OBJECT-ORIENTED PROGRAMMING IN C++

Feza BUZLUCA

Istanbul Technical University
Computer Engineering Department
https://akademi.itu.edu.tr/en/buzluca
https://www.buzluca.info

INTRODUCTION

Main Objectives of the Course :

- To introduce **Object-Oriented Programming** and **Generic Programming**
- To show how to use these programming schemes with the C++ programming language to build "**good**" (high-quality) programs.

Need for high-quality design and good programming methods:

- Today, almost every electronic device includes a computer system controlled by software.
- Software plays a vital role in our daily lives.

Problems:

- Software project costs (especially maintenance costs) are high.

  Maintenance: Changes (requirement changes or bug fixes) and extensions must be made to the software system after it has been delivered to the customer.
- Software errors may cause loss of lives and financial losses.

Examples of software failures:

- In 2018, a software bug was discovered in the UK's NHS (National Health Service) system that put over 10,000 patients at risk of getting the wrong medication.
- In 2024, a faulty security update from CrowdStrike, a cloud security platform, caused widespread system crashes (blue screens of death) on Windows devices globally.

  This affected millions of machines across various sectors, including banking, healthcare, and transportation, leading to an estimated $3 billion in financial losses.
- Two Boeing 737 Max jets crashed, one in Indonesia in 2018 and another in Ethiopia in 2019, resulting in a total of 346 fatalities.

  The crashes were caused by flaws in software design and not by the pilots or the airline's performance.
- Tesla recalled 12,000 cars in 2021 after finding a glitch in its Full-Self Driving beta software.

  A software bug caused vehicles to falsely detect forward collisions, triggering the automatic emergency braking (AEB) system and bringing them to a sudden stop.

The goal of a software development project:

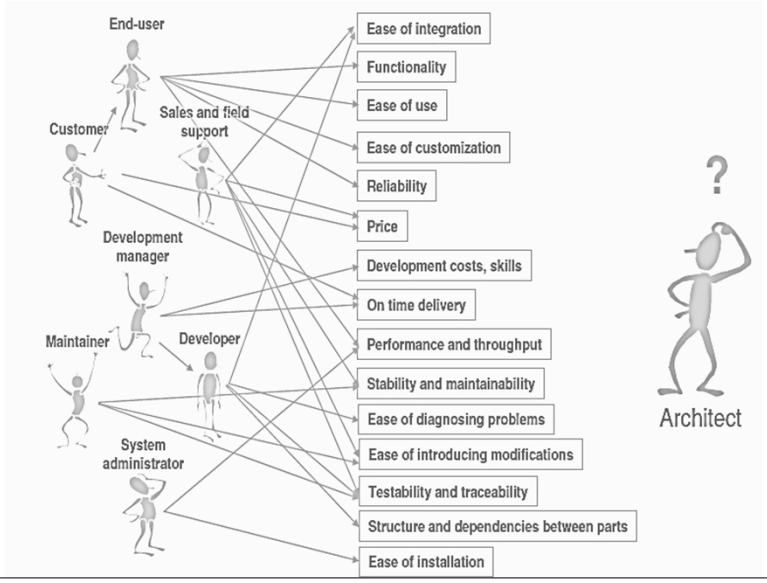
- To deliver a software system that
  1. meets the **quality** needs of different stakeholders (user, developer, customer …)
     - Functionality
     - Performance (speed, accuracy, etc.)
     - Efficiency (processor, memory, network, etc.)
     - Reliability (error-free)
     - Security (access control,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

     Some of the software quality attributes
  2. is on time,
  3. is within budget.
- Once the system is operational, the challenges of being on time, on budget, and having the expected quality do not disappear.
- The system must be maintained and developed to meet changing needs and environments.

Just writing a code that runs somehow is not sufficient!

- One must consider the quality needs of the system's stakeholders.

Expectations of different stakeholders (Quality needs):



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.

