Object Oriented Software Engineering

Shahela Saif

COURSE OUTLINE

COURSE LEARNING OUTCOMES

TEXTBOOK(S)

Textbook(s):

Object Oriented Software Engineering: Using UML, Patterns, and Java, Bernd Bruegge, Allen H. Dutoit, Prentice Hall, 2010.

Reference Book(s):

Object Oriented Software Engineering, Singh, Yogesh Malhotra, Ruchika, PHI Learning, 2012.

Object Oriented Software Construction, Bertrand Meyer, 2nd Edition, Prentice Hall, in 1997

CONTACT:

Office:

- First Floor, CS Dept, Academic Block II

Email:

- shahela.saif@comsats.edu.pk

INTRODUCTION TO SOFTWARE ENGINEERING

SOFTWARE ENGINEERING

Software engineering is the application of engineering principles to the development of software solutions

WHAT IS SOFTWARE ENGINEERING

From modeling perspective software engineering include following activities

- Modeling
- Problem Solving
- Knowledge Acquisition
- Rationale Driven

1. MODELING

A **model** is an abstract representation of a system that enables us to answer questions about the system.

Models are useful when dealing with systems that are too large, too small, too complicated, or too expensive to experience firsthand.

Models also allow us to visualize and understand systems that either no longer exist or that are only claimed to exist.

1.2 HOW SOFTWARE ENGINEER DOES IT

First, software engineers need to understand the environment in which the system has to operate.

- For a stock trading system, software engineers need to know trading rules.
- They only need to learn the application domain concepts that are *relevant* to the system. In other terms, they need to build a model of the application domain.

Second, software engineers need to understand the systems they could build, to evaluate different solutions and trade-offs.

- Most systems are too complex to be understood by any one person, and most systems are expensive to build.
- To address these challenges, software engineers describe important aspects of the alternative systems they investigate.
- In other terms, they need to build a model of the **solution domain**.

1.3 ROLE OF OBJECT-ORIENTED METHODS

Object-oriented methods combine the application domain and solution domain modeling activities into one.

The application domain is first modeled as a set of objects and relationships.

- A stock trading system includes transaction objects representing the buying and selling of supplies.

Then, solution domain concepts are also modeled as objects.

- A financial transaction are objects that are part of the solution domain.

The idea of object-oriented methods is that the solution domain model is a transformation of the application domain model.

2. PROBLEM SOLVING

Software engineering is an engineering activity.

Engineering is a **problem-solving** activity

1. Formulate the problem.
2. Analyze the problem.
3. Search for solutions.
4. Decide on the appropriate solution.
5. Specify the solution

2.1MAPPING SOFTWARE ENGINEERING ACTIVITIES WITH ENGINEERING ONE

Object-oriented software development typically includes six development activities:

Requirements elicitation, and Analysis

- During requirements elicitation and analysis, software engineers formulate the problem with the client and build the application domain model.

System design,

- During system design, software engineers analyze the problem, break it down into smaller pieces, and select general strategies for designing the system.

Object design,

- During object design, they select detail solutions for each piece and decide on the most appropriate solution.

Implementation, and Testing.

- During implementation, software engineers realize the system by translating the solution domain model into an executable representation.

3. KNOWLEDGE ACQUISITION

Knowledge acquisition is a nonlinear process.

- The addition of a new piece of information may invalidate all the knowledge we have acquired for the understanding of a system.

Risk-based development attempts to anticipate surprises late in a project by identifying the high-risk components.

4. Rationale

For software engineers, Assumptions that developers make about a system change constantly.

- Even though the application domain models eventually stabilize once developers acquire an adequate understanding of the problem, the solution domain models are in constant change.
- Design and implementation faults discovered during testing and usability problems discovered during user evaluation trigger changes to the solution models.
- Changes can also be caused by new technology.

In order to deal with changing systems, however, software engineers must address the challenges of capturing and accessing rationale.

