QUIZ

Software Engineering and Testing

SUBMITTED BY: AMARESH MUDDEBIHAL

REG NUMBER: 20030141IT015

ALLIANCE UNIVERSITY

QUIZ
Software Engineering and Testing

1. **Basic Object-Orientation Concepts, Class Relationships, How to Identify Class Relationships.**

Ans: **Basic Object-Orientation Concepts**

Object-orientation is a programming paradigm that has "objects" as its basis in which class instances encapsulate data together with their behavior. Classes in object-orientation establish both the structure and functions of objects. Each class contains attributes comprising fields or properties, which signify the state, as well as methods-function or procedure -that describes the behavior. The four chief principles of object-orientation include encapsulation, inheritance, polymorphism, and abstraction. In the bundling of data and methods within a class combined with the restriction of access to internal details, comes modularity and safety. Inheritance allows a class (sub-class) to derive its properties and methods from another class (superclass), thus promoting code reuse. Polymorphism is an object-oriented technique whereby the same method behaves differently according to context. Abstraction simplifies complex systems by exposing only the required aspects while concealing the details inside.

Class Relationships

Class relationships are a representation of the different classes and how they are related to each other in an object-oriented system. Class relationships within an object-oriented model make it possible to better model aspects of the real world in software. Association is one of the most basic kinds of class relationship, wherein one class merely has a relation with another class. Usually, the association is drawn by connecting the two classes using a line. Aggregation is a stronger form of association that depicts a "whole-part" relationship wherein the part can exist independently of the whole, such as a department containing employees. Composition, a stronger form of aggregation, implies a "whole-part" relationship wherein the part cannot exist independently of the whole, such as a house with rooms. Inheritance is a relationship where one class (child) derives from another (parent), acquiring its properties and behaviors. Dependency arises where one class depends on another to perform its function and is typically drawn with a dashed line containing an arrow.

Identifying Class Relationships

Identifying class relationships involves understanding how the entities of the problem domain operate in the system. This is done by first creating use case diagrams or by examining requirements and listing the primary classes involved. For each class, one would then determine the responsibilities in order to understand how they relate to one another. A class such as an Order should, therefore, have association with a Customer class since orders are made by customers. Likewise, if an Order possesses multiple OrderItems then this association can be considered as aggregation since the OrderItems are not dependent on some other class. Composition can be established in such cases where one class is closely coupled to another. For instance, the Engine of a Car; here, Engine has strong relation with Car as it is a part and parcel of the Car. Identify inheritance by looking for hierarchical relationships where one class would logically inherit characteristics from another-for example, a Manager class might inherit

from an Employee class. And dependency relationships-when one class needed to use another in order to perform an action-would be identified by deciding when one class was depending on another class to complete an action, such as the PaymentProcessor class being dependent upon the BankService class.

2. Implementing object-oriented modeling techniques using UML.

Ans : Unified Modeling Language is the standard graphical language for modeling object-oriented systems. There is a whole set of diagrams which UML represents various aspects of a system, so it becomes an effective tool in the implementation of object-oriented modeling techniques. UML's prime objective is to aid the creation of a blueprint which helps in the design of a structure and understanding of the architecture, behavior, as well as interactions of the system.

Class diagrams can be represented with high importance in object-oriented modeling because they represent the static structure of the system. A class diagram defines classes, their attributes, methods, and relationships between them like association, dependence, inheritance, and composition. For instance, a class diagram for a library management system may define classes such as Book and Member and Librarian along with attributes as title, author, memberID, and borrowingBook() and returningBook(). Relationships among classes are represented with connectors: lines to represent association, diamonds for showing aggregation/composition, and arrows for inheritance.

Actors, who are users or other systems, interact with use cases, which are application functions. For example, in an application such as food delivery, actors can be Customer, Restaurant, and Delivery Person interacting with use cases Place Order, Prepare Food, and Deliver Order. Requirements can be captured by using these use cases and will become a medium of communication between stakeholders and development teams.