Quality characteristics of a software system

- **ISO** (the International Organization for Standardization) and **IEC** (the International Electrotechnical Commission) prepared standards for quality models.
- For definitions of the quality attributes of a software system, refer to the following standards:
  - **ISO/IEC 25010:** *Systems and software Quality Requirements and Evaluation (SQuaRE)* - **Product quality model**

    This standard defines quality characteristics related to the software development team.
  - **ISO/IEC 25019:** *Systems and software Quality Requirements and Evaluation (SQuaRE)* - **Quality-in-use model**

    This is the external quality of the system and the impact on stakeholders (customers, direct and indirect users, etc.) in specific contexts of use.
- Details of the quality models are outside the scope of this course.
- They will be covered in BLG 468E Object-Oriented Modeling and Design (8th semester, undergrad) and BLG 625 Software Design Quality (graduate).
- This course (OOP) will provide a brief insight into a software system's quality attributes (see 1.7) that must always be considered during software development.

Exemplary quality attributes of a software system

- **Functional completeness:** Capability of a product to provide a set of functions that covers all the specified tasks
- **Functional correctness:** Capability of a product to provide accurate results
- **Time efficiency:** Capability of a product to perform its functions within specified time and throughput rates
- **Resource efficiency:** Capability of a product to use no more than the specified amount of resources to perform its function (Resources: processor, memory, disk, network, etc.)
- **Reliability:** Capability of a product to perform specified functions under specified conditions for a specified period of time without interruptions and failures
- **Security**: Capability of a product to protect information and data
- **Modifiability:** Capability of a product to be effectively and efficiently modified without introducing defects or degrading existing product quality
- **Modularity:** Capability of a product to limit the effect of changes to one component on other components
- **Reusability:** Capability of a product to be used as an asset in more than one system, or in building other assets
- **Flexibility:** Capability of a product to be adapted to changes in its requirements, contexts of use, or system environment
- ...

While designing and coding a program (***and learning a programming language***), these quality attributes must always be considered.

Key object-oriented (OO) concepts essential for meeting quality criteria

The object-oriented (OO) approach provides tools to meet quality criteria.

The essential concepts are presented in three layers.

***OO* Design Patterns** (examples):
- Strategy
- Factory
- Adapter
- ...

- They are proven object-oriented experiences (best practices).
- They are covered in the "Object-oriented Modeling and Design" course.

***OO* Design Principles** (examples):
- Open-closed principle. (Open for extension, closed for modification)
- Single-responsibility principle
- Design to interface, not to implementation.
- Strive for loosely-coupled designs.
- ...

- Basic advice about object-oriented design.
- We will cover some of them briefly in this course.
- Details are covered in the "Object-oriented Modeling and Design" course.

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

- Related to programming (coding)
- We will cover them in this course.

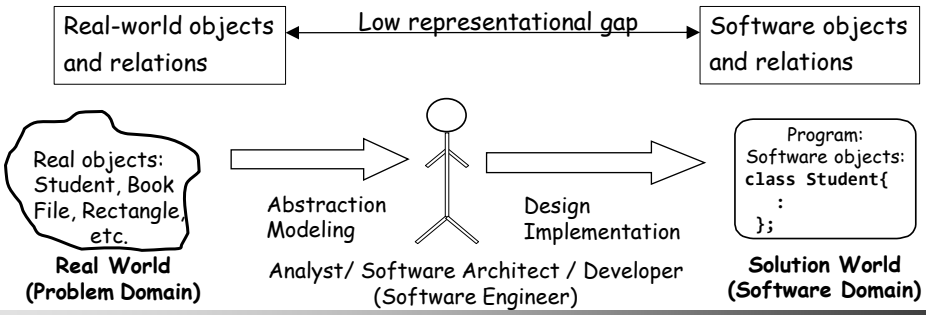
The Object-Oriented Approach

- The fundamental principle of object-oriented programming is the "**low representational gap.**"

The real world (problem) consists of objects.

The software system (solution) also consists of objects.

- Computer programs may contain computer-world representations of the things (objects) that constitute the solutions to real-world problems.
- The close match between objects in the programming sense and objects in the real world increases the quality of the design.



Example:

If you look at a university system, there are many functions and a lot of complexity.

