OBJECT-ORIENTED MODELING AND DESIGN

Feza BUZLUCA, Ph.D
Istanbul Technical University
Computer Engineering Department

http://www.faculty.itu.edu.tr/buzluca
http://www.buzluca.info

Introduction

Properties of Software Development and the Goal of the Course

- Programming is fun, but developing quality software is hard. (*Philippe Kruchten*)
- In this course we focus on the challenges of developing "industrial-strength" software.

  They have a very rich set of behaviors, include a lot of components, which cooperate with each other to fulfill some functionalities.

Complexity:

- This type of software systems are developed to solve problems in **complex** real-world systems. For example; banking systems, air or railway traffic control systems, a cellular phone switching system.
- Software inherits **complexity** of the problem domain.
- Today software products are often more complex than other engineering artifacts such as buildings, bridges or vehicles.

Many Components:

- Large software systems include many components and they are developed by teams including a lot of members. **Communication** (interaction) and **cohesion** (harmony) between components play an important role.

Properties of Software Development (Cont'd)

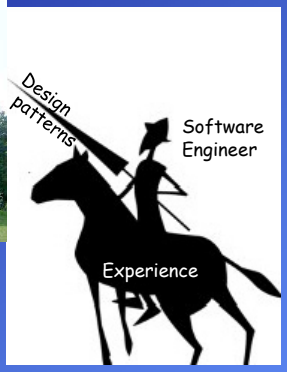
Changes:

- Software systems tend to have a long life span. Requirements change.
- They must be **flexible to be adapted** to new needs.
- They must be **reusable**.

The software crisis

- The failure to handle the complexity of software results in projects that are late,
  over budget (cost is too high),
  and deficient in their stated requirements (many errors).
- Lack of flexibility causes that software can not be easily modified, improved and reused.
- Software maintenance costs are around between 50% and 90% of total software life-cycle cost.
  Maintenance: Changes that have to be made to software after it has been delivered to the customer.

(1)



(2)

Our ***main objective*** is to deal with complexity, handle changes , build ***flexible*** and ***reusable*** software systems, and as a consequence reduce the (maintenance) cost.

(1) From: http://www.photoeverywhere.co.uk
(2) From: http://www.cafepress.co.uk/

Our Tools:

- Software development is both an art and an engineering.
- There isn't any magic formula or any silver bullet (fortunately).

  Intuition and **experiences** play important roles.
- Bjarne Stroustrup: "There are no 'cookbook' methods that can replace intelligence, experience, and good taste in design and programming."

Some helpful tools:

- Software development process:

  The Unified Process (UP): Iterative and evolutionary development
- Use case methodology
- Object-oriented design principles
- Software design patterns
- The Unified Modeling Language (UML)

The main objective of this course is to present
object-oriented design principles and **software design patterns**.

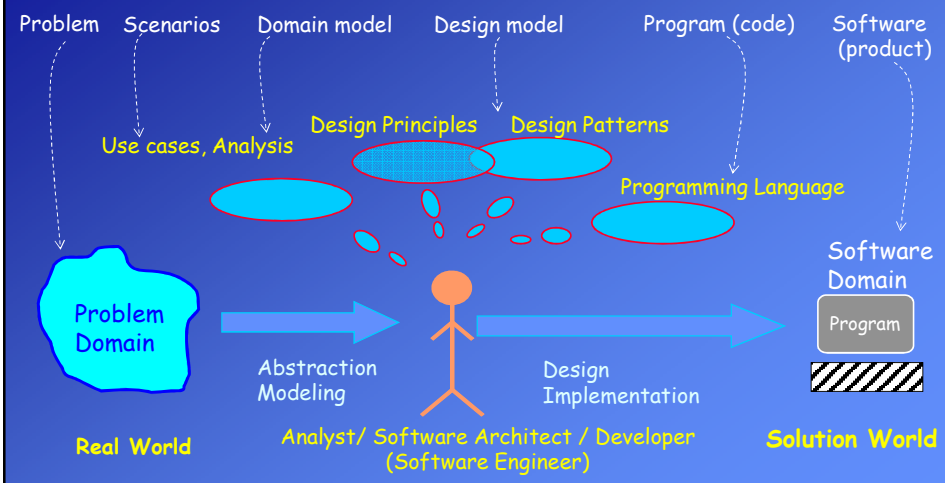
Basic Concepts

Steps of software development :

