IF2261 Software Engineering

Introduction to OOSE

Program Studi Teknik Informatika
STEI ITB

This material are adopted from:

Object-Oriented Analysis and Design Methods: a Comparative Review

Authors: Sjaak Brinkkemper, Shuguang Hong, Arjan Bulthuis, Geert van den Goor.

January 1995

OOSE Jacobson

- combines three different techniques:
  - **object-oriented programming**
  - **conceptual modeling** is used to create different models of the system or organization to be analyzed.
  - **block design**, originates from hardware design in the telecommunications area.

General Approach

- OOSE has a so-called 'use case driven approach‘:
  - a use case model serves as a central model
  - a use case model describes the complete functionality of the system
  - the use case model is the basis in the phases analysis, construction and testing.

General Approach (2)

- The aim of analysis is to understand the system according to its functional requirements.
  - The **objects** are **found**, **organized** and **object interactions** are described.
  - The **operations** of objects and the internal view of objects is described as well during analysis.
- Construction encompasses design and implementation in source code.
  - It is important that objects in the analysis phase can be found back during construction. This is called **traceability**.
  - **Components** are important during construction (piece of source code that can be used for implementing objects).
- In testing, the system is verified
  - The correctness of the system is checked according to its specifications

Concepts and Construct

- **Actor:** An actor defines a role that a user can play in exchanging information with the system
- **Primary actor:** An actor who uses the system directly, performing one or some of the main tasks
- **Secondary actor:** An actor who supervises or maintains the system
- **Abstract** actor: An actor that describes a role that should be played against the system. Different actors may inherit from an abstract actor if they similar roles
- **Role:** The role is defined by the operations of an Object. It describes the purpose wherein one Object participates with another
- **User:** A user is the person who actually uses the system. A user is instance of the Actor class
- **Use case**: A use case is a complete course of events specifying all the actions between the user and the system
- **Abstract use case:** A description of commonality in other use cases. An abstract use case will not be instantiated on its own
- **Concrete use case:** A use case that really will be instantiated

Concepts and Construct (2)

- **Object:** An object is characterized by a number of operations and a state which remembers the effect of these operations
- **Entity object:** An object about which information is stored that lasts for a long time, even when a use case is completed
- **Interface object:** An object which contains functionality of the use cases that interacts directly with the environment
- **Control object:** An object that models functionality that is not in any other object, e.g. calculating taxes using several different criteria
- **Central interface object:** An interface object that contains other interface objects
- **Attribute:** An attribute contains information and its type
- **Class:** A group of objects that have similar behavior and information structures
- **Abstract class:** A class that is developed with the main purpose to be inherited by other classes
- **Concrete class:** A class that is developed with the main purpose to create instances of it

Concepts and Construct (3)

- **Stimulus:** An event that is sent from one object to another and that initiates an operation
- **Message:** Intra-process stimulus: a normal call inside one process
- **Signal:** Inter-process stimulus: it is sent between two processes
- **Operation:** An activity inside a block that may lead to a state change of the corresponding object.
- **Subsystem:** A defined group of objects in order to structure the system, subsystem packages the objects to reduce complexity
- **Service package:** The lowest level of a subsystem that is to be viewed as an atomic change unit
- **Block:** Design object, which is an abstraction of the actual implementation. Blocks are later implemented as source code
- **State:** The union of all values describing the present situation
- **Transition:** The change of a state
- **Object module:** The implementation in source code of a block. A block may be implemented in more than one object module
- **Public object module:** An object module that is accessible from the outside of a block
- **Private object module:** An object module that is not accessible from the outside of a block

Relationships between Classes and Objects

- Class association
  - Inherit association
    - The operations and the information structure of a superclass are inherited by the subclasses.
    - A subclass may inherit from more than one superclass (multiple inheritance).
  - Extension association:
    - Extension means inserting one use case description into another use case description that must be a complete course in itself. This description is thus independent of any inserted description.

Relationships between Classes and Objects (2)

- Instances association
  - Association (Relation):
    To model relationships between objects, OOSE mentions the role one object can play in relation to the other object. For example, if there is a relationship between the object 'Car' and the object 'Person' named 'driven by', then 'Person' plays the role of 'driver' in relation to the object 'Car'.
  - Acquaintance association:
    An object is an acquaintance of another object if it knows that the other object exists. It is a static relationship, meaning that the objects cannot exchange information with each other.
  - Consists-of association:
    This is a special type of acquaintance association denoting that is composed of other objects. The term aggregation is also used.

Relationships between Classes and Objects (3)

- Communication association:
  Communication associations are used to model communication between objects. Through communication associations objects send and receive stimuli. The arrow denotes the direction of a stimulus.

