OBJECT-ORIENTED MODELING AND DESIGN

Assoc.Prof. Feza BUZLUCA
Istanbul Technical University
Computer Engineering Department

http://akademi.itu.edu.tr/en/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

This course focuses 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 (contd)

Main Challenges:

Complexity:

- Software systems of this type are developed to solve problems in **complex** real-world systems.

  For example, banking systems, air or railway traffic control systems, cellular phone switching systems, e-commerce systems, etc.
- Software inherits the **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 teams with many members develop them.
- **Communication** (interaction) and **cohesion** (harmony) between components are essential.
- A component can be an object (a class), a group of classes such as a service in SOA, a microservice, a package in Java, or another program.

Properties of Software Development (contd)

Changes:

- Software systems tend to have a long life span. Requirements change.
- They must be **extensible** (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, on Mondays, it may be 10% and Thursdays, 5% off all sales.
- It may be 150TL off if the sale total exceeds 1000TL.
- For customers with a loyalty card, there may be other discounts.

The company may change these policies or create new sales promotions.
How can our software system adapt to these changes without a significant 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 to be easily extended, modified, improved, and reused.
- Software maintenance costs are between 50% and 90% of total software life-cycle costs.

  Maintenance: Changes (improvement, correction, adding new functionalities) that must be made to software after it is delivered to the customer.
- Software errors may cause loss of lives and jobs.

  In 2019, the Boeing 737 Max crash was caused by flaws in software design and not by the pilots or the airline's performance.

Goal of a software development Project:

The ability to deliver a software system

1. that meets the **quality needs** of different stakeholders (user, developer, customer …)
   - Functionality
   - Performance (speed, accuracy, etc.)
   - Efficiency (processor, memory, network, etc.)
   - Reliability (error free)
   - Security (access control)
   - Maintainability (modify, extend, reuse)
   - …

   } Some of the software quality attributes
2. on time,
3. within budget.

Once the systems are operational, the challenges of being on time, on budget, and with the expected quality do not disappear.
They need to be sustained and evolved to meet changing needs and changing environments.

Just writing a code that runs somehow is not sufficient!

You should consider the quality needs of the system's stakeholders.

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

Quality characteristics of a software system

ISO (the International Organization for Standardization) and **IEC** (the International Electrotechnical Commission) prepared standards for quality models.

You may find definitions of the quality attributes of a software system in the following standard.

ISO/IEC 25010: *Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models*

This standard includes two quality models.

A) Quality in use model:

This is the external quality of the system; the impact on stakeholders (customers, direct and indirect users, etc.) in specific contexts of use.

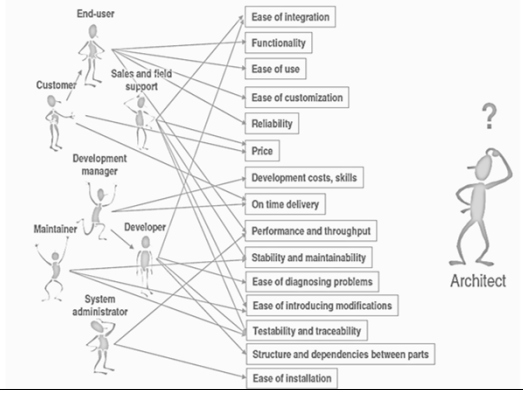
B) Product Quality:

These characteristics relate to the software development team.

You can get the standards in İTÜ campus from the website of the British Standards Online: http://bsol.bsigroup.com/

Details of the quality models are covered in the graduate course "BLG 625 Software Design Quality".

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.

The Goal of the Course

(1)

(2)

Besides meeting the functional requirements of the stakeholders,

our ***objective*** is to learn how to deal with complexity,

handle changes,

build ***extensible***, ***flexible***, ***reusable***, ***error-free*** software systems,

and as a consequence, **reduce the** (maintenance) **cost**.

For this reason, this course presents **object-oriented design principles** and **software design patterns**.

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

"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 must 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

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

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

Some helpful tools:

- Knowledge of Object-Oriented Programming (*OOP course*)
- Software development process: (*SwEng. course*)
  The Unified Process (UP): Iterative and evolutionary development
- Use case methodology (*SwEng. course*)
- **Object-oriented design principles** (*This course*)
- **Software design patterns** (*This course*)
- The Unified Modeling Language (UML) (*OOP and this course*)
- Software testing (*BLG 475E*)
- Software quality measurement and assessment (*BLG 625 PhD Course*)

Object-Oriented (OO) Tools:

OO Design Patterns show you how to build systems with good OO design qualities.

They are proven object-oriented experiences.

Patterns rely on OO basics and principles.