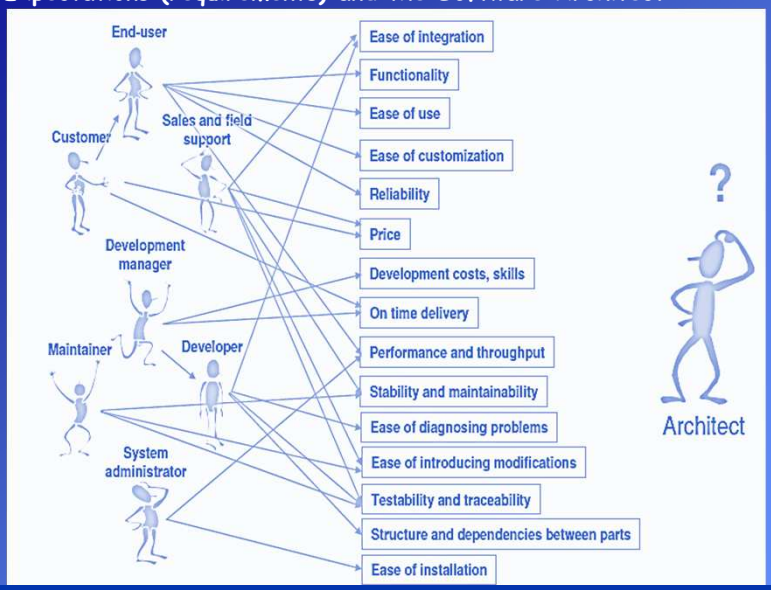
- Specification (Requirements)
  Understanding what the user wants. Writing use cases.
- Domain analysis
  Understanding the system (the problem). What should the system do?
- Design
  Designing the system as collaborating objects.
  Assignment of responsibilities to classes.
- Implementation
  Coding (Programming)
- Evaluation
  Testing, measurement, performance analysis, quality assessment
- Evolution:
  Management, improvement, refactoring

This course focuses on design level, assignment of responsibilities to objects.
We will also see basic concepts about use cases and domain analysis.

The World of a Software Engineer



Expectations (requirements) and the Software Architect



Source: D. Falessi, G. Cantone, R. Kazman, and P. Kruchten, "Decision-making techniques for software architecture design," *ACM Computing Surveys*, vol. 43, pp. 1-28, Oct. 2011.

Object-Oriented Analysis (OOA):

If a civil engineer is building bridges then all s/he needs to know is about bridges.

Unlike this, if you are developing software you need to know

- about *software domain* (because that is what you are building) and
- about the *problem domain* (because that is what you are building a solution for).

Here, analysis means **understanding**.

The analysis (domain) model represents the **real world**.

It does not include our decisions or solutions.

Object-Oriented Design (OOD):

Software classes are designed.

Responsibilities are assigned to classes. All requirements of the system are met.

Object-oriented design principles and software design patterns are used.

The design (software) model represents the **solution world**.

It includes our decisions or solutions.

Analysis: Understanding. The answer of **what?**
Design: Solution. The answer of **how?**

A Simple Example:

Before we go into details of the topics I give an example to show the big picture.

Dice game: We need a software, which simulates a player rolling two dice. If the total is seven, player wins; otherwise, player loses. (*Taken from C.Larman*)

1. Understanding *Requirements*, Defining *Use Cases*

Writing scenarios (stories), which show how the system interacts with its environment.

Example:

Basic flow:

1. The player rolls two dice.
2. The system adds the dice face values and prints the total.
3. The game ends.

Alternative flows:

2.a. The dice face values total 7. The system prints that the player wins.

2.b. The dice face values do not total 7. The system prints that the player loses.

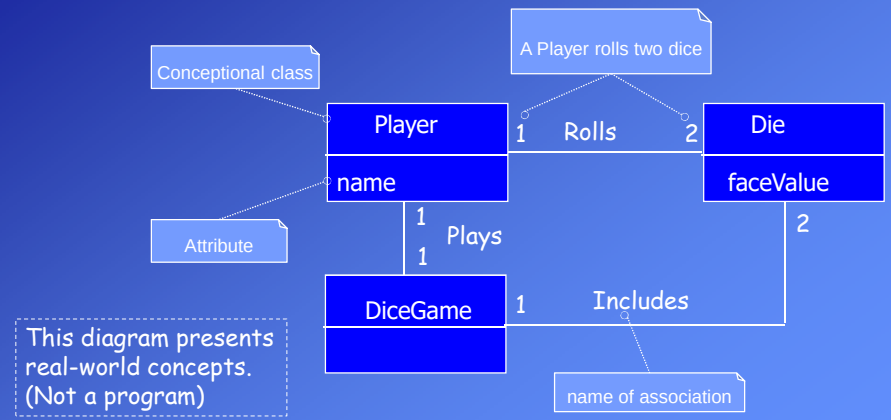
With the help of use cases we will discover **responsibilities** of the system.