- Students have IDs, they attend courses, they get grades, and their GPAs are calculated.
- Instructors teach courses, manage some industrial and scientific projects, have administrative duties, and their salaries are calculated each month.
- Courses are offered in specific time slots in classrooms. They have a schedule.

Considering every element at once and focusing on functions, a university system becomes very complex.

Object-oriented modeling:

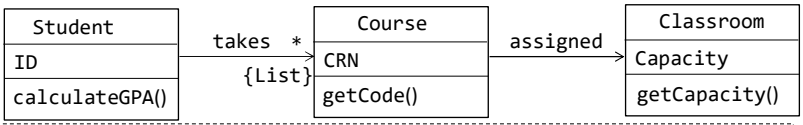
If you wrap what you see in the problem up into **objects**, the system is easier to understand and handle.

- There are students, instructors, courses, and classrooms.
- These objects have attributes and behaviors (abilities or responsibilities).
- There are relations between them.

Thinking in terms of objects:

To solve a problem in an object-oriented language, the programmer should consider three factors:

1. What are the **objects** that make up the problem domain?
   Student, course, instructor, classroom, etc.
2. What are the **responsibilities** of objects?
   Students can calculate their GPAs; instructors can enter grades; classrooms can express their capacities, etc.
3. What are the **relations** between objects?
   Students take courses; students contain a list of courses; a master's student is a special type of student, and courses are assigned to classrooms.



(The Unified Modeling Language (**UML**) is a useful tool to express the model.)

Internal mechanisms and parts that work together are wrapped into a *class.*

What is an object?

Real-world objects have two parts:

1. ***Attributes*** (*property* or *state*: characteristics that can change),
2. ***Behavior*** (or *abilities*: things they can do or *responsibilities*).

Examples:

- Object: Student
  Attributes: ID, Name, Birthdate, List of taken courses, etc.
  Behavior (responsibilities): Calculating their GPAs, listing the course names, etc.
- Object: Classroom
  Attributes: Capacity, timetable
  Behavior (responsibilities): Entering the date of the course into the timetable, showing the schedule, and listing course names assigned to this classroom.

Software objects (classes) also have two parts as real-world objects do:

1. ***Data*** *represent attributes*,
2. ***Functions (methods)*** represent behavior.

What kinds of things become objects in object-oriented programs?

Based on the application domain, various entities with attributes and behavior can be objects.

Examples:

- Human entities: Employees, customers, salespeople, workers, managers
- Graphics program: Points, lines, squares, circles
- Mathematics: Complex numbers, matrices
- Computer user environment: Windows, menus, buttons
- Data-storage constructs: Customized arrays, stacks, linked lists

Example of an Object: A Point in a graphics program

Requirements (from stakeholders):

- A point on a plane has two attributes: x-y coordinates.
- A point's abilities (behavior, responsibilities) are moving on the plane, appearing on the screen, and disappearing.

We can create a model for two-dimensional points with the following parts:

- Two integer variables (**x** , **y**) to represent x and y coordinates
- A function to move the point: **move**,
- A function to print the point on the screen: **print**,
- A function to hide the point: **hide**.

Model (class) of a point:

UML

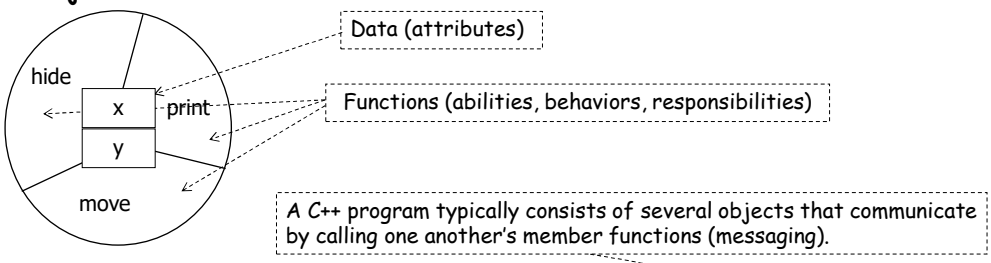
Point
- x, y: Integer
+ void move(Integer, Integer) + void print() + void hide()

Once the model (class) of the point has been built and tested, it is possible to create and activate many point objects from this model.

