# SOFTWARE DEVELOPMENT

In today's digital age, software is the basis of almost all technological solutions. We use it at school, at home, at work, in the car. It is part of our smartphones, computers, traffic control systems, healthcare, and financial services. Software helps us solve problems, automate tasks, communicate, have fun, and learn. Therefore, it is important to understand how software is created, what its principles are, and how to develop it effectively.

## Software

Software is a set of instructions and data that allows computers and other devices to perform specific tasks. Unlike hardware, which represents the physical components of a device, software is intangible and serves to control and control the hardware. Software is essential for the functioning of a computer and all digital devices.

Software can be divided into:

1. **System software** – Includes operating systems (e.g., Windows, Linux, macOS) and essential programs that make a computer work. This software controls devices, manages memory, and ensures the interaction between hardware and application software.
2. **Application software** – Refers to programs that provide specific tools or services to users, such as web browsers, a text editor, or mobile applications. The software allows users to perform various tasks, such as document creation, data management, or internet access.

Software is a constantly evolving product that adapts to new technologies and user requirements. It plays a key role in all areas of life, and its constant development allows for innovation and advancement in technology.

## Software Engineering

Software engineering is a discipline that focuses on applying engineering principles and methodologies to the development, deployment, and maintenance of software. It is a systematic, methodical and effective approach to software development. It covers the entire software lifecycle, from requirements analysis and specification, through architecture design and implementation, to testing, deployment, and long-term maintenance. Software engineering uses knowledge from computer science, project management, and other fields. The goal of software engineering is to create reliable, efficient, and maintainable software that meets the requirements of users and customers.

A key aspect of software engineering is the use of methodologies and tools that help in the development of quality software. These include, for example, software development models (e.g., agile model, waterfall model), UML diagrams for visualization and documentation, code version control tools, and software testing and debugging tools. Software engineers also deal with the issues of project management, team collaboration, communication with clients and users to understand their needs and expectations, and to ensure the successful completion of the project.

## Stages of software development

Software development is the process of creating software. It is a complex set of activities that includes several phases from the initial idea and planning, through coding and testing, to the deployment of the created software in the real environment and its maintenance.

The main stages of software development include:

1. Requirements Analysis
2. Proposal
3. Implementation
4. Testing
5. Deployment
6. Maintenance

Software development can be simply compared to building a house. First, it is determined what the house should look like and what it should meet (requirements). Then plans are drawn (design), materials and tools are purchased (programming language, development environment) and the construction of the house (programming) begins. Finally, the house is furnished and maintained (deployment and maintenance).

### *Requirements Analysis*

Requirements analysis is the first and one of the most important phases of software development. Its goal is to understand what the user or client expects from the software being developed and to precisely define its functionality, features and limitations. A high-quality requirements analysis minimizes the risk of misunderstandings and errors in later stages of development.

Requirements analysis involves identifying all stakeholders, collecting their requirements through various methods such as interviews, questionnaires, document analysis, etc. Subsequently, these requirements are analyzed, categorized and specified, distinguishing functional requirements from non-functional ones. An important step is also the prioritization of requirements and their documentation in a clear document, which serves as the basis for further phases of development. Finally, it is important to verify that the requirements are properly understood and meet the customer's expectations.

As mentioned above, there are two main types of requirements:

1. **Functional requirements** – determine what the software should do (e.g. the software allows the user to register by entering an email address and password, the manager can view issued invoices, etc.).
2. **Non-functional requirements** – relate to qualitative aspects of the software, such as performance, security and usability (e.g. card verification is carried out within two seconds, the software is supposed to be scalable, compatible with other systems, etc.).

### *Proposal*

It is the second step of software development after requirements analysis. The main goal is to transform the requirements into a description of the technical solution and specifications for the software being developed.

In this phase, the plan and architecture of the software solution are created so that the system is efficient, sustainable and easily expandable. Decisions are also made about the technologies, programming language, and overall architecture of the system. For better visualization, specification and understanding of software design, UML (Unified Modeling Language) diagrams are often used, which show the structure and functions of the software.

An important part is the design of the user interface (UI), which should be intuitive and user-friendly. When designing a user interface, software previews (designs for the layout of elements on the screen) are created, also called wireframes or mockups.

The result of this phase is comprehensive documentation that contains all the details of the software design, including UML diagrams and wireframes. The result of the software design can also be a prototype that serves as an interactive demonstration of the future product. The prototype demonstrates the functionality and appearance of the proposed software and thus allows for a better understanding and verification of the requirements before the actual implementation.