SOFTWARE ENGINEERING CONCEPTS

SOFTWARE ENGINEERING CONCEPTS SHOWED AS A UML CLASS DIAGRAM

SOFTWARE ENGINEERING DEVELOPMENT ACTIVITIES

- Requirements Elicitation
- Analysis
- System Design
- Object Design
- Implementation
- Testing

I. REQUIREMENTS ELICITATION

During **requirements elicitation**, the client and developers define the purpose of the system.

The result of this activity is a description of the system in terms of actors and use cases.

Actors represent the external entities that interact with the system.

Actors include roles such as end users,

- other computers the system needs to deal with (e.g., a central bank computer, a network), and the environment (e.g., a chemical process).

Use cases are general sequences of events that describe all the possible actions between an actor and the system for a given piece of functionality.

II. ANALYSIS

During **analysis**, developers aim to produce a model of the system that is correct, complete, consistent, and unambiguous.

Developers transform the use cases produced during requirements elicitation into an object model that completely describes the system.

During this activity, developers discover ambiguities and inconsistencies in the use case model that they resolve with the client.

III. SYSTEM DESIGN

During **system design**, developers define the design goals of the project and decompose the system into smaller subsystems that can be realized by individual teams.

Developers also select strategies for building the system, such as

- the hardware/software platform on which the system will run,
- the persistent data management strategy, the global control flow, the access control policy, and the handling of boundary conditions.

The result of system design is a clear description of each of these strategies, a subsystem decomposition, and a deployment diagram representing the hardware/software mapping of the system.

Analysis deals with entities that the client can understand.

System design deals with a much more refined model that includes many entities that are beyond the comprehension (and interest) of the client.

IV. OBJECT DESIGN

During **object design**, developers define solution domain objects to bridge the gap between the analysis model and the hardware/software platform defined during system design.

This includes
precisely describing object and subsystem interfaces,

selecting off-the-shelf components,
restructuring the object model to attain design goals such as extensibility or understandability,
and optimizing the object model for performance

V. IMPLEMENTATION

During **implementation**, developers translate the solution domain model into source code.

This includes implementing the attributes and methods of each object and integrating all the objects such that they function as a single system.

The implementation activity spans the gap between the detailed object design model and a complete set of source code files that can be compiled.

VI. TESTING

During **testing**, developers find differences between the system and its models by executing the system (or parts of it) with sample input data sets.

The goal of testing is to discover as many faults as possible such that they can be repaired before the delivery of the system.

The planning of test phases occurs in parallel to the other development activities:

- System tests are planned during requirements elicitation and analysis,
- integration tests are planned during system design, and
- unit tests are planned during object design.

MANAGING SOFTWARE DEVELOPMENT

Communication

Rationale Management

Software Configuration Management

Project Management

Software Life Cycle

A. COMMUNICATION

Communication

- includes the exchange of models and documents about the system and its application domain,
- reporting the status of work products,
- providing feedback on the quality of work products,
- raising and negotiating issues,
- and communicating decisions.

Communication is made difficult

- by the diversity of participants' backgrounds,
- by their geographic distribution,
- and by the volume, complexity, and evolution of the information exchanged.

B. RATIONALE MANAGEMENT

Rationale is the justification of decisions

Rationale is the most important information developers need when changing the system.

- If a criterion changes, developers can reevaluate all decisions that depend on this criterion.
- If a new alternative becomes available, it can be compared with all the other alternatives that were already evaluated.
- If a decision is questioned, they can recover its rationale to justify it.

C. SOFTWARE CONFIGURATION MANAGEMENT

Software configuration management is the process that monitors and controls changes in work
products.

For each configuration item, its evolution is tracked as a series of versions.

Selecting versions enables developers to roll back to a well-defined state of the system when a change fails.

D. PROJECT MANAGEMENT

project management includes the oversight activities that ensure the delivery of a high-quality system on time and within budget.

This includes

- planning and budgeting the project during negotiations with the client,
- hiring developers
- and organizing them into teams,
- monitoring the status of the project,
- and intervening when deviations occur.

E. SOFTWARE LIFE CYCLE

A general model of the software development process is called a *software life cycle*