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

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License. (CC BY-NC-ND 4.0)
https://creativecommons.org/licenses/by-nc-nd/4.0/
https://creativecommons.org/licenses/by-nc-nd/4.0/legalcode

Introduction

Programming is fun, but developing quality software is hard. (*Philippe Kruchten*)

Properties of Software Development and the Goal of the Course

In this course we focus on the challenges of developing "industrial-strength" software.

- They have a very rich set of behaviors.
- They include many components, which cooperate with each other to fulfill some functionalities.
- They are developed by teams including many members.
- They have a long life span. They must be adapted to new requirements.
- Their modules (components) must be reusable to decrease the cost of later projects.

Properties of Software Development (Cont'd)

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, e-commerce system, etc.
- 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.
- A component can be an object (a class),
- a group of classes such as a service in SOA, microservice, or a package in Java,
- another program.

Properties of Software Development (Cont'd)

Changes:

- Software systems tend to have a long life span. Requirements change.
- They must be **extendible** (adding new functionalities according to new needs).
- They must be **flexible to be adapted** to changing requirements.
- They must be **reusable** (reducing the cost).

Example:

Assume that you design a software system for an e-commerce company.

The company has many different, changing discount policies.

For example,

- At the end of the season, there may be 30% or 50% discounts depending on the item.
- In some weeks, Mondays it may be 10% and Thursdays 5% off all sales.
- It may be 150TL off, if the sale total is greater than 1000TL.
- For customers with a loyalty card there may be other discounts.

The company may change this policies or create new sales promotions.

How can our software system adapt to these changes without a big effort?

We want to sell our system to other companies that may have different policies.

How can we reuse components of our existing software system to reduce the cost?

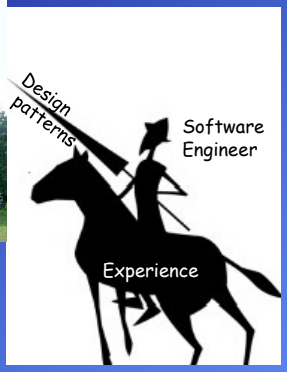
The consequences of failures

- The failure to handle the complexity of software results in projects that are
  late,
  over budget (cost is too high, return on investment (ROI) is low),
  and deficient in their stated requirements (also with some errors).
- Lack of flexibility causes that software cannot 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.

The Goal

(1)



(2)

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

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

License: https://creativecommons.org/licenses/by-nc-nd/4.0/

"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)

We cannot handle software systems as just long texts.
We must consider software systems as complex machines that consist of many components and layers.
Sometimes we need to change, replace, fix, or reuse these components.

```
class ProductSpecification
{
   private:
      ItemID id;
      Money price;
      string specification;
   public:
ProductSpecification( const ItemID &id, const Money &price, const
string &spec )    {
         this->id = id;
         this->price = price;
         specification = spec;
                  }
    const ItemID & getItemId() { return id; }
    const Money & getPrice() { return price; }
    const string & getSpecification() { return specification; }
};

class Sale
{
.....
.....
```

From Instagram @whatchandlove

Our Tools:

- Software development is both an art and an engineering.
- There isn't any magic formula or any silver bullet (~~un~~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: (*SE course*)
  The Unified Process (UP): Iterative and evolutionary development
- Use case methodology (*SE course*)
- Object-oriented design principles (*This course*)
- Software design patterns (*This course*)
- The Unified Modeling Language (UML)