Sequence Diagrams model the dynamic behavior of a system by capturing, over time, the interactions between objects and each other. These diagrams are very helpful for describing how objects will collaborate in some use case. For example, in the CRM system for customer service, one can have a sequence diagram that shows how interactions start through a Customer asking his query and then subsequently through the Customer Service Agent making the query to the Support System while communicating to obtain or update information. The sequence diagram illustrates the message flow, and time sequence of these interactions.

Activity Diagrams are used to draw workflows and procedures in the system. They are used to model flow of control or data from one activity to another. Therefore, an online purchasing process can be shown using activities such as Browse Items, Add to Cart, Enter Payment Details, Payment Verification, and Order Confirmation. It can also include decision points, parallel actions and end points to make the process explicit.

State Diagrams are representations of the lifecycle of an object, showing its states and transitions resulting from events. In the case of a traffic light control system, for example, it would be made up of the Green, Yellow, and Red states. It would reveal when and how the state of the system changes over time, responding to one or more specific event, possibly using timers or sensors.

System physical aspects are modeled through the use of component diagrams and deployment diagrams. A component diagram explains how to organize or interconnect software components,

whereas a deployment diagram describes the physical deployment of artifacts across different hardware nodes.

These UML modeling techniques will enable the developers to understand complex systems, ease coherent communication, and enable the simplification of the design process. The structured approach disintegrates the system into workable parts, class relationships, model interactions, and maps out processes thereby enabling an easy development of robust object-oriented software systems.

3. An Object-Oriented Analysis and Design (OOAD) Methodology-applying knowledge to create UML models.

Ans: Object-Oriented Analysis and Design, or OOAD, is the design approach for analyzing a system. It is the process of visualizing a system as a collection of interacting objects that are defined by the roles or responsibilities it takes within the system. Here, in this context, object-oriented principles would be applied to model complex systems to improve not only understanding but communication as well throughout the development lifecycle. In applying OOAD, Unified Modeling Language is utilized to create what could be considered as comprehensive diagrams about the structure and behavior of the system.

1. Requirements Gathering and Analysis: The most initial process of OOAD is gathering and analyzing requirements of the system. As a part of this process, it develops and involves stakeholders to formulate key functionalities, constraints, and objectives. Use case diagrams are very crucial in this phase as they represent a very high-level view from the user's perspective about what the system should do. Each use case stands for a specific interaction between an actor-usually an actual or external system-and the system itself, thus aiding requirement capturing and user needs understanding.

2. Conceptual Modeling: In the conceptual phase, analysts identify the key objects that will form part of the system and define their relationships. Class diagrams are developed for depicting the structure of these objects and how the attributes, methods, and relationships between them are present. For example, classes such as Product and Customer would require inclusion on a class diagram for an e-commerce system along with associations that delineate how these classes relate to one another. Attributes and methods are determined based on whether they are associated with functionality described in use cases.

3. Dynamic Modeling: Dynamic modeling is a drawing of the behavior of the system along with how objects tend to interact at different times. Sequence diagrams usually are very applicable in doing this. They are often used to describe the flow of messages between objects within a use case or operation. For example, for the use case Place Order it may describe interactions between Customer, Shopping Cart and Payment Gateway objects that represent how the order is created and processed.

4. Modeling States and Activities with State and Activity Diagrams: A state diagram is used to represent the lifecycle of an object-the various states it may be in, as well as how these states relate to each other. An example of a state diagram for an Order object might describe states like Pending, Confirmed, Shipped, and Delivered, along with transitions made by respective actions like payment confirmation or shipping updates. An activity diagram is used to represent workflows and processes. An activity diagram of the process flow for order processing might be Order Received, Verify Payment, Pack Items, Dispatch along with decision nodes wherein different paths are taken.