Operations and communication

- Operations of an entity object may include:
  - creating and deleting the entity object
  - storing and fetching information
  - behavior that must be changed if the entity object is changed
- Objects communicate by sending each other stimuli.
  - A stimulus causes an operation to be performed in the receiving object.
  - A stimulus can either be a message, denoting synchronous intraprocess communication, or a signal, denoting synchronous or asynchronous interprocess communication.

Techniques : Requirement model

- This model **delimits the system** and **defines its functionality**.
  It consists of three parts:
  - **Use case model**; describes actors and use cases
  - **Problem domain object model** for developing a logical view of the system
  - **Interface descriptions**

Techniques : Requirement model (2)

- Use case model
  - describes actors and use cases.
    - Actors define roles that users can play in exchanging information with the system and
    - use cases represent functionality inside the system.
  - associations between use cases:
    - *extends*, which defines alternative course of events:
      - optional parts of use cases
      - complex and alternative courses which seldom occur
      - separate sub-courses which are executed only in certain cases
      - situation where several different use case can be inserted into a special use case
    - *uses*, which identifies a use case as a part of another use case definition

Techniques : Requirement model (3)

- Problem domain object model
  - The model is the **identification of the objects** that have a direct counterpart in the application environment and that the system must know about.
  - The refinement of the problem domain objects can be obtained in different possible degrees:
    - *object name*
    - *logical attributes*
    - *static instances associations*
    - *inheritance*
    - *dynamic instance associations*
    - *operations*

Techniques : Requirement model (4)

- Interface descriptions
  - The interface has to **capture the user's logical view of the system**, because the main interest is the consistency of this logical view and the actual behavior of the system.
  - It can deal with both **user interfaces** and **interfaces with other systems** by creating **sketches** or **prototypes** of what the user will see on the screen when performing the use case.
  - It is important that users are involved in making detailed interface descriptions.

Techniques : Analysis model

- **The analysis model** structures the system (the requirements model) by modeling three types of objects:
  - interface objects,
  - entity objects and
  - control objects.
- The **behavior** that is modeled in the use cases is **spread among the objects** in the analysis model.
- The model is used to provide the system with **a robust and changeable object structures**. These objects are structurally placed and connected to other objects or actors to describe the object-object or object-actor relationship.
- The analysis model provides a **foundation for design**.

Techniques : Design model

- **The design model** will **refine** the analysis model and will **adapt** it **to the implementation environment**.
- Interfaces of objects and semantics of operations are defined and decisions can be made about:
  - Database Management Systems (DBMS) and
  - programming languages.
- Blocks are introduced for object types to hide (the abstraction of) the actual implementation.
- The design model consists of:
  - interaction diagrams and
  - state transition graphs.

Techniques : Design model (2)

- *An interaction diagram*
  - *An interaction diagram* is drawn **for each concrete use case**.
  - It describes the use cases **in terms of communicating objects**. This communication is modeled as blocks sending stimuli to each other.
  - Interaction diagrams support use cases with extensions.
    - In this case, probe positions are added to an interaction diagram. A probe position indicates a position in the use case that is to be extended and often a condition is required for when the extension is allowed to take place.
  - In parallel design process, several stimuli with the same purpose or meaning are defined by several designers. Therefore these stimuli should be consolidated to obtain as few stimuli as possible. This process is called *homogenization*.

Techniques : Design model (3)

- There are two structure types of interaction diagrams:
  - *Fork* which indicates a centralized structure and is characterized by the fact that it is an object controls the other objects interacted with it. This structure is appropriate when:
    - The operations can change order
    - New operations could be inserted
  - *Stair* which indicates decentralized structure and is characterized by delegated responsibility. Each object only knows a few of the other objects and knows which objects can help with a specific behavior. This structure is appropriate when:
    - The operation have a strong connection. Strong connection exists if the objects:
      - form a 'consist-of' hierarchy
      - form an information hierarchy
      - form a fixed temporal relationship
      - form a (conceptual) inheritance relationship
    - The operation will always be performed in the same order

Techniques : Design model (4)

- *State transition graphs*
  - used to describe object behavior in terms of which stimuli can be received and what stimuli may cause.
  - uses an extension of the SDL notation (Specification and Description Language which is a CCITT standard).

Techniques : Implementation model

- Consists of **the source code** of the specified objects in the design model.
- It is desirable that a block can be easily translated into the actual object module.
- In a smooth implementation environment, this is typically the case.

