Computer Science I

Dr. BENTATA khadidja

UNIVERSITY OF M'SILA

FACULTY OF SCIENCE AND TECHNOLOGY

EMAIL: KHADIDJA.BENTATA@UNIV-MSILA.DZ

1.0 JANUARY 2024

Dr. BENTATA khadidja

Table des matières

Objectifs

By the end of this module, the student will be proficient in this subject area according to the cognitive actions specified by Bloom. Here is a detailed breakdown for each level of Bloom's Taxonomy:

1. Knowledge Level

This level involves the ability to recall basic information and facts.

1.a. **Basic Definitions and Concepts:**

· Hardware comprises the physical components of a computer system.

· Software encompasses the programs, applications, and data that run on a computer system.

1.b. **Components of Hardware:**

· Central Processing Unit (CPU): Often referred to as the 'brain' of the computer.

· Memory: Also known as RAM (Random Access Memory).

2. Comprehension Level

This level involves understanding and interpreting the presented information.

2.a **Explaining Component Functions:**

· The CPU executes instructions stored in memory.

· Memory temporarily holds data and instructions that the CPU needs to access quickly.

·2.b **Describing Software Processes:**

· The design phase of software development involves conceptualizing the structure and functionality of a program.

3. Application Level

This level involves using knowledge in new or practical situations.

Applying Concepts in Programming:

· Writing Code: Translating the design specifications into executable code using a programming language.

· Testing: Validating the functionality and correctness of the software through various testing techniques.

4. Analysis Level

This level involves breaking down information into its basic components and understanding its structure.

4.a. **Analyzing Software Development Phases:**

· Overview of the software development lifecycle, including requirements analysis, design, implementation, testing, and maintenance phases.

·4.b. **Analyzing Programming:**

· Algorithm Design: Developing algorithms to solve specific computational problems efficiently.

5. Synthesis Level

This level focuses on combining information to create something new or propose innovative solutions.

5.a. **System Design:**

· System Design: Defining the architecture, components, and interactions of the software system.

5.b. **Innovative Software Creation:**

· Designing data structures to organize and manage data effectively.

6. Evaluation Level

This level involves assessing information or ideas using specific criteria.

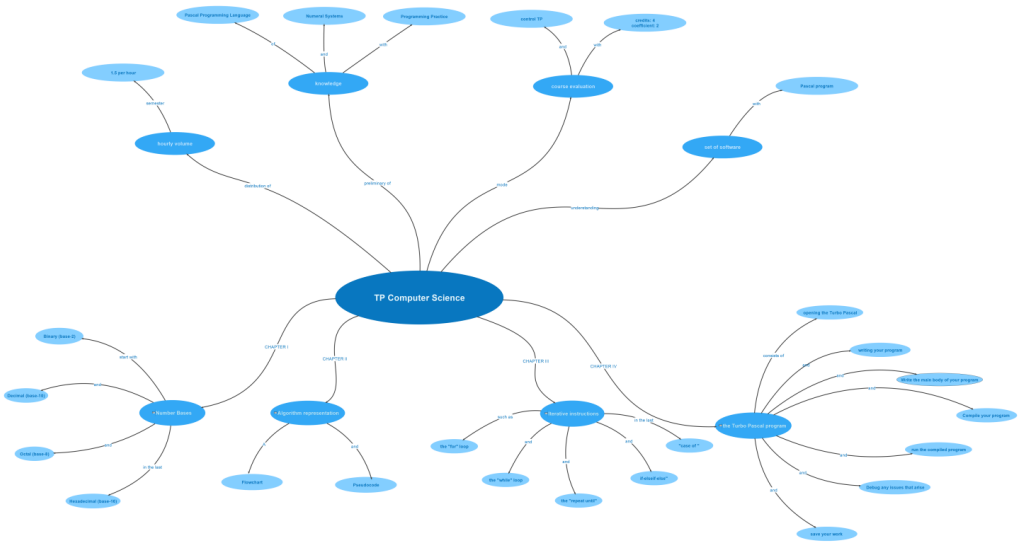
6.a. **Evaluating Programming Performance:**

· Debugging: Identifying and fixing errors (bugs) in the code to ensure the software behaves as expected.

·6.b. **Comprehensive System Review:**

· Through diligent inquiry and critical reflection, we invite readers to embark on a journey of intellectual exploration and discovery.

Introduction



Graphique 1 Map of computer sciences module modu

1[1]In our introductory discourse on Computer Science Essentials, we embark on a scholarly endeavor to unravel the intricate fabric of fundamental principles and essential concepts that define the realm of computer science. Through meticulous inquiry and rigorous exploration, we endeavor to shed light on the foundational elements that underpin this dynamic field, catering to the diverse needs and aspirations of learners at all stages of their academic journey*[1]*[1].