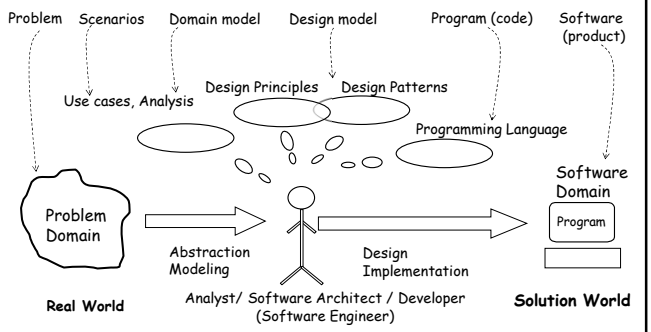
OO Design Patterns (example):
- Strategy:
  Problem: How to design for varying but related algorithms or policies?
  Solution: Define each algorithm/policy/strategy in a separate class with a common interface.

OO Design Principles (examples):
- Strive for loosely coupled designs.
- Find what varies and encapsulate it.
- Favor object composition (has-a) over class inheritance (is-a).
- Design to interface, not to implementation.

OO Basics:
- Encapsulation, Data hiding
- Inheritance
- Polymorphism

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

The World of a Software Engineer



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 the **design level**, i.e., the assignment of responsibilities to objects.

Object-Oriented Analysis (OOA):

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

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

1. about *software domain* (because that is what you are building) and
2. 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 software that simulates a player rolling two dice. If the total is seven, the player wins; otherwise, the player loses. (*Taken from C.Larman*)

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

We write scenarios (stories) that 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.

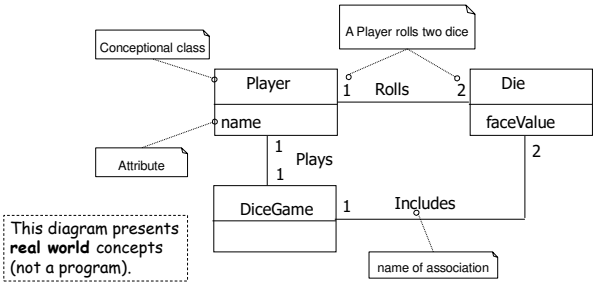
With the help of use cases, we will discover the entities (**classes**) and **responsibilities** of the system.

2. Analysis, Defining the Domain Model

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

A domain model is not a description of software objects but a visualization of the concepts or mental models of the ***real-world*** (problem domain).

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

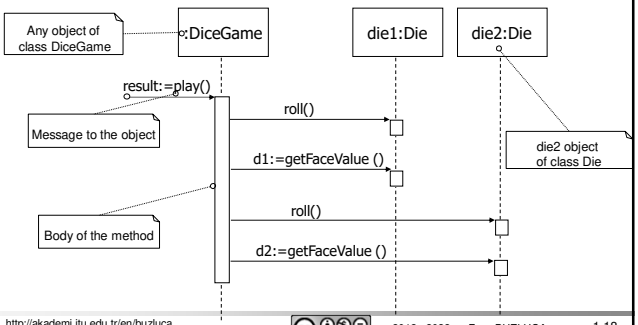


3. Design (Solution Model), Assigning object responsibilities

Defining software classes, their responsibilities, and collaborations.

Design artifacts are presented with two different UML diagrams.

a. Interaction Diagram:

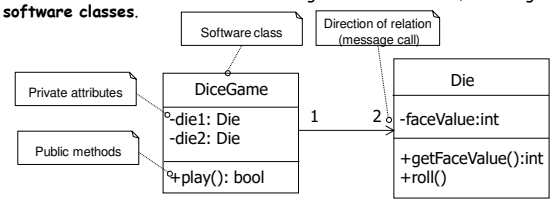


License: https://creativecommons.org/licenses/by-nc-nd/4.0/

b. Class Diagram (Design Model)

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 1.17), 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?

We will create two main models:

1. Domain model
2. Design model

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

  The 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.

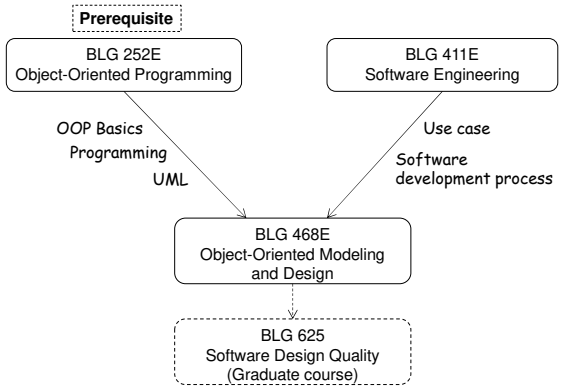
  A model also permits us to evaluate our design against criteria such as safety or flexibility before implementation (coding).

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

  This proves to be an essential communication vehicle for the development team.

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

Related Courses:



Text books:

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

Eric Freeman, Elisabeth Robson, *Head First Design Patterns: Building Extensible and Maintainable Object-Oriented Software*, O'REILLY, 2nd ed. 2020.
You may also use the 1st edition (2004) of this book.

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