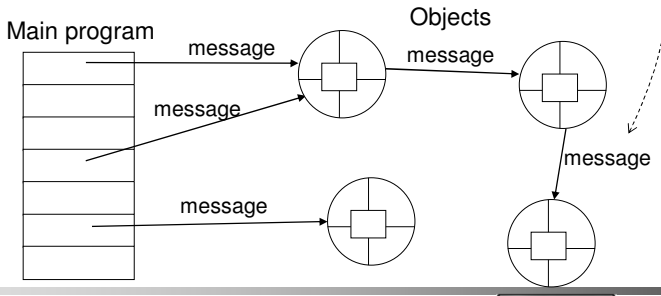
In the example on the right, `point1`, `point2`, and `point3` are three different objects of the same class (model) `Point`.

```
Point point1, point2, point3;
    :
point1.move(50,30);
point1.print();
point2.move(0,100);
```

The Model of an object:



Structure of an object-oriented program in C++:



Key Terms of the Object-Oriented Approach: Encapsulation - Data Hiding

- **Encapsulation:** To create software models of real-world objects, *data*, and *functions* that operate on that data are combined into a single program entity.
  - Data represent the attributes (state), and functions represent the behavior of an object.
  - Data and its functions are said to be *encapsulated* in a single entity (class).
- **Data Hiding:** The data is usually *hidden (private)*, so it is safe from accidental alteration.
  - An object's functions, called *member functions* in C++, typically provide the only way to access its data.
  - If you want to modify the data in an object, you know exactly what functions interact with it: the member functions in the object. No other functions can access the data.
  - This simplifies writing, debugging, and maintaining the program.
- ***Encapsulation*** and ***data hiding*** are key terms in the description of object-oriented languages.
- The other essential concepts of the OOP are ***inheritance*** and ***polymorphism***, which are explained in subsequent chapters.

Summary 1

- The object-oriented approach provides tools for the programmer to represent elements in the problem space (*Low representational gap*).
- We refer to the elements in the problem space (*real world*) and their representations in the solution space (*program*) as "objects."
- OOP allows you to describe the problem in terms of the problem rather than in terms of the computer where the solution will run.
- So, when you read the code describing the solution, you also read words expressing the problem.
- Some benefits of the OOP if the techniques are applied properly:
  - Understandability: It is easy to understand a good program. Consequently, it is easy to analyze, modify, and improve the program.
  - Reliability: Reducing the possibility of errors
  - Extensibility: Reducing the cost of adding new features to the program
  - Modifiability: Reducing the effort to adapt an existing system (quicker reaction to changes in the business environment, requirements)
  - Reusability: Existing modules can be used in new projects.
  - Teamwork: Modules can be written by different team members and integrated easily.

Summary 2

- Programming is fun, but it is related (only) to the implementation phase of software development.
- Development of quality software is a bigger job, and besides programming skills, other capabilities are also necessary.
- This course will cover OO basics: Encapsulation, data hiding, inheritance, and polymorphism.
- Although OO basics are important building blocks, a software architect must also be aware of **design principles** and **software design patterns**, which help in developing high-quality software.
  See the chess vs. software analogy in the following slides.
- Design principles and patterns are covered in another course:
  Object-Oriented Modeling and Design (8th semester).
  http://www.ninova.itu.edu.tr/tr/dersler/bilgisayar-bilisim-fakultesi/2097/blg-468e/

Analogy: Learning to play chess – Learning to design software

Chess:

1. **Learning basics:**
   Rules and physical requirements of the game, the names of all the pieces, and the way that pieces move and capture.
   At this point, people can play chess, although they will probably not be outstanding players.
2. **Learning principles:**
   The value of protecting the pieces, the relative value of those pieces, and the strategic value of the center squares.
   At this point, people can become **good players**.
3. **Studying the games of other masters (Patterns):**
   Buried in those games are **patterns** that must be understood, memorized, and repeatedly applied until they become second nature.
   At this point, people can become chess **masters**.

