Object-Oriented Analysis and Design (OOAD)

IMSE1013
Henry Lau

Object-Oriented Methods
- A technique for system modelling
- A technique to manage complexity inherent in analysis, design, and implementation
- For the analysis and design of system
- Provide integrated view of hardware and software
- Provide a methodology for system development

Object-Orientation?
- What is Object-Orientation & Object-Oriented (OO) Methods? (C++, Java, Oracle? etc.)
- What is OO methods used for?
- What are the features of OO methods?
- How are they different from SSADM?

How?
“Using object-orientation as a base, we model a system as a number of objects that interacts.”
Is It Any Good?

A system which is designed and modelled using an object-oriented technology is:

- Easy to understand
- Directly related to reality - semantic gap
- Natural partitioning of the problem
- More flexible and resilient to change
- Systems can be developed more rapidly and at a lower cost

Some Qualities of OO

- **Understanding** of system is enhanced, as the semantic gap is reduced
- **Modification** to the model tend to be local as they often result from an individual item, which is represented by a single object
- Ideally suited to model real systems, and simulating systems

Some Examples of Using OO

- Object technology is key to re-engineering business process at Xerox
- Space telescope uses OO technology and Booch Method to build interface to Hubble Database
- British Airways choose OO tools for airline applications

OO Methods

- **Object-Oriented Design (OOD)** - Booch (1983), pioneering but not quite scalable
- **Object-Oriented System Analysis (OOSA)** - Shlaer & Mellor (1988), essentially information analysis based on data modelling
- **Object-Oriented Analysis (OOA)** - Coad & Yourdon (1991), a method for developing OO system model
Object-Oriented Modelling

- Attach the behaviour and information that is important to objects
- Associate relations between objects to describe the static and dynamic organisation and structure of real situations
The Basics
- Objects
- Classes
- Relationships
- An Instance
- Idea of encapsulation

An Object
- Some concept of reality
- A physical entity
- It is characterised by:
  - a number of operations,
  - a state which remembers the effect of these operations

An Object
- Operations:
  - Work
  - Dance
  - Drive
  - Jump
- Attributes:
  - Height
  - Eye colour
  - Hair colour
  - Weight

Relationships
- Static:
  - relations existing over a long time
  - objects know about each other existence
- Dynamic:
  - relations which two objects communicate with each other
  - object sending stimuli to other
  - stimuli - events, messages
Consist of Relation

‘Creating’ Objects
- Composition - structure object from parts
- Partition - into hierarchy (‘is a’)
- Consist of - build objects from others
- Aggregate - to join together (‘has a’)

‘Is a’ Relation

Encapsulation
- A concept of ‘Self-containing’
- Information hiding - ‘internal’ structure is hidden from their surroundings
- Behaviour and information is represented or implemented internally
- Functionality and behaviour characterised by ‘interfacing’ operations
Class
- A class represents a template for several objects and describes how these objects are structured internally.
- Objects of the same class have the same definition both for their operations and their information structure.
- Class is an implementation of objects.

Instance
- An instance is an object created from a class.
- A class describes the behaviour and information structure of an instance, while the current state of the instance is defined by the operations performed on the instance.
- System’s behaviour is performed via the interactions between instances.

Key Concepts
- Polymorphism – same object has different implementations.
- Inheritance – to adopt, permutate, and derive from some generic objects.

Polymorphism
- A concept in type theory.
- A common name may denote instances of different classes.
- One type of operation can be implemented in different ways by different classes.
- Overloading in modern OO language.
Why Polymorphism

- A very strong tool for allowing system designers to develop flexible systems
- Designer only need to specify what shall occur and not how it shall occur
- To add an object, the modification will only affect the new object, not those using it

Inheritance

“If class B inherits class A, then both operations and the information structure described in class A will become part of class B”

Why Inheritance?

- Show similarities
- Reuse common descriptions
- ‘Software Reuse’
- Easy modification of model by performing modification in one place
- Avoid redundancy, leading to smaller and more efficient model, easier to understand
Limitations of SSADM

- Treat data and function separately - function/data oriented method
- More suited to classical hardware
- More difficult to maintain and re-configure
- Method require more abstraction - not too natural
- Large semantic gap between external and internal view of a system

Object-Oriented Methods

- Advocate integral objects which encapsulate both function and data
- Main activities include:
  - Identification of objects, and
  - Analysing their behaviour and information
- Uses object-oriented techniques and ideas:
  - inheritance
  - polymorphism
  - function/data abstraction

Object-Oriented Analysis & Design

1. Finding objects
2. Organising objects
3. Describing how objects interacts
4. Defining the operations of objects
5. Defining objects internally

Finding Objects

- Naturally occurring entities - physical
- A concept of some abstract ideas - conceptual
- Should be stable
- Classes of objects
  - active/passive
  - temporary/permanent/persistent
  - part/whole
  - generic/specific
  - private/public
Object-Oriented Design
- The objects found are to be implemented.
- Once objects for a system are identified, they are refined, organised and related.
- Classes that define the implementation are structured and consolidated.
- Classes are refined with implementation details (e.g. OS, language, hardware, etc.).
- Classes are coded.