2. Analysis, Defining the Domain Model

Identification of the concepts, attributes, and associations that are considered noteworthy. The objective is ***understanding*** the system.

A domain model is not a description of software objects; it is a visualization of the concepts or mental models of a ***real-world*** domain.

It is also called a conceptual object model.

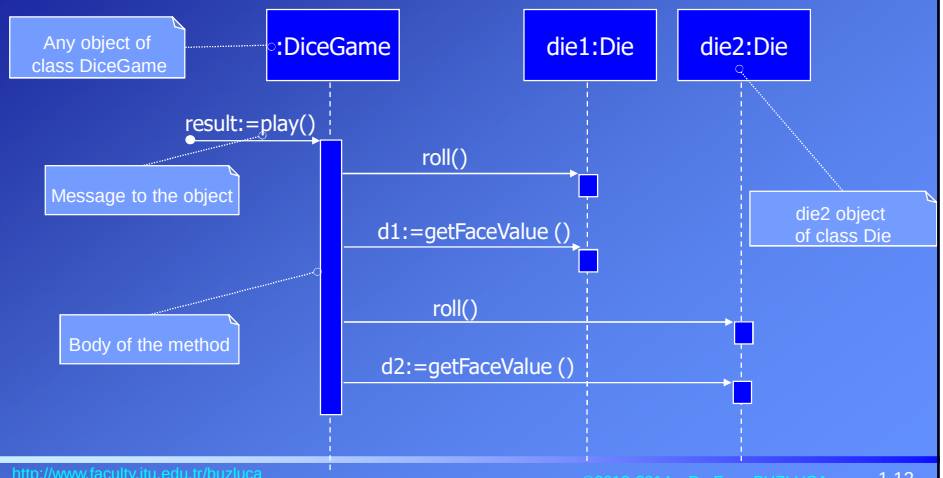


3. Design (Solution Model), Assigning object responsibilities

Defining software classes, their responsibilities and collaborations.

Design artifacts are presented with two different UML diagrams.

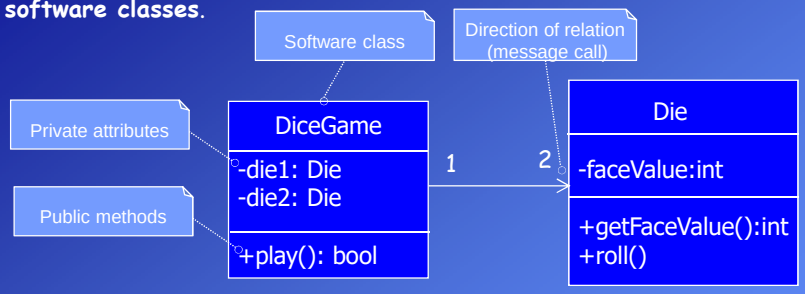
a. Interaction Diagram:



b. Class Diagram

A static view of the class definitions.

In contrast to the domain model showing real-world classes, this diagram shows **software classes**.



Notice that although this design class diagram is not the same as the domain model, some class names and content are similar.

This diagram presents software concepts.

Steps after design:

4. Coding, 5. Testing, 6. Evaluation, 7. Evolution

Why Modeling?

Domain (analysis) models aid our **understanding** of especially complex systems and help to ensure we have correctly interpreted the system under development.

Design models can be used to ensure that all systems requirements are met.

A model also permits us to evaluate our design against criteria such as safety or flexibility before implementation.

Changes are much easier and less expensive to make when they are made in the early phases of the *software lifecycle*.

Models help us capture and record our software design decisions as we progress toward an implementation.

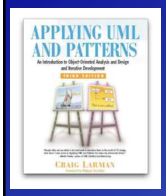
This proves to be an important communications vehicle for the development team.

For example the airplanes can be prototyped in fiberglass and tested in wind tunnels before they are really constructed.

"Progress is possible only if we train ourselves to think about programs without thinking of them as pieces of executable code."
Edsger W. Djikstra (1930-2002)

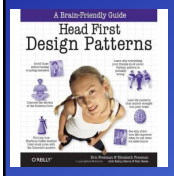
Main reference:

Text books:

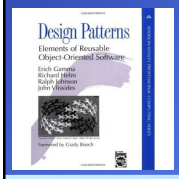


Craig Larman, Applying UML and Patterns , An Introduction to OOA/D and Iterative Development, 3/e, 2005.

Other references:



Eric & Elisabeth Freeman: Head First Design Patterns, O'REILLY, 2004.



Gamma E., Helm R., Johnson R., Vlissides J., *Design Patterns : Elements of Reusable Object-Oriented Software*, Reading MA, Addison-Wesley, 1995.