Software design serves as the basis for the next stage of development, which is the implementation of the software.

### *Implementation*

The implementation phase in software development is the process of translating a software design into functional code that can be executed by machines (computers, smartphones...) At this stage, programming languages, development environments (IDEs), frameworks (a predefined set of tools, components and rules that provides the structure and basis for the development of software applications, thus speeding up and simplifying the implementation process) and libraries (files of reusable code that provide functions, methods and tools to solve common tasks, thus facilitating development and reducing the need to write code from scratch) are used to implement defined functionalities of the system. A high-quality implementation requires adherence to coding standards, clean code principles, and design patterns that increase the readability and efficiency of code creation.

An important part of the implementation is code versioning, which allows for efficient management of code changes through version control systems such as Git. This approach is useful for team collaboration, allows you to go back to previous versions, resolve conflicts efficiently, and maintain a history of changes, which increases control and security while working on a project. The completion of the implementation is a functional software product that goes through further testing and optimization processes before being deployed into production.

### *Testing*

Software testing is a key step in the process of ensuring the quality and reliability of the software product being developed. Testing focuses on identifying errors and flaws in the code, functionality, or behaviour of the system, while observing whether the application meets defined requirements and standards. Testing is often supported by automation tools that allow tests to be run repeatedly at different stages of development and with each code change.

Different types of tests are used at this stage. In terms of software functionality, these are:

1. unit tests – verify the functionality of individual components,
2. integration tests – focus on the interaction between different modules,
3. System tests – evaluate the overall functionality of the system in real conditions.

Other types of tests include:

1. Regression testing – ensures that new changes to the code do not affect the existing functions of the system.

2. Performance testing – it is carried out depending on the software requirements. The reaction time, coping with high loads, and stability of the software under different conditions are examined.
3. Security tests – detect potential vulnerabilities.

Testing is not a one-time process, but takes place throughout the entire software lifecycle, with collaboration between developers and testers being essential to achieve the desired product quality and reliability.

### Deployment

Deployment is the software development phase in which the software being developed is moved from a development or test environment to production, where it will be made available to end users. This process includes not only the actual running of the software on the production servers, but also the other steps required to get it fully operational, such as system configuration, database setup, implementation of security measures, and integration with other systems. The goal of deployment is to ensure that the software works reliably and efficiently in a real-world environment and meets user requirements. If data migration is part of your deployment, it's essential to plan and execute it carefully to avoid data loss. It is also important to ensure that users are properly trained to use the new software. Deployment can be a complex process that requires coordination between multiple teams and stakeholders.

### Maintenance

Once you've deployed your software to production, it's essential to monitor its performance and stability, as well as perform regular maintenance. Maintenance includes all the activities necessary to maintain, optimize, and improve a system throughout its lifecycle. At this stage, bug fixes are carried out that appear after deployment in the real environment, as well as software updates to consider changes in user requirements or technological conditions. Maintenance may also include performance improvements, modifications to improve system security, and the implementation of new features that are necessary to maintain competitiveness or meet new regulatory requirements.

In addition to technical modifications, maintenance also manages compatibility issues with older versions of the system, as new updates may affect existing functionalities. This process is often supported by continuous monitoring and diagnostics, which makes it possible to identify potential problems before they affect users. Maintenance is a long-term process that requires regular care and accountability on the part of the development team to ensure the stability, security, and efficiency of the system throughout its lifetime.

## Software Developer Team and Their Roles

Depending on the size and complexity of the project, an entire team of software developers may be involved in software development. A software development team is a complex structure in which each member fulfills a specific role and contributes to the success of the project.

Key roles in this team include:

1. **Team leader** – Is responsible for leading and coordinating the team's work. Their job is to plan and organize work, motivate team members, and ensure that the project is completed successfully. A team leader should have strong leadership skills, technical knowledge, and the ability to communicate effectively.
2. **Analyst** – Is responsible for analysing user requirements and translating them into technical specifications. Its task is to communicate with users, collect requests and propose a solution. Analysts should have good communication skills and the ability to analyse and solve problems.
3. **Software architect** – His task is to design and define the technical structure and architecture of a software product so that it is scalable, sustainable and efficient. A software architect selects the appropriate technologies, design patterns, and components to ensure optimal system performance, security, and compatibility. In addition to technical decisions, the software architect is also responsible for ensuring that the system meets the requirements for flexibility and extensibility in the future.
4. **Developer** – Is responsible for writing code and implementing software functionality. The team can include different developers specializing in different areas, such as frontend developers, backend developers, or database specialists. Developers should have good programming skills and the ability to work in a team.
5. **Tester** – Is responsible for testing the software and identifying bugs. Its task is to carry out different types of tests and document the results. Testers should have good analytical skills and be able to work with testing tools.