Learning to play chess – Learning to design software (cont'd)

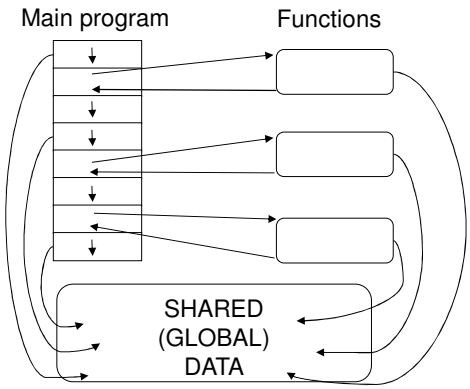
Software:

1. **Learning basics:**
   The rules of the languages, data structures, algorithms, and OOP basics.
   At this point, one can write programs, albeit not always very "good" ones.
2. **Learning principles:**
   Object-oriented modeling and design.
   Importance of abstraction, information hiding, cohesion, dependency (coupling) management, etc.
3. **Studying the designs of other masters (Patterns):**
   Deep within those designs are **patterns** that can be used in other designs.
   Those patterns must be understood, learned, and repeatedly applied until they become second nature.

This chess analogy has been borrowed from Douglas C. Schmidt
He states that it is courtesy of Robert Martin.

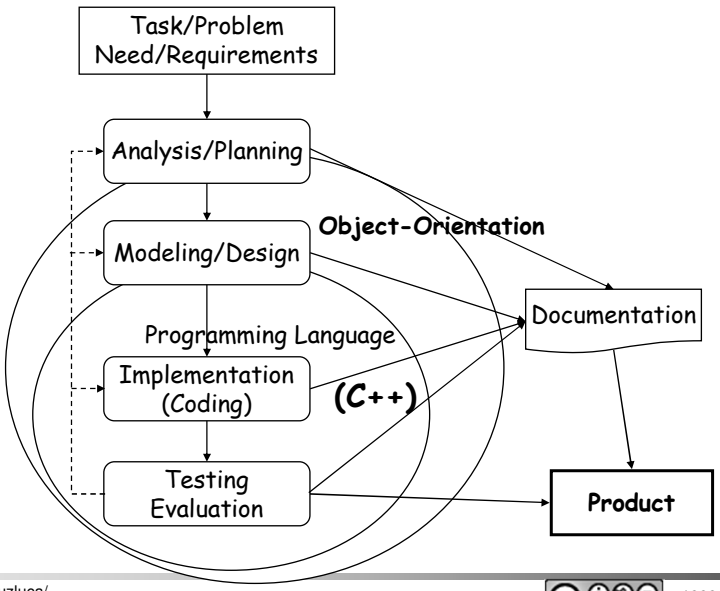
Imperative/Procedural Programming Technique

- In an imperative/procedural programming language (technique), each statement (command) in the program tells the computer to do something.
- The emphasis is on **doing things (functions/procedures)**.
- A program is divided into **functions** (procedures) and - ideally- each function has a clearly-defined purpose and a clearly-defined interface to the other functions in the program.



- Imperative programming also has some advantages, and it is also possible to write good programs using procedural programming (e.g., C programs).
- However, object-oriented programming offers programmers many advantages enabling them to write high-quality programs.

Software Development Process



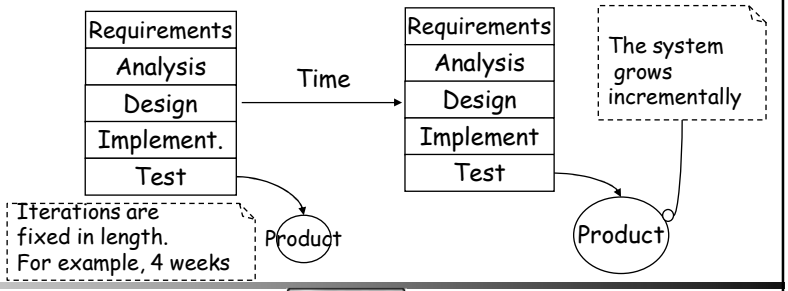
Basic steps of the software development process