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

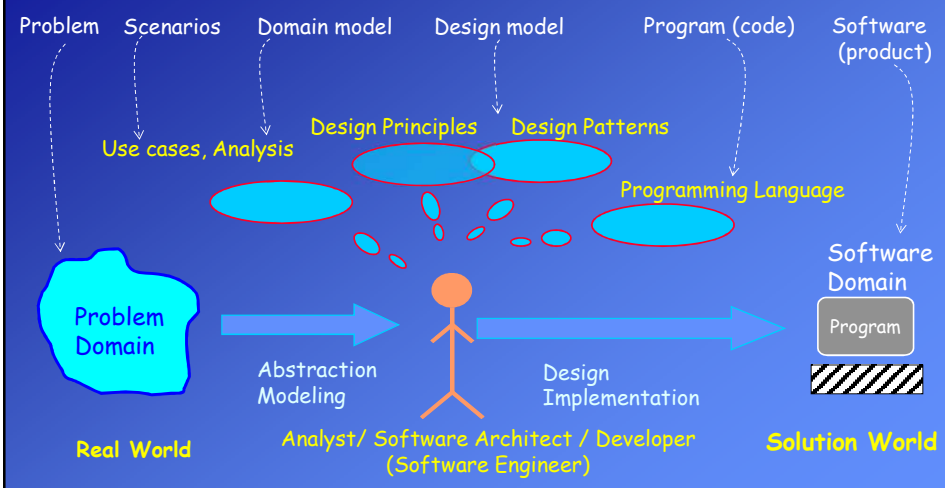
Basic Concepts

Steps of software development :

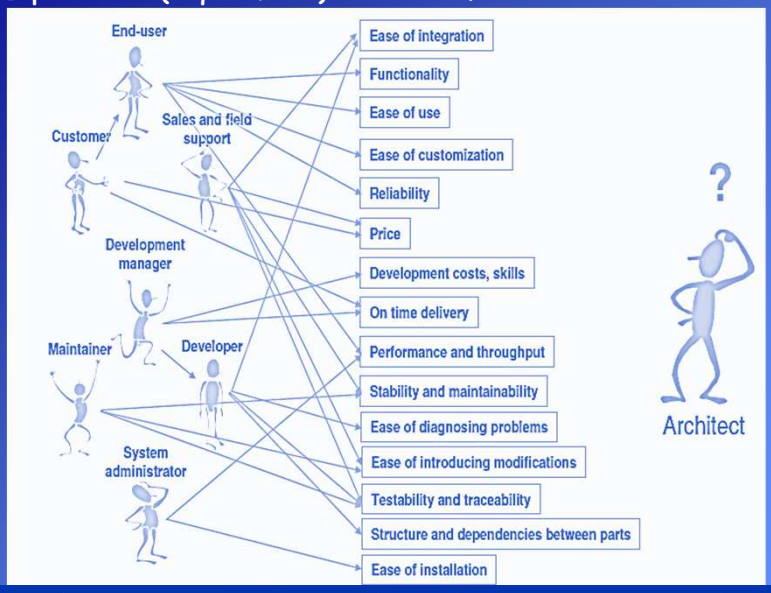
- Specification (Requirements) (*SE course*)
  Understanding what the user wants. Writing use cases.
- Domain analysis (*SE and this course*)
  Understanding the system (the problem). What should the system do?
- Design (*This course*)
  Designing the system as collaborating objects.
  Assignment of responsibilities to classes.
- Implementation (*Programming, data structures*)
  Coding (Programming)
- Evaluation (*Testing and graduate courses*)
  Testing, measurement, performance analysis, quality assessment
- Evolution: (*SE, this course and graduate courses*)
  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** (problem domain).

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 to "**what**"?
Design: Solution. The answer to "**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 entities (**classes**) and **responsibilities** of the system.

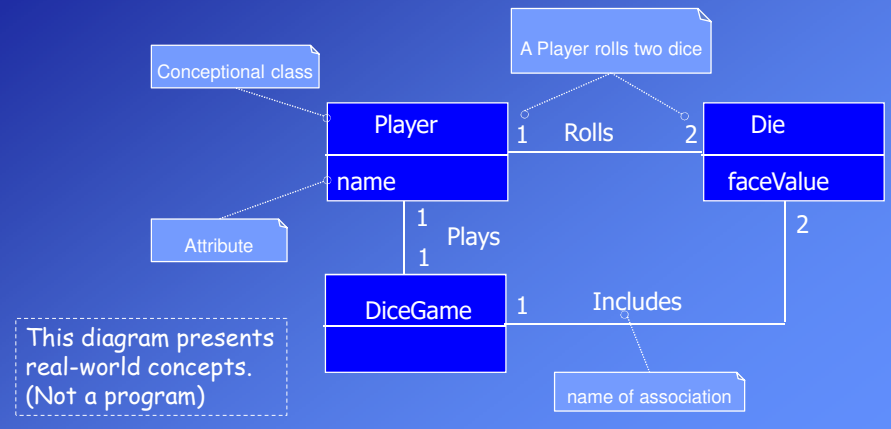
License: https://creativecommons.org/licenses/by-nc-nd/4.0/

