlled environment, usually on-site, where the developers have access to the source code and can closely monitor the testing process.

4. Objective: The primary objective of alpha testing is to identify defects, bugs, and design flaws in the software before it is exposed to external users. It helps the development team to evaluate the system's overall functionality and stability.

5. Scope: Alpha testing is usually a focused and limited test, where specific features and functionalities are targeted for evaluation.

6. Feedback Cycle: Since the testing is done internally, feedback and communication with testers are more immediate and direct. Developers can quickly respond to reported issues and fix them promptly.

7. Confidentiality: Alpha testing is conducted in a closed environment, and the software is typically not shared with the public or customers.

Beta Testing:

1. Timing: Beta testing takes place after alpha testing and is conducted in a real-world environment, typically outside the development organization.

2. Participants: Beta testing involves external users, known as beta testers, who are not part of the development team. These testers are representative of the target audience or end-users.

3. Environment: Beta testing is conducted in a more diverse and varied environment, as it involves different setups, configurations, and user scenarios.

4. Objective: The main objective of beta testing is to collect valuable user feedback and discover potential issues in the software under real-world usage conditions. It helps assess user satisfaction and validate the software's usability and performance.

5. Scope: Beta testing is broader in scope, as it covers the entire software application and its various functionalities.

6. Feedback Cycle: The feedback loop in beta testing might take longer, as it involves external users. The testers report issues and experiences, which may take some time for the development team to address.

7. Confidentiality: Beta testing often involves a wider audience, but access to the beta version might be restricted to selected participants through invitations or sign-ups.

4. Outline coding standards and guidelines for selecting a programming language

Coding Standards and Guidelines:

Coding standards are a set of rules and conventions that developers follow while writing code to ensure consistency, readability, and maintainability of the software. Some common coding standards and guidelines include:

- Consistent indentation and formatting.
- Meaningful variable and function names.
- Avoiding the use of magic numbers and hard-coded values.
- Limiting the size of functions and methods.
- Proper commenting and documentation.
- Following a consistent naming convention for classes and files.

Guidelines for selecting a programming language:

- Consider the project requirements and domain.
- Evaluate the language's performance and scalability.
- Assess the availability of libraries and frameworks.
- Examine the language's community and support.
- Determine the language's ease of learning and developer familiarity.
- Consider the language's security features and vulnerabilities.
- Analyse the long-term sustainability of the language.

5. Explain the Taxonomy of Architectural Styles

Same as 2.

6. Illustrate Class-based Modelling

Class-based modelling identifies classes, attributes and relationships that the system will use. In the airline application example, the traveller/user and the boarding pass represent classes. The traveller's first and last name, and travel document type represent attributes, characteristics that describe the traveller class. The relationship between traveller and boarding pass classes is that the traveller must enter these details into the application in order to get the boarding pass, and that the boarding pass contains this information along with other details like the flight departure gate, seat number etc.

Class based modelling represents the object. The system manipulates the operations.

The elements of the class based model consist of classes and object, attributes, operations, class – responsibility - collaborator (CRS) models.

Classes

Classes are determined using underlining each noun or noun clause and enter it into the simple table.

Classes are found in following forms:

External entities: The system, people or the device generates the information that is used by the computer based system.

Things: The reports, displays, letter, signal are the part of the information domain or the problem.

Occurrences or events: A property transfer or the completion of a series or robot movements occurs in the context of the system operation.

Roles: The people like manager, engineer, salesperson are interacting with the system.

Organizational units: The division, group, team are suitable for an application.

Places: The manufacturing floor or loading dock from the context of the problem and the overall function of the system.

Structures: The sensors, computers are defined a class of objects or related classes of objects.

Attributes

Attributes are the set of data objects that are defining a complete class within the context of the problem.

For example, 'employee' is a class and it consists of name, Id, department, designation and salary of the employee are the attributes.

Operations

The operations define the behaviour of an object.

The operations are characterized into following types:

- The operations manipulate the data like adding, modifying, deleting and displaying etc.
- The operations perform a computation.
- The operation monitors the objects for the occurrence of controlling an event

CRS Modelling

- The CRS stands for Class-Responsibility-Collaborator.
- It provides a simple method for identifying and organizing the classes that are applicable to the system or product requirement.
- Class is an object-oriented class name. It consists of information about sub classes and super class
- Responsibilities are the attributes and operations that are related to the class.
- Collaborations are identified and determined when a class can achieve each responsibility of it. If the class cannot identify itself, then it needs to interact with another class.

7. Outline current trends in software engineering

1. DevOps and CI/CD: Integration of development and operations for continuous integration and continuous delivery.

2. Artificial Intelligence and Machine Learning: Widespread adoption of AI/ML techniques to enhance software applications.

3. Cloud Computing: Growing use of cloud-based infrastructure and services for scalable software deployments.

4. Internet of Things (IoT): Integration of software with IoT devices for connected solutions.

5. Agile and Agile at Scale: Continued emphasis on Agile methodologies for iterative development and scaling Agile practices.

6. Micro services and Server less Architectures: Shift towards building applications with smaller, independent services.

7. Ethical Software Development: Increasing awareness of ethical considerations in software engineering.

8. Remote Collaboration and Distributed Teams: Accelerated adoption of remote work and distributed software development teams.

9. Continuous Testing: Automation of testing for faster feedback and improved software quality.

10. Edge Computing: Utilizing edge devices for processing data closer to the data source.

8. Differentiate between Verification and Validation.

Verification:

- Definition: Verification is the process of evaluating software artifacts to determine whether they satisfy specified requirements and meet predefined standards. It involves checking and reviewing the work products at various stages of the development process.

- Objective: The main objective of verification is to ensure that the software is being built correctly according to its design and requirements. It focuses on detecting and eliminating defects early in the development lifecycle.

- Activities: Verification activities include code reviews, inspections, walkthroughs, and static analysis. These activities are usually carried out by individuals other than the authors of the artifacts to provide an independent assessment.

- Goal: The goal of verification is to answer the question, "Are we building the product right?"

Validation:

- Definition: Validation is the process of evaluating the final software product to determine whether it satisfies the specified requirements and meets the intended use. It involves executing the software in a real-world environment and comparing its actual behaviour against the expected behaviour.

- Objective: The main objective of validation is to ensure that the software meets the user's needs and expectations and performs its intended functions correctly.

- Activities: Validation activities include testing the software through various techniques such as unit testing, integration testing, system testing, and acceptance testing. The software is evaluated for functional correctness, performance, usability, and other relevant attributes.

- Goal: The goal of validation is to answer the question, "Are we building the right product?"