- **Analysis:** Gaining a clear understanding of the problem. (Role: Analyst)
  Understanding requirements. Understanding what the user wants. Requirements may change during (or after) the development of the system!
  It is about understanding the system (the problem). **What should the system do?**
- **Design:** Identifying the concepts (entities) and their relations involved in a solution. (Role: Software architect, designer)
  Here, our design style is object-oriented. So, entities are objects (classes).
  This stage has a strong effect on the quality of the software.
- **Implementation (Coding):** The solution (model) is expressed in a program. (Role: Developer)
  Coding relates to the programming language. In this course, we will use C++.
- **Documentation:** Each phase of a software project must be clearly explained.
- **Evaluation:** Testing, measurement, performance analysis, quality assessment.
  The behavior of each object and the whole program for possible cases must be examined. (Role: Quality assurance, Tester)
  Details of the software development process are covered in the "**Software Engineering**" course.

The Unified (Software Development) Process - UP

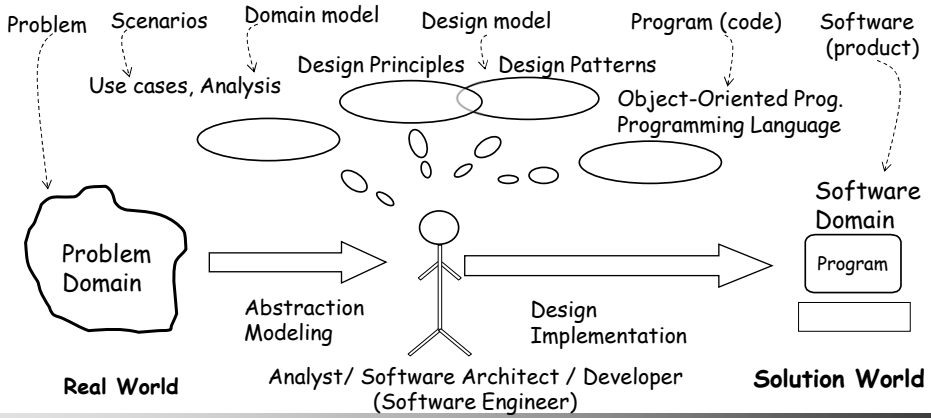
The Unified Process is a popular iterative software development process for building object-oriented systems. It promotes several best practices.

- **Iterative:**
  - Development is organized into a series of short, fixed-length (for example, three-week) mini-projects called iterations.
  - The outcome of each iteration is a tested, integrated, and executable partial system.
  - Each iteration includes requirements analysis, design, implementation, and testing activities.
  - An iteration step of four weeks, for example.
- **Incremental, evolutionary:**
  - The system grows incrementally.
- **Risk-driven:**
  - Risky parts first



What is programming? Steps of software development

- Similar to human languages, a programming language provides a way to express concepts.
- Program development involves **creating models** of real-world situations and building computer programs based on these models.
- Computer programs may contain computer-world representations of **the things (objects)** that constitute the solutions to real-world problems.



AI (Artificial Intelligence) in Software Development

- Tools such as generative AI (Artificial Intelligence), code completion systems, and automated testing platforms reduce the need for engineers, developers, and programmers to manually write code, debug, or conduct time-consuming tests.
- Some of the key areas where AI is used in software development:
  - Code generation: AI tools assist developers by suggesting code or generating entire functions from natural language inputs, speeding up development by automating routine tasks.

    Exemplary tools: GitHub Autopilot, GitHub Copilot, and IBM watsonx Code Assistant.
  - Bug detection and fixing: AI-driven tools can automatically detect bugs, vulnerabilities, or inefficiencies in the code.
  - Testing automation: AI tools generate test cases from user stories and optimize tests, which reduces manual testing time and increases coverage.
  - Documentation: AI tools use NLP (Natural Language Processing) to generate and maintain documentation, turning code into readable explanations and helping ensure up-to-date project information.
  - Refactoring and optimization: AI suggests code improvements to optimize performance and make code easier to maintain.