Depending on the size of the project, there may also be various other roles in the team, such as project manager, product manager, UX/UI specialist, graphic designer, scrum master, or for smaller projects, some roles can be merged, and multiple roles can be performed by one person. However, team members should have clearly defined goals and responsibilities. Effective collaboration and communication between team members are crucial to the success of a project.

## Programming

Programming is the process of creating and implementing algorithms through programming languages to control the operation of computer systems and various devices. Programming is the process of creating instructions for a computer to perform required tasks. These instructions are written in programming languages that are understandable to the computer. Programming languages are similar to human languages, but with precise rules and syntax. The entire process includes logic design, syntactic and semantic correctness of the code, performance optimization, and ensuring the reliability of the resulting solution. Effective programming requires not only knowledge of the syntax of a particular language, but also an understanding of the algorithmic and data structures that enable efficient information processing.

Programming is extremely important in today's world. It is the basis for the operation of many of the technologies we use every day. Programming is a cornerstone of software development and is used to create different types of software solutions, from simple mobile games to applications that simplify and streamline daily activities, to complex systems that manage the operation of companies. Through programming, it is possible to create applications that are used for data processing, communication, entertainment, education or management of business operations. Programming also allows you to automate various processes, which increases work efficiency and productivity. It is also crucial for analysing data and solving various problems.

## *The importance of learning to programming*

Programming is one of the most valuable skills in the digital era, as it provides a wide range of applications in various fields, from information technology to science, finance, or healthcare. With the growing demand for programmers in the job market, programming offers stable and well-paying career options. The ability to write code and program allows individuals to create software solutions, automate repetitive tasks and processes, leading to increased efficiency and productivity at work, industry, and many other industries. It also allows large amounts of data to be collected, processed, and analysed, which is important for scientific research, business, and decision-making in various fields.

However, programming is not just about writing code. It's about problem-solving, creativity, and being able to see things from a new point of view. J. M. Wing (2006) even argues that the ability to "think like a computer scientist" should be among the basic skills such as reading, writing and arithmetic. At the same time, programming develops logical and analytical thinking, because effective programming requires understanding algorithms, optimizing computational processes, and solving complex problems.

Therefore, learning programming is not only beneficial for future professional programmers, but it is also beneficial for every single person. In addition to professional opportunities, learning to code also brings personal benefits. Programming develops important cognitive abilities. Professional publications state that programming and its learning develops various skills important in the 21st century, such as problem-solving skills, computational thinking, critical thinking, algorithmic thinking, creative thinking, analytical thinking, logical thinking, abstract thinking, deductive thinking, ability to collaborate and cognitive skills in general.

Regardless of the field, programming literacy is a great advantage that allows you not only to better understand modern technologies, to be able to use them better and more effectively, but also to actively participate in their development and innovation.

## *Programming languages*

A programming language is a formal system designed to write algorithms and instructions that a computer can run and thus control the operation of computer systems. An algorithm is a sequence of steps that lead to the solution of a certain problem. The algorithm is thus the basis of any program and defines the logic of its operation. Programming languages are essential for software development because they allow programmers to communicate with a computer in a way that the computer understands. There are many programming languages, each with its own specifics and suitable for different types of tasks.

Each programming language has its own syntax and semantics. Syntax defines the rules for writing code. Each programming language has its own syntax that programmers must follow for the code to be translated correctly and to be executed. Semantics determines the meaning of individual constructions in a language. Defines what effect each instruction has on the state of the program.

Programming languages can be divided into:

1. **Low-level** – they are close to computer hardware and machine code. An example of such a programming language is Assembly.
2. **High-level** – these are more abstract and allow programmers to work with data and operations at a higher level. Examples of such programming languages are C, C++, Java, Python.

## *Control structures in programming*

Control structures are basic constructions in programming that determine the sequence of execution of instructions in a program. They enable decision-making, task repetition, and code organization so that the program performs the required operations efficiently and correctly. Control structures are an essential part of programming and understanding them is crucial for successful software development. The basic control structures include sequence, branching, and cycle.

### Sequence

This is the simplest type of control structure, where instructions are executed sequentially, in the order in which they are written in the code. Figure 1 shows the general notation of a sequence through a flowchart. Designations p1 ... PN depicts orders that are executed in the order in which they are written.