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.

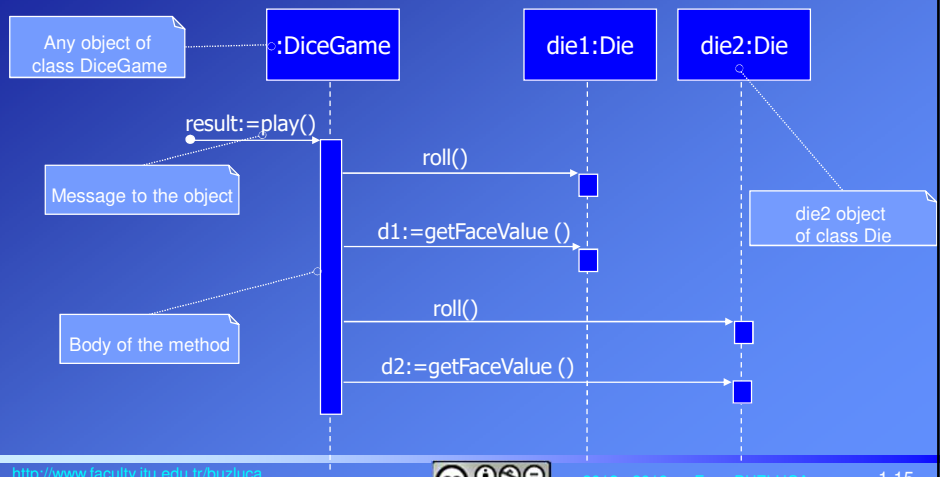
It is also called a **conceptual** class/object model.



3. Design (Solution Model), Assigning object responsibilities

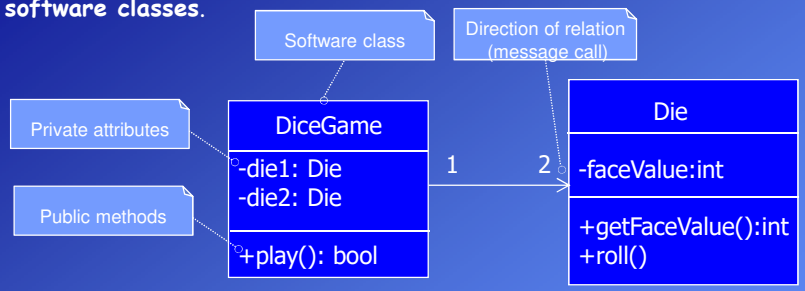
Defining software classes, their responsibilities and collaborations.
Design artifacts are presented with two different UML diagrams.

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 (slide 1.14), 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.

Domain model is also the source of software classes in the design model

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*.

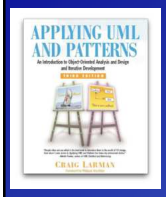
Models help us capture and record our software design decisions as we progress toward an implementation.

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.

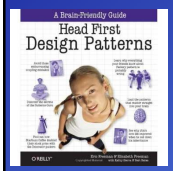
BY NC ND

Main reference: **Text books:**

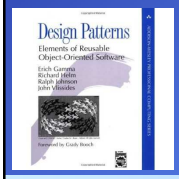


Craig Larman, Applying UML and Patterns , An Introduction to OOA/D and Iterative Development, 3/e, 2005.

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.

BY NC ND