Object-Oriented analysis: Overview

This topic describes the process of creating an object-oriented analysis (OOA) model for software development projects. The OOA model is composed of graphical or text-based representations that define class attributes, relationships, behaviors, and inter-class communication. OOA begins with scenario-based descriptions (use cases) of how actors (people, machines, systems) in the problem space interact with the product to be built. Class-Responsibility-Collaborator modeling translates use-case information into representations of classes and their collaborations. An object-relationship model can be built from the collaborator network. The object-behavior model is represented using a state transition diagram. The OOA model needs to be reviewed for quality like any other software engineering product.

OOA Tasks

1. Basic user requirements must be communicated between the customer and the software engineer.
2. Classes must be identified (e.g. define attributes and methods)
3. Specify class hierarchy
4. Represent object-to-object relationships
5. Model object behavior
6. Reapply 1 through 5 iteratively until model is complete

OOA Generic Steps

- Elicit customer requirements for system
- Identify scenarios or use cases
- Select classes and objects using basic requirements as a guide
- Identify attributes and operations for each system object
- Define structures and hierarchies that organize classes
- Build object-relationship model
- Build object-behavior model
- Review OOA model against use-cases (scenarios)

Unified Modeling Language Perspectives

- User model view (describes usage scenarios from the end-user's perspective)
- Structural model view (static structure of data and functionality is modeled - classes, object, relationships)
- Behavioral model view (represents dynamic system aspects - interactions or collaborations between structural elements in the user and structural models)
- Implementation model view (representing the structural and behavioral aspects of the system as they are to be built)
- Environment model view (representation of the structural and behavioral aspects of the environment in which the system will be implemented)

Domain Analysis Activities

- Define the domain to be investigated
- Categorize the items extracted from the domain
- Collect a representative sample of applications in the domain
- Analyze each application in the sample

- Identify candidate reusable objects
- Indicate reasons the objects may be reused
- Define adaptations of the objects that may be reused
- Estimate percentage of applications in the domain that might make reuse of the objects

- Identify objects by name and use configuration management techniques to control them

- Develop an analysis model for the objects

OOA Model Generic Components

- Static view of semantic classes (classes based on semantics of customer requirements)
- Static view of attributes (attributes describe classes and suggest operations relevant to classes)
- Static view of relationships (represent relationships in a way that allows identification of operations and the design of a messaging approach)
- Static view of behaviors (behaviors accommodating system usage scenarios implemented by sequences of operations)
- Dynamic view of communication (among objects based on events that cause state transitions)
- Dynamic view of control and time (describe the nature and timing of events causing state transitions)

Use Case Objectives

- Define the functional and operational requirements of system by defining a scenario of usage agreed upon by the end-user and software engineer
- Provide an unambiguous description of how the end-user and system interact with one another
- Provide a basis for validation testing

Class-Responsibility-Collaborator (CRC) Modeling

- Develop a set of index cards that represent the system classes
- One class per card
- Cards are divide into three sections (class name, class responsibilities, class collaborators)
- Once a complete CRC card set is developed it is reviewed examining the usage scenarios

Criteria for Inclusion of a Class on a CRC Card

- Class information should be retained
- Provides needed services
- Contains multiple attributes
- Common set of attributes apply to all object occurrences
- Common set of operations apply to all object occurrences
- External entities that produce or consume information

Allocating Responsibilities to Classes

- Distribute system intelligence evenly
- State each responsibility as generally as possible
- Information and its related behaviors should reside within the same class
- Localize all information about one thing in a single class
- Share responsibilities among related classes when appropriate

Collaborators

- Any time a class cannot fulfill a responsibility on its own it needs to interact with another class

- A server object interacts with a client object to fulfill some responsibility

Reviewing CRC Models

- Each review participant is given a subset of the CRC cards (collaborating cards must be separated)
- All use-case scenarios and use-case diagrams should be organized into categories
- Review leader chooses a use-case scenario and begins reading it out loud
- Each time a named object is read a token is passed to the reviewer holding the object's card
- When the reviewer receives the token, he or she is asked to describe the responsibilities listed on the card
- The group determines whether one of the responsibilities on the card satisfy the use-case requirement or not
- If the responsibilities and collaborations on the index card cannot accommodate the use-case requirements then modifications need to be made to the card set

Deriving the Object-Relationship Model

- Using the CRC model a network of collaborators can be drawn
- Reviewing the CRC model index card, responsibilities and collaborators are evaluated, each unlabeled connected line is named
- Once the named relationships are established each end is evaluated to determine cardinality (0 to 1, 1 to 1, 0 to many, 1 to many)
- Continue the process until a complete object-relationship model has been produced