Source: https://www.ibm.com/think/topics/ai-in-software-development

AI in Software Development (cont'd)

- AI is fundamentally redefining the role of software engineers and developers, moving them from code implementers to orchestrators of technology.
- Many believe it augments developers instead of replacing them, enabling focus on system optimization and innovation.
- By automating routine tasks, AI boosts productivity and frees engineers to focus on higher-level problem-solving, such as architectural planning, design, system integration, strategic decision-making, and creative challenges.

  This shift can drive greater innovation and efficiency.
- Human expertise is still required to guide and refine AI outputs, helping ensure that the technology complements rather than disrupts the development process.

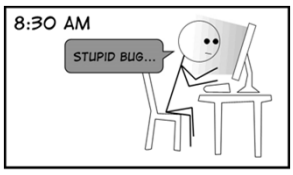
  AI tools do not always generate correct solutions.

Source: https://www.ibm.com/think/topics/ai-in-software-development

A possible road in your professional life

If you will work in the world of software development

Writes the code. (AI tools can also write codes.)	Designs the architecture. Coaches the team. **Decides.**	Management duties. Not only about software
Programmer/ Developer	**Software Architect**	Project Leader/Manager



Source:
http://www.smashingapps.com/

Source:
http://www.planetgeek.ch

Source:
http://www.businessadministrationinformation.com/

Learning a Programming Language

- Knowledge about a programming language's grammar rules (syntax) is not enough to write "good" programs.
- The essential thing to do when learning to program is to focus on concepts (**design and programming techniques**) and not get lost in language-technical details.
- AI tools are quite proficient at coding.
- Try to understand the programming scheme (e.g., object-oriented design), rather than the programming language's rules.

  Understanding design techniques comes with time and practice.
- Learn and use design principles and design patterns.
- Always consider quality characteristics (understandability, flexibility, …).
- Focus on higher-level problem-solving, such as architectural planning, object-oriented design, strategic decision-making, and creative challenges.

Why C++?

The main objective of this course is not to teach a programming language. However, examples are given in C++.

Properties of the C++ programming language:

- C++ supports object-oriented and generic programming.
- Performance (especially speed) of programs written with C++ is high.
- It is helpful in low-level programming environments where direct control of hardware is necessary.

  Embedded systems and compilers are created with the help of C++.
- C++ gives the user control over memory management (also increases the programmer's responsibility "*with authority comes responsibility*").
- C++ is used by hundreds of thousands of programmers in every application domain.
  - Hundreds of libraries support this use.
  - Hundreds of textbooks, several technical journals, and many conferences.
- C++ programmers can quickly adapt to other object-oriented programming languages such as Java or C#.

The applications domain of C++:

- Game (engine) development: Speed and control over hardware are crucial.
  Examples: Fortnite and Unreal Engine
- Graphics and user interface programs
- Systems programming: Operating systems, device drivers. Here, direct manipulation of hardware under real-time constraints is essential.
- High-performance applications: Scientific computing and financial modeling.
- Embedded systems: For example, systems for cars and medical devices.
  It is possible to implement relatively small and efficient programs that can run on limited hardware resources.

Examples of applications written in C++:

- Apple's Mac OS X,
- Adobe Systems,
- Backend services of Facebook,
- Google's Chrome browser,
- Microsoft Windows operating systems, MS Office, Visual Studio
- Mozilla Firefox, Thunderbird,
- MySQL

are written in part or in their entirety with C++.

C++ Standards

C++ is standardized by the working group WG 21 of the International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC).

Official: ISO/IEC JTC1 (Joint Technical Committee 1) / SC22 (Subcommittee 22) / WG21 (Working Group 21): JTC1/SC22/WG21

- The current C++ standard is **C++23, i.e.**, ISO/IEC 14882:2024.
  It was finalized in 2023 and published in 2024.
- The next planned standard is C++26.
- You can get the standard in İTÜ campus from the website of the British Standards Online: http://bsol.bsigroup.com/
- Information about C++ standards: https://isocpp.org/std/the-standard
- Be aware of programming standards and use compilers that support the current one.
  https://en.cppreference.com/w/cpp/compiler_support
  For example, you can use MS Visual Studio, Clang, or GCC.