Techniques : Test model

- produced by testing the implementation model.
- **Test specification**, which is the test's class when a test is seen as an object, and test result, the execution of an instance from the test's class, are the main concepts.
  - the lowest level of the system is tested (e.g. object modules and blocks),
  - use cases can be tested and
  - a test can be performed on the whole system.
- The requirements model can serve as a verification for the test model.

Activities : Analysis

- Two models are yielded:
  - the requirements model and
  - the analysis model.
- Requirement analysis activity produces the requirement model that describes:
  - all the functional requirements from user perspective
  - the way the system is to be used by end users
- The requirements model:
  - structured into an analysis model in robustness analysis activity.
  - structured from a logical perspective into a robust and maintainable system.

Activities : Analysis (2)

- The use case identified in the requirement model is partitioned according to the following principles:
  - Functionalities which are directly dependent on the system's environment are placed in *interface objects*
  - Functionalities dealing with the storage and handling or object are placed in the *entity object*
  - Functionalities specific to one or a few use cases and not naturally placed in any of the other objects are placed in *control objects*.
- Objects identified may also be partitioned using subsystem.
- One use case may relate to several subsystems.
- Use cases can be viewed as a dynamic behaviour partitioning of the system, while subsystems are static partitioning of the system.
- OOSE does not explicitly describe steps of how these two models can be obtained.

Activities : Construction

- **Design (Activity 2.1)**
  - the implementation environment has to be identified (Activity 2.1.1). The consequences that the implementation environment will have on design are studied. The identification and investigation of the implementation environment results in strategic implementation decisions.
  - a first approach to a design model is developed (Activity 2.1.2) in which the analysis objects are translated into design objects fitting in the implementation environment.
    At first, the analysis model are transformed directly into design objects called blocks.
    Then, to achieve maximum robustness changes can and should occur in various way:
    - Introduce new blocks in the design model which do not have any representation in the analysis model
    - Delete blocks from the design model
    - Change blocks in the design model (splitting and joining existing block)
    - Change the associations between the blocks in the design model

Activities : Construction (2)

- the interaction of objects (i.e. the stimuli) in each use case is described (Activity 2.1.3),
  - resulting in object interfaces or even new supporting object definitions (classes).
  - These object interfaces may then be refined by looking at the internals of the object using state transition graphs to increase the understanding of the block before going into details.

Activities : Construction (3)

- **Implementation (Activity 2.2)**
  - each object is implemented in the programming language (Activity 2.2.1).
- there is a subprocess called the *component development process*, which develops and maintains components to be (re)used.

Activities : Testing

- Building of test planning (Activity 3.1) in which it is examined:
  - whether existing test programs and data can be used,
  - whether they can be modified or if new development has to take place.
  - decisions about subtests can be made, e.g. concerning standards for testing and required resources for the subtests.
  - information throughout the test process is stored in a test log, which serves also as the basis for further refinement of the test process and the planning of new tests.
- tests are identified (Activity 3.2).
  - what should be tested and
  - how much resources are needed approximately.
- specify the tests (Activity 3.3).
  - brings out an overview description of the test and its purpose.
  - a detailed procedural description of each step is made in the test.

Activities : Testing (2)

- the tests are performed (Activity 3.4), each use case one by one.
  - A decision table is used to store the test results of each subtest.
  - A test result is called an outcome and can be 1 for OK and 0 for fail.
  - the overall test can be measured by adding weight factors to subtests according to their importance,
  - the overall test can be compared to a limit.
    If the weighted sum of outcomes of the subtests exceeds the limit, then the test is approved.
- In case the test failed, the failure is analyzed (Activity 3.5).
- After analysis, tests are respecified and performed again until the test approves. In that case the test is finished (Activity 3.6).
  - The equipment and test bed should be restored as well as the documentation made during preparation until the completion of the test.
  - The test log is filed and can be used in next tests.

Remark

- The Object-oriented Software Engineering (OOSE) method considers a full range of the development stages in software engineering, from requirement gathering to testing.
  - Jacobson et. al.treat maintenance stage as another full life cycle of system development.
  - The management aspects has also been introduced, even though it is still very limited.
- The idea of using use case as a base for the whole development process is a breakthrough in the world of object-oriented.
  - This technique is simple yet a very convenient way of describing the functional requirement of a system.

Remark (2)

- The reuse aspect in developing the system has also been introduced and is represented as a full process (component management) together with its heuristic to support the Construction Process.
  - the component concept in this method is not as sophisticated as the reuse concept defined nowadays.
  - the component concept does not distinguish types and granularity of reuse.

Remark (3)

- The notation used in this method is basically simple, except for the state transition graphs, but they use different notation in different model to represent objects.
  - Although the transformation between model (or pragmatic aspect) of the method is provided, the use of these notations in this manner may confuse the user.