Object-Behavior Model Construction

- Evaluate all use-cases to understand the sequence of interaction within the system
- Identify events that drive the interaction sequence and how events relate to specific objects
- Create an event-trace for each use-case
- Build a state transition diagram for the system
- Review the object behavior model to verify accuracy and consistency

Object-Oriented design: Overview

This topic discusses the steps required to develop an object-oriented software design from an object-oriented analysis model. Object-oriented design (OOD) is divided into two major activities: system design and object design. System design defines the product architecture (the system functions and classes encapsulated in the subsystems). System design focuses on the specification of three components: the user interface, data management functions, and task management facilities. Object design focuses on the internal details of the individual classes and the messaging scheme. The OOD projects must be reviewed to ensure quality.

Object-Oriented Design Layers

- Responsibilities layer (highest layer - contains data structure detail and algorithmic detail for each object's attributes and operations)
- Message layer (establishes the internal and external interfaces for the system, including the details of communication among object collaborators)
- Class and object layer (contains class hierarchy including representations of each object)
- Subsystem layer (lowest level - contains representations of each of the subsystems and the necessary infrastructure that enable the software to achieve the customer's requirements)

Object-oriented Design Issues

- Decomposability - facility of design method that allows the designer to decompose the problem into easily solved subproblems
- Composability - degree to which design method ensures that modules constructed for one project can be reused in another
- Understandablity - ease with which a component can be understood without examining other components
- Continuity - ability to isolate changes made in one module and restrict the propagation of changes to other modules
- Protection - architectural characteristic that reduces the propagation of side effects when errors occur

Generic Object-Oriented Design Steps

- Describe each subsystem and allocate it to processors and tasks
- Choose a design strategy for implementing data management, interface support, and task management
- Design an appropriate system control mechanism
- Perform object design by creating a procedural representation for each operation and data structures for each attribute
- Perform message design using collaborations between objects and object-relationships
- Create a messaging model
- Review the design model and iterate as required

Unified Approach to Object-Oriented Design

- System design - UML (unified modeling language) design activity that focuses on software architecture and definition of subsystems
- Object design - UML design activity that focuses on describing object and their interactions at a level of detail that will allow them to be implemented in some programming language

Object-Oriented System Design Process

- Partition the analysis model into subsystems

- subsystems should have well defined communication interfaces
- with few exceptions classes should collaborate within their subsystem
- keep number of subsystems small
- partition subsystem internally to reduce complexity

- Identify concurrency dictated by the problem
- Allocate subsystems to processors and tasks

- allocate each subsystem to an independent processor (or)
- allocate subsystems to same processor and provide concurrency support through operating system features

- Develop user interface design
- Choose basic strategy for implementing data management

- management of data critical to the application itself
- creation of infrastructure for storage and retrieval of objects

- Identify global resources and control mechanisms to access them

- Design control mechanism for system (including task management)
- Consider how subsystem boundary conditions should be handled

- list each request (contract) that can be made by subsystem collaborators
- for each contract note the operations required to implement the responsibilities implied by the contract
- for each contract create a table with these entries: type, collaborators, class, operation, message format
- if subsystem interaction modes are complex create a subsystem-collaboration diagram

- Review and consider trade-offs

Object Design Process

- Object descriptions

- protocol description - object interface specified by defining each message an object can receive and the operation triggered by message (or)
- implementation description - shows implementation details for each operation implied a message passed to the object

- Designing algorithms and data structures

- algorithm categories: data manipulation, computation, monitors
- refinement of operation programs defined during OOA

- Design optimization

- review object-relationship model to ensure implemented design leads to efficient resource utilization, add redundancy where necessary
- revise attribute data structures and related operation algorithms to improve processing efficiency
- create attributes to save derived information and avoid recomputation

- Modularity is important aspect of object-oriented design quality (the program design language should support object definition)

Design Pattern Specification Components

- Name
- Intent
- Design forces motivating the pattern
- Solution that mitigates these design forces
- Classes required to implement the solution
- Responsibilities and collaborations among the solution classes
- Implementation suggestions
- Source code examples or templates
- Cross-references to related design patterns

Using Design Patterns in Object-Oriented Design

- Inheritance - makes use of an existing design pattern to create a template for a new subclass
- Composition - assembling complex objects or subsystems out of selected design patterns using only interface information

REFER TO CHAPTER 21 and 22 *"Pressman. R. S., Software Engineering a practitioners Approach. 5th Edition"* ACCORDINGLY.