5. Detailed Class Design After the high-level models are defined, detailed class diagrams are created, with specific attributes, methods, and relationships. It is at this stage of development where inheritance hierarchies and interfaces can be specified for modular and reusable code structures; design patterns, such as Observer or Factory patterns, may be applied to solve recurring design challenges and allow for scalability and maintainability improvement.

6. Component and Deployment Modeling In case of large systems, component diagrams can be presented to show the organization into modules or components of various parts of the system. They help the users understand the dependencies among different parts of the software. The deployment diagrams represent the physical deployment of the software components across servers, databases, and other hardware nodes. These depict the operational architecture of the system.

Applying UML in OOAD: A core of OOAD is UML, which offers a standard visual language allowing developers, analysts, and stakeholders to share each other. Through class diagrams, sequence diagrams, state diagrams, activity diagrams, use case diagrams, component diagrams, and deployment diagrams, teams are able to document and design the system according to the principles of object-oriented analysis. This approach also enforces a clear, consistent understanding of the system's design and behavior, enhances code reuse, and facilitates iterative development and refinement.

4. Unified Process, Overview of The OOAD Methodology, Use Case Model Development, Domain Modelling.

Ans : Unified Process (UP) : Unified Process (UP) is a framework for an iterative and incremental, software development process. It is centered on using a disciplined approach to task and responsibility assignment within a development team in a highly effective way. UP can produce high-quality software at predictable schedules and budgets. UP is flexible and adaptable and makes it possible to modify it to suit specific project requirements. The most well-known application of UP is the Rational Unified Process, commonly referred to as RUP, which is divided into four major stages: Inception, Elaboration, Construction, and Transition. All these stages consist of iterations that produce and incrementally refine the product by detailed requirements analysis, system design, implementation, and testing.

Overview of the OOAD Methodology

Object-Oriented Analysis and Design, or OOAD in short, refers to an approach to structured analysis and design for a system based on object-oriented concepts. The OOAD methodology applies a straightforward systematic process towards transforming the requirements into a design that can be implemented with OOP. Some of the key steps include the following:

1. Requirements Gathering: Identifying what the system needs to do through interaction with stakeholders on both functional and nonfunctional requirements.

2. Analysis Phase: System structure and behavior by identifying key objects, their attributes, and their interactions.

3. Design Phase: Refining the analysis model into a blue print for the implementation of the system in class hierarchies, relationships, and interactions between objects.

4. Implementation Phase: The design is translated into actual code through an OOP language, ensuring adherence to design principles.

5. Testing and Refining: Checking that the system will behave exactly as desired by running it on other test examples and refining the course of action if required.

OOAD emphasizes its dependency on UML, or Unified Modeling Language, to model and represent different facets of the system in class diagrams, sequence diagrams, use case diagrams, and state diagrams.

Develop a Use Case Model

A use-case model is an important tool in OOAD for capturing the functional requirements of a system. It identifies and organizes functionalities and interactions of a system with the outside world. Use case models are represented by use case diagrams where various actors are shown, with their use cases. Each use case relates to a different interaction between an actor and the system that would result in a desirable and valuable outcome.

To make a use case model:

1. Actors: These are people or organizations that interact with the system. The actors can be user, external systems, or any other device.

2. Identifying Use Cases: Main ways through which actors interact with the system to achieve the objective.

3. Describe Use Cases: Describe each use case with a detailed description of the primary flow of events, alternate flows, and exceptions.

4. Purge and Connect: organize use cases, and demonstrate relationships like include, extend, and generalizations to indicate similarities and differences between the behavior of use cases.

Domain Modeling

This technique provides an analysis of the conceptual structure of the problem domain, thereby pinpointing the system's entities, relationships, and rules in its environment. A domain model is normally presented with a class diagram in UML, which depicts classes and their attributes and methods along with relationships. It represents a bridge between the problem space, that is, the requirements and the solution space, that is, the system design.

Steps to produce a domain model are:

Identification of Domain Entities: Understand requirements to list all the crucial objects that actually exist in the system, that is, Customer, Order, Product etc.