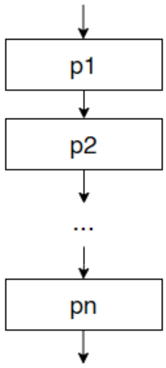


Figure 1 General Sequence Notation via Flowchart

### Branching

It allows different pieces of code to be executed based on the fulfilment of a certain logical condition. By default, this condition divides the flow of the program into two parts – **branches**. Regarding the fulfilment or non-fulfilment of the condition, either one or the other branch is executed, which may contain a different number of orders (Figure 2). For example, if the correct login data has been entered (condition), then log in the user (command in one branch), otherwise it will warn him of the incorrect data entered (command in the second branch).

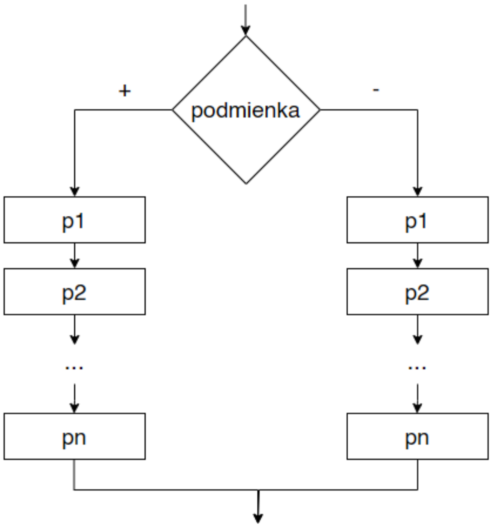


Figure 2 General Branching Notation via Flowchart

**Cycle**

It ensures that orders are executed repeatedly until a condition is met. By repeating a certain piece of code, cycles reduce redundancy and increase the efficiency of programs. There are several types of cycles, and their main division is according to whether we know the exact number of repetitions of the cycle before starting the cycle. Figure 3 illustrates the general notation of a cycle through a flowchart, where p represents one or more commands that are repeated within the cycle.

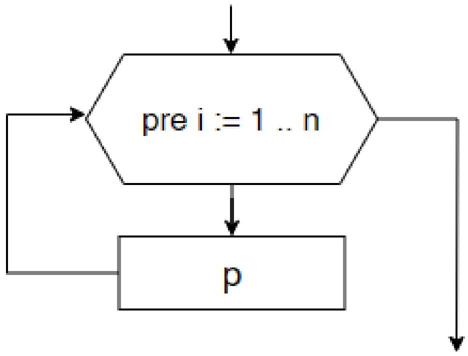


**Figure 3 General Cycle Notation via Flowchart**

### *Integrated development environment*

An Integrated Development Environment (IDE) is a software tool that unifies the various tools needed in software development. An IDE provides software developers with comprehensive tools to efficiently develop, debug, and manage software projects. It includes a text editor with support for programming language syntax, a code compiler (compiler or interpreter), debugging tools, and other functionalities that facilitate the development process. The IDE simplifies the work of software developers by combining all the necessary tools into a single interface, increasing their productivity. Modern IDEs also provide advanced features such as code autocomplete, refactoring, visualization of the project structure, or integration with versioning systems (e.g. Git).

Popular IDEs include Visual Studio, IntelliJ IDEA, or Eclipse. Using an IDE streamlines the development process and allows programmers to focus on the logic and functionality of the software instead of dealing with the technical details of code translation or debugging.

## Trends and the future of software development

Software development is constantly evolving due to technological innovations and changing market demands. **Artificial intelligence is** currently booming and influencing the performance of activities in various industries, including programming and software development. Artificial intelligence is fundamentally changing the way software is programmed and developed. Automating many routine tasks, such as code generation, error detection, or algorithm optimization, allows developers to focus on a higher level of software design and architecture. Modern AI tools can assist in writing code, suggest solutions to programming problems, and even recommend optimal practices for given tasks, thus contributing to more reliable code and thus the software being developed. This development significantly speeds up the programming process and increases the productivity of development teams. Artificial intelligence is also creating opportunities for new types of applications, such as intelligent chatbots and image and speech recognition systems.

The use of the so-called **Platforms with no or minimal code** - **no-code** and **low-code platforms**. These two approaches to software development allow users without advanced programming knowledge to create applications. At the same time, by making application creation available to more people, the digital transformation of companies is accelerated. Zero- or minimal-code platforms simplify the software development process by minimizing or eliminating the need to write text code. Applications are created through visual programming, drag and drop, and pre-built components, or with minimal text-based programming. These platforms are ideal for quickly developing simpler applications, prototypes, and automating workflows.