Organising Objects
- Classification
- Similar objects - inheritance: ‘is a’
- Interactions/relationships between objects
- Whole/Part relationship: ‘has a’

Object Interactions
- Identify how objects fit into a system.
- Use of scenarios - unique situations.
- Objects’ communication.
- Objects’ interfaces.
- Refined relationships.

Object’s Functionality
- Operations performed by an object.
- Behaviour of an object.
- Specification of interfaces, external and internal functions.
- Objects with complex functionality should be partitioned into simpler objects.
Object Implementation

- The specification of **CLASSES**
- Define information that an object encapsulates - **ATTRIBUTES** and **METHODS**

![Object Diagram](image)

Object Implementation

- **METHODS:**
  - Specify external functions
  - Specify internal functions that are not seen or usable by others objects
- **Languages:** C++, Smalltalk, Ada, Eiffel, Modula-2, Simula, Java++

Testing of Objects

- Similar to SSADM
- Unit/Integration/System tests
- Validation and verification
  1. Cross references
  2. Object diagrams
  3. Simulation of classes

OO Methods - Some Advantages

- Reduce semantic gap between domain (the actual) and model (the design)
- Closer to reality e.g. classification of objects close to how human understand surroundings
- Easier to understand and maintain
- Easier to modify (e.g. polymorphism)
Jacobson Use Case Method (OOSE)

The Jacobson Use Case Method

- An OO methodology that emphasizes on the identification of objects - requirement analysis
- Based on 3 techniques:
  1. Conceptual modeling
  2. Block design
  3. Object-oriented programming

Model Building with JM

Jacobson Method

- The development cycle is organized into five models:
  1. Requirement model
  2. Analysis model
  3. Design model
  4. Implementation model
  5. Test model
System Development

- Requirement (Use Case) Model
  - Structured by
  - Realized by
  - Implemented by
  - Tested in

Classes

- Analysis Model
- Design Model
- Implementation Model
- Test Model

Requirement Model

- Capture the functional and non-functional requirements
- Formalize the requirement
- Identify and document essential system entities
- Identify and document essential system behaviors

Requirement - Use Case Model

Actors:
- Essential system entities from an user view point
- Interacts with system
- Changes system behavior
- Control system functionality
Requirement Model
- System - define the boundary
- Use Cases:
  - System scenarios
  - System behaviors
  - Unique transaction between actors and the system
  - User point of view

Analysis Model
- **Structure** a system independently to the actual implementation
- Capture information, behavior and presentation
- Specify objects

Design Model
- Refine the object structure to the chosen implementation environment
- Objects are consolidated into 'blocks' - abstracted classes
- Blocks interactions are also documented using interaction diagrams

Implementation Model
- The blocks in the design model are implemented using classes
- Class diagrams are used to express relationships between classes
- Class specifications
- Annotated source code (pseudo-code) for methods and attributes of classes
Object Diagrams

Machine Centre A

GUI

1: Load NC Program
2: Start checking
3: Status OK
4: Ready to Operate

Test Model

- States the method and result of testing
- Test specification describe how classes and system are to be tested
- Test results document outcome of the tests executed
- Verification and validation

System Analysis and Design with Jacobson

1. Requirement Analysis
2. Functional Analysis and Design

Requirement Analysis

- Generate requirement model with actors, and use cases identified
- **STEP 1:** System context diagram
- **STEP 2:** Identify principal actors (external and internal)
- **STEP 3:** Construct use cases
- **STEP 4:** Represent actors and use cases with a use case actor diagram
Functional Analysis & Design

- Structure system functionality into objects
- To distribute the behavior of the use cases among objects - analysis model
- Create objects that are common to actors or use cases
- Identify the responsibilities of objects
- Identify the relationships between objects

Analysis and Design

- **STEP 1:** Identify scenarios from use cases
- **STEP 2:** Associate actors and use cases to scenarios
- **STEP 3:** Produce event lists for scenarios
- **STEP 4:** Identify primary object types
- **STEP 5:** Consolidate objects into control, entity and interface
- **STEP 6:** Identify and represent relationships

Unified Modeling Language (UML)

Forces in Software

The challenge over the next 20 years will not be speed or cost or performance; it will be a question of complexity.

Bill Raduchel, Chief Strategy Officer, Sun Microsystems

Our enemy is complexity, and it's our goal to kill it.

Jim Hafner
The Value of UML
- Is an open standard
- Supports the entire software development lifecycle
- Supports diverse applications areas
- Is based on experience and needs of the user community
- Supported by many tools

Overview of the UML
- The UML is a language for
  - visualizing
  - specifying
  - constructing
  - documenting
- A model is a complete description of a system from a particular perspective

Graphical Tools
- Sequence Diagrams
- Use Case Diagrams
- Class Diagrams
- Object Diagrams
- Collaboration Diagrams
- Statechart Diagrams
- Activity Diagrams
- Component Diagrams
- Deployment Diagrams