Our discourse commences with an exploration of the historical evolution and contemporary significance of computer science, delving into its theoretical underpinnings and practical applications. Through a critical lens, we examine the intricate interplay between computational theory and real-world problem-solving, seeking to elucidate the inherent complexities and inherent opportunities inherent in this burgeoning field.

Central to our discourse is a comprehensive examination of core concepts such as number systems, algorithms, and data structures. By distilling these intricate topics into digestible insights, we endeavor to empower learners with a nuanced understanding of the fundamental building blocks that form the bedrock of computational thinking*[2]*[2].

Furthermore, our discourse navigates the multifaceted landscape of programming languages and paradigms, emphasizing the cultivation of practical skills and problem-solving strategies. Through experiential learning and hands-on experimentation, we aim to equip learners with the requisite tools and techniques to navigate the dynamic terrain of software development and computational innovation.

As we draw our discourse to a close, we reflect on the dynamic nature of computer science and the imperative of lifelong learning in an ever-evolving technological landscape. By fostering a culture of inquiry and intellectual curiosity, we seek to inspire learners to embark on a journey of continuous discovery and innovation, poised to confront the myriad challenges and opportunities that lie ahead in the boundless expanse of computer science*[3]*[3].

In summary, our discourse serves as a scholarly exposition into the realm of Computer Science Essentials, offering a comprehensive overview of its foundational principles and practical applications. Through diligent inquiry and critical reflection, we invite readers to embark on a journey of intellectual exploration and discovery, as we unravel the mysteries and marvels of the digital age*[4]*[4].

I Overview of Computer Science and Programming

1. Introduction

This chapter serves as a gateway to understanding the fundamental principles and essential concepts that underpin the field of computer science, while also introducing the core elements of programming. Through a systematic examination of hardware, software, and programming environments, learners are introduced to the key components and processes involved in computational thinking and problem-solving. By delving into the intricacies of hardware components, such as *CPUs* and memory, alongside software systems and programming languages, learners gain insights into the inner workings of computing systems. Moreover, this chapter sets the stage for subsequent explorations into more specialized topics within computer science and programming, laying a solid foundation for further study and application.

2. Computer Science

Définition :

Computer science (*CS*) stands at the intersection of theory and practice, encompassing the study of computational systems, algorithms, and their applications. It examines the theoretical foundations of computing, such as automata theory, complexity theory, and algorithms, while also addressing practical considerations related to hardware, software, and programming*[3]*[3].

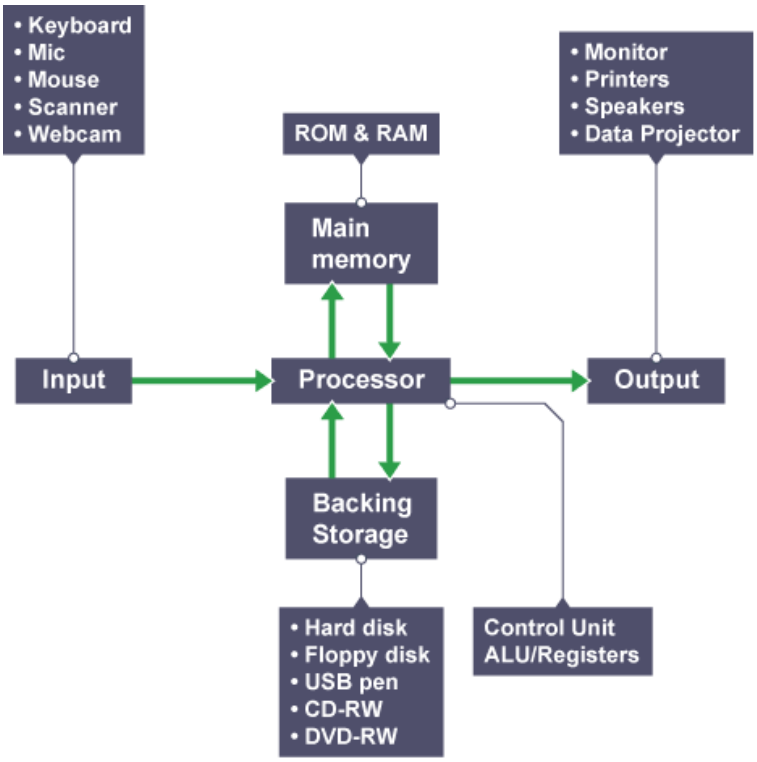


Figure 1.1: Diagram illustrating the components of a computer system, including hardware, software, and input/output devices.

2.1. Hardware

Définition :

[1]Hardware comprises the physical components of a computer system, each playing a crucial role in its operation*[1]*[1].