Definition of Attributes and Methods: Define properties and functions which the respective entities should possess.

Definition of Relationships : Identify associations, dependencies, aggregation, and composition relationships between entities

Refine and Validate: To be sure that the model that one has constructed reveals the actual world domain by refining it along with feedback and validation with stakeholders.

5. **Identification of Entity Objects, Booch's Object Identification Method, Interaction Modelling, Class-Responsibility-Collaborator (CRC)**

Ans:

Identify Entity Objects

Identify Entity Objects is an important thing in object-oriented analysis. An entity object represents key elements within the system's domain. The real-world object or concept usually has a distinct identity, encapsulates related data and behavior that usually reflects some domain. Examples of domain entities in a library management system are the entities Book, Member, and Loan. In the object identification process, a thorough examination of all the requirements, use cases, and the domain model is made to glean nouns and concepts that may express potential objects. Entity objects then become the foundation from which to build a robust class structure and even serve as the nuclei from which relationships in the system may be structured.

Booch's Object Identification Method

With an object-oriented design pioneer such as Grady Booch, one widely adopted method to define the objects is using systematic analysis of the problem space. His method decomposes the system into manageable parts and studies those parts in coming up with candidate objects. This includes the general steps in this method:

1. Identify Classes and Objects: Analyze the requirements document in finding nouns and noun phrases that may also represent classes or objects.

2. Class vs. Instance Explain which terms label classes (e.g. Customer) and which denote instances of unique items (e.g. a specific customer).

3. Determine Class Attributes and Methods Determine the primary distinguishing features and operations that each object should possess.

4. Define Relationships Determine how objects communicate with each other, including inheritance or dependency.

Booch's approach is iterative in that it continues to evolve as one gains insight into the system.

Interaction Modelling

Interaction modeling is one of the major aspects of OOAD, which deals with how the objects in a system collaborate to achieve use cases or some given system behavior. Such modeling helps define how messages are exchanged between objects and where responsibilities lie in achieving the desired outcome. Object interactions are usually presented graphically using UML sequence diagrams and collaboration diagrams.

- Sequence Diagrams These detail the way objects interact over time, ordered by when the messages have been exchanged. This is the sequence of messages and the order in which they are received/ sent. For example, in an online shopping site, the sequence diagram for the Place Order use case involves interactions between a Customer, an Order, an Inventory object and possibly a Payment object.

These are also known as communication diagrams. They detail how objects relate and interact with a system from a relationship-centric view, rather than the sequence of interactions.

Class-Responsibility-Collaborator (CRC) Cards

Class-Responsibility-Collaborator cards are an extremely lightweight mechanism for brainstorming and to model classes in an object-oriented system. Each CRC card refers to a class and has three primary components:

1. Class Name: The name of the class or entity being modeled-for example, Order.

2. Responsibilities: Most important work or actions that this class should do. Complete responsibility that comes out of the use cases of a system. Actual role that the class will be playing within a component of the system will get.

3. Collaborators: Other classes or objects with whom this class collaborates to achieve his responsibilities. For example, an order class might collaborate with a customer class and an inventory class.

CRC cards are commonly used in teams where all the members discuss and validate the roles and how classes interact with each other. Hence this methodology helps develop collaborative design and ensure that class responsibilities are properly allocated such that any system design is more maintainable and flexible.

Advantages of using CRC Cards:

•Simplifies Initial Design: Teams can focus on one class, making it easier to conceptualize responsibilities and interactions.

•Enhances Collaboration: The method promotes communication from team members toward a dynamic, inclusive design process.

•Supports Iterative Development: CRC cards are aligned with iterative development whereby classes and their interactions can evolve as the understanding of the system improves.

Booch's method, coupled with the identification of entity objects, interaction modeling, and CRC cards, lets developers work with a much more elaborate and focused toolkit for structuring and modeling object-oriented systems effectively.