Software Development Methodologies

Lecturer: Raman Ramsin

Lecture 3

Seminal Object-Oriented Methodologies: A Feature-Focused Review
Responsibility-Driven Design (RDD)

- Introduced in 1990; a UML-based and use-case driven version was released in 2002.

- The process starts when a detailed requirements specification of the system has already been provided.

- The system is modeled as a collection of objects that collaborate to fulfill their *responsibilities*.

- Responsibilities include two key items:
  - The knowledge an object maintains.
  - The actions an object can perform.
RDD: Process

- The process is divided into two phases:
  - *Exploratory Phase*: classes and their responsibilities and collaborations are identified.
  - *Analysis Phase*: subsystems and class details are specified.

![RDD Process Diagram]

[Wirfs-Brock et al. 1990]
RDD: Exploratory Phase

- The major tasks to be iteratively performed in this phase are:
  - Discovering the *classes* required to model the application
  - Determining what behavior the system is responsible for and assign these *responsibilities* to specific classes
  - Determining what *collaborations* must occur between classes of objects to fulfill the responsibilities.

- Results are modeled in CRC (Class-Responsibility-Collaborator) cards

**CRC Card**

<table>
<thead>
<tr>
<th>Class: name of class</th>
<th>(Abstract or Concrete)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Superclasses:</strong> list of superclasses</td>
<td></td>
</tr>
<tr>
<td><strong>Subclasses:</strong> list of subclasses</td>
<td></td>
</tr>
<tr>
<td><strong>Responsibilities:</strong></td>
<td><strong>Collaborations:</strong></td>
</tr>
</tbody>
</table>

[Wirfs-Brock et al. 1990]
RDD: Analysis Phase

- Major tasks to be performed in this phase:
  - Factoring the responsibilities into inheritance hierarchies modeled in *Inheritance Graphs*, and identifying class contracts
  - Identifying possible subsystems of objects and modeling them and their client-server relationships in *Collaboration Graphs*
  - Determining *Class Protocols* (method signatures) and completing a specification of classes, subsystems of classes, and client-server contracts
RDD: Analysis Phase

Collaboration Graph

[Wirfs-Brock et al. 1990]
Booch

- Introduced, purely as a design method, in 1991 [Boo91]; extended version, which also covered analysis, was introduced in 1994.

- Process designed as a repeating process (referred to as “The Micro Process”) within a lifecycle-level repeating process (referred to as “The Macro Process”).

- The macro process serves as a controlling framework for the micro process.
Booch: Macro Process

- Represents the activities of the development team on the scale of weeks to months.
- Many parts of this process are basic software management practices such as quality assurance, code walkthroughs, and documentation.
- The focus at this level is more upon the customers and their desires for things such as quality, completeness, and scheduling.

[Booch 1994]
Booch: Micro Process

- Driven by scenarios and architectural specifications that emerge from the macro process
- Represents the daily activities of the individual or small group of developers

[Booch 1994]
Object Modeling Technique (OMT)

- Introduced by Rumbaugh et al. in 1991

- Categorized as *combinative* [MP92], since it is based on three different models and defines a method for integrating them.

- **Object Model (OM):** depicts object classes in the system and their relationships, as well as their attributes and operations, in a *Class Diagram*

- **Dynamic Model (DM):** indicates the dynamics of the objects, their changes in state and the flow of events; captured in *Event-Trace Diagrams* and *State Transition Diagrams (State Charts)*

- **Functional Model (FM):** a hierarchical set of *Data Flow Diagrams (DFDs)* of the system
OMT Process

Knowledge about application domain
Problem statement
User interactions

Init. OM
Init. DM
Init. FM

Init. OM
Init. DM
Init. FM

OM, DM, FM

OM, DM, FM

OM, DM, FM

Object design document
System design document

Object design document
Source code
Test scenarios

Class libraries

Reuse Database of OMT Specifications

[1991 Rumbaugh et al.]

Department of Computer Engineering

Sharif University of Technology
OMT Process: Analysis Phase

- The goal is to build a correct and comprehensible model of the real world.

- Once the initial problem is defined, the following tasks are carried out:
  - Building the object model, including a *Class Diagram* and a *Data Dictionary*.
  - Developing the dynamic model, including *State Transition Diagrams* and global *Event-Trace Diagrams*.
  - Constructing the functional model including *Data Flow Diagrams* and constraints.
  - Verifying, iterating, and refining the three models.
OMT Process: Analysis Phase

Class Diagram

Person
- name
- address
- social security no.
- charge-time
- earn-salary

Worker

Manager

Project
- product name
- budget
- priority

Company
- name
- address
- phone number
- primary product
- hire
- fire

Department
- dept name

Product
- product name
- budget
- priority

employee

Works-for

employer

worker-type

Manages

Responsible-for

Manages

Manufactures

[Rumbaugh et al. 1991]
OMT Process: Analysis Phase

Event-Trace Diagram

[Event-Trace Diagram showing the interaction between Caller, Phone Line, and Callee with events like caller lifts receiver, dial tone begins, dials, dial tone ends, dials digit, ringing tone, phone rings, answers phone, tone stops, ringing stops, phones connected, phones connected, connection broken, callee hangs up, connection broken, caller hangs up.]

[Rumbaugh et al. 1991]
OMT Process: Analysis Phase

State Transition Diagram

[Rumbaugh et al. 1991]
OMT Process: Analysis Phase

Data Flow Diagram

[Figure showing a data flow diagram with nodes such as Icon definitions, Window, expand into vectors, clip vectors, offset vectors, Screen buffer, and operations like location, size, and pixel operations.]