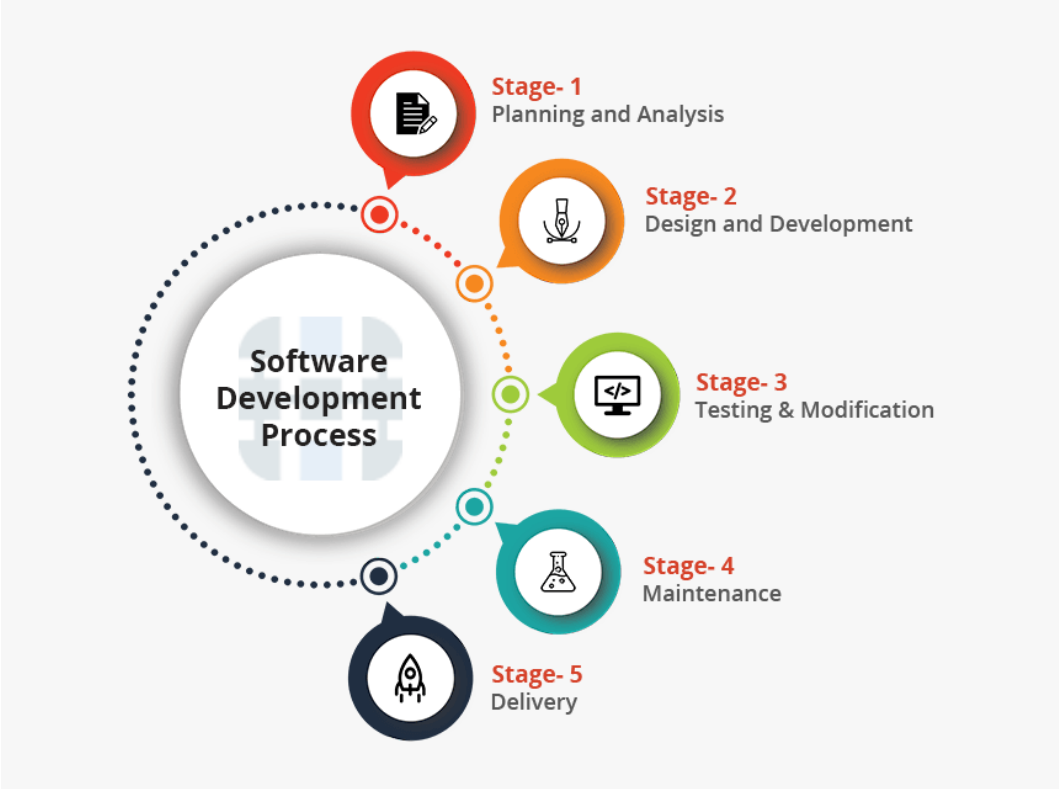


Figure 1.2: Overview of the software development lifecycle, including requirements analysis, design, implementation, testing, and maintenance phases.

*HW*Key components include:

- Central Processing Unit (*CPU*): Often referred to as the "brain" of the computer, the *CPU* executes instructions stored in memory*[2]*[2].
- Memory: Also known as *RAM* (Random Access Memory), memory temporarily holds data and instructions that the *CPU* needs to access quickly*[3]*[3].
- Storage Devices: These devices, such as hard disk drives (*HDDs*) and solid-state drives (*SSDs*), store data persistently even when the computer is turned off*[1]*[1].
- Input/Output (*I/O*) Devices: These include peripherals such as keyboards, mice, monitors, and printers, allowing users to interact with the computer*[3]*[3].
- Networking Hardware: Facilitates communication between computers and other devices, enabling data exchange over networks*[2]*[2].

2.2. Software (SW)

Définition :

Software encompasses the programs, applications, and data that run on a computer system. It includes operating systems, utility programs, compilers, interpreters, and applications. Software can be divided into two main categories: system software and application software*[4]*[4].