[Rumbaugh et al. 1991]
OMT Process: System Design

- High-level structure of the system is defined

- Tasks include:
  - Organizing the system into subsystems
  - Identifying concurrency
  - Allocating subsystems to processors and tasks
  - Choosing the strategy for implementing data stores in terms of data structures, files, and databases
  - Identifying global resources and determining mechanisms for controlling access to them
  - Choosing an approach to implementing software control
  - Considering boundary conditions
  - Establishing trade-off priorities
OMT Process: Object Design

- Concerned with fully specifying the existing and remaining classes, associations, attributes, and operations necessary for implementing the system.

- Operations and data structures are fully defined along with any internal objects needed for implementation.

- All details for fully determining how the system will be implemented are specified.
Object-Oriented Software Engineering (OOSE)

- First introduced by Jacobson et al. in 1992
- A simplified version of Jacobson’s Objectory methodology, first introduced in 1987 and later the property of Rational Corporation (now acquired by IBM)
- Covers the full generic lifecycle
OOSE: Process

- **Analysis**: focusing on understanding the system and creating a conceptual model. Consists of two non-sequential, iterative subphases:
  - Requirements Analysis, aiming at eliciting and modeling the requirements of the system. A Requirements Model is produced.
  - Robustness Analysis, aiming at modeling the structure of the system. An Analysis Model is produced.

- **Construction**: focusing on creating a blueprint of the software and producing the code. Consists of two subphases:
  - Design, aiming at modeling the run-time structure of the system, and also the inter-object and intra-object behaviour. A Design Model is produced.
  - Implementation, aiming at building the software. An Implementation Model (including the code) is produced.

- **Testing**: focusing on verifying and validating the implemented system. A Test Model is produced.
OOSE: Process

Analysis
- Requirements analysis
  - use case modeling
  - user interface design
  - domain modeling
  - robustness analysis
  - three object types

Analysis Model
- Modeling the system with three object types
- Subsystems

Requirements Model
- Use case model
- Interface descriptions
- A domain model

Design Model
- Implementation environment descriptions
- Interaction diagrams
- State transition graphs
- Block design
- An object model

Construction
- Design
  - implementation environment
- Implementation

Implementation Model
- Source code

Testing
- Unit testing
- Integration testing
- System testing

Testing Model
- Test specifications
- Test results

[Jacobson et al. 1992]
OOSE: Analysis – Requirements Analysis

- **Aim:** Specify and model the functionality required of the system, typical means and forms of interacting with the system, and the structure of the problem domain.

- The model to be developed is the *Requirements Model*, further divided into three submodels:
  
  - A *Use Case Model*: which delimits the system and describes the functional requirements from the user’s perspective.
  
  - A *Domain Object Model*: consists of objects representing entities derived from the problem domain, and their *inheritance*, *aggregation* and *association* relationships.
  
  - *Interface Descriptions*: provide detailed logical specifications of the user interface and interfaces with other systems.
OOSE: Analysis - Requirements Model

Use Case Model

[Jacobson et al. 1992]
OOSE: Analysis - Requirements Model

Domain Object Model

- Customer
- Group
- Individual Customer

Relationships:
- Customer inherits Group
- Customer contains [0..N] Customer Base

Inheritance:
- Customer inherits Individual Customer

[ Jacobson et al. 1992 ]
OOSE: Analysis – Robustness Analysis

- **Aim**: Map the *Requirements Model* to a logical configuration of the system that is robust and adaptable to change.

- **The model to be developed is the *Analysis Model***:
  - Shows how the functionality of each and every use case is realized by collaboration among typed objects (called *Analysis Objects*).
  - Shows the subsystems of the system.

- **Analysis Objects** can be of three types:
  - *Entity*: Represent entities with persistent state, typically outliving the use cases they help realize. They are usually derived from the domain object model.
  - *Interface*: Represent entities that manage transactions between the system and the actors in the outside world.
  - *Control*: Represent functionality not inherently belonging to other types of objects. They typically act as controllers or coordinators of the processing going on in the use cases.
OOSE: Analysis - Analysis Model

Analysis Model

[Jacobson et al. 1992]
OOSE: Construction - Design

- **Aim:** Refine the *Analysis Model* by taking into account implementation features.

- The model to be developed is the *Design Model*:
  - describes the features of the implementation environment
  - describes the details of the design classes (referred to as *blocks*) necessary to implement the system
  - describes the way run-time objects should behave and interact in order to realize the use cases
OOSE: Construction - Design

- Three Subphases:

1. Determination of the features of the implementation environment (DBMS, programming language features, distribution considerations, ...)

2. Definition of *blocks* (design classes) and their structure:
   1. Each object in the Analysis Model is mapped to a *block*.
   2. Implementation-specific blocks are added and the collection is revised.
   3. Interfaces and semantics of operations are defined.

3. Specification of the sequences of interactions among objects and the dynamic behaviour of each block:
   1. An *Interaction Diagram* is drawn for each of the use cases.
   2. A *State Transition Graph* is used for describing the behaviour of each block.
OOSE: Construction - Design Model

Interaction Diagram

[ Jacobson et al. 1992 ]
Aim: Produce the code from the specifications of the packages and blocks defined in the design model.

The model to be developed is the *Implementation Model*, which consists of the actual source code and accompanying documentation.
OOSE: Testing

- Aim: Verify and validate the implementation model

- The model to be developed is the *Testing Model*, which mainly consists of:
  - Test plan
  - Test specifications
  - Test results.

- Testing is done at three levels, starting from the lowest level:
  - Blocks are tested first
  - Use cases are tested next
  - Finally, tests are performed on the whole system
OOSE: Pivotal role of the Use Case Model

[Jacobson et al. 1992]
References