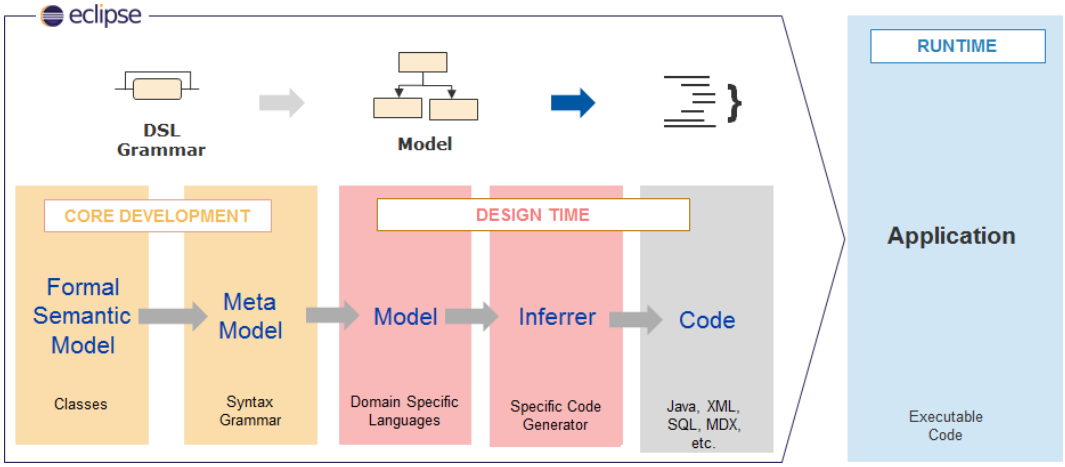


Figure 1.3: Schematic representation of a programming environment, highlighting the components such as IDEs, text editors, compilers, and debuggers.

It can be categorized into:

- System Software: Includes operating systems (e.g., Windows, macOS, Linux), device drivers, and utility programs (e.g., antivirus software, disk utilities) that manage computer hardware and provide essential services*[2]*[2].
- Application Software: Consists of programs designed for end-users to perform specific tasks, such as word processors, web browsers, and video editing software*[1]*[1].

3. Exercice : Exercise 1

- **Question:** Explain the difference between system software and application software with examples.

4. Exercice : EXERCISE 3EXERCIEX

Given a simple problem, choose an appropriate programming language and development environment, and justify your choice.

5. Programming Environment (PE)

Définition **:**

A programming environment provides the tools and resources necessary for developing software applications. This includes integrated development environments (*IDEs*), text editors, compilers, debuggers, and version control systems*[3]*[3].

Common components include:

- Integrated Development Environments (*IDEs*): Software applications that provide comprehensive tools for software development, including text editors, compilers, debuggers, and project management features*[2]*[2].
- Text Editors: Programs used for writing and editing source code, with features such as syntax highlighting and code completion*[2]*[2].

- Compilers: Software that translates source code written in a high-level programming language into machine code that can be executed by the computer[1][1].
- Debuggers: Tools used to identify and fix errors (bugs) in software code[2][2].
- Version Control Systems: Software tools that track changes to source code files, enabling collaboration among developers and maintaining a history of revisions[3][3].

Language	Paradigm	Popularity (2022)	Common Uses
Python	Multi-paradigm	High	Web development, data analysis, automation
Java	Object-oriented	High	Enterprise software, Android app development
C++	Multi-paradigm	Moderate	Game development, system programming
JavaScript	Multi-paradigm	High	Web development, front-end scripting
C#	Object-oriented	Moderate	Game development, Windows app development

Comparison of Programming Languages

5.1. Design

Définition :

The design phase of software development involves conceptualizing the structure and functionality of a program. This includes defining requirements, creating diagrams (such as flowcharts and UML diagrams), and designing algorithms and data structures[4][4].

including:

- Requirements Analysis: Gathering and documenting the functional and non-functional requirements that the software must meet.
- System Design: Defining the architecture, components, and interactions of the software system.
- Algorithm Design: Developing algorithms to solve specific computational problems efficiently.
- Data Structure Design: Designing data structures to organize and manage data effectively[4][4].

Operating System	Kernel Type	Market Share (2022)	Common Uses
Windows	Monolithic	High	Desktop computing, enterprise servers
macOS	Hybrid	Moderate	Desktop computing, creative work
Linux	Monolithic	High	Servers, embedded systems, development

Comparison of Operating Systems

5.2. Program Production

Définition :

Once the design phase is complete, the program production phase begins. This involves writing code according to the design specifications, testing the code for errors (bugs), debugging and fixing any issues, and documenting the code for future reference[4][4].

This process involves:

- Writing Code: Translating the design specifications into executable code using a programming language.
- Testing: Validating the functionality and correctness of the software through various testing techniques, such as unit testing, integration testing, and system testing.
- Debugging: Identifying and fixing errors (bugs) in the code to ensure the software behaves as expected.
- Documentation: Writing documentation that describes the software's functionality, usage, and implementation details, aiding in future maintenance and understanding*[5]*[5].

Component	Description
Central Processing Unit (CPU)	Executes instructions and performs calculations.
Memory	Temporary storage for data and instructions accessed by the CPU.
Storage Devices	Persistent storage for data, including hard disk drives (HDDs) and solid-state drives (SSDs).
Input/Output Devices	Peripheral devices for interacting with the computer, such as keyboards, mice, and monitors.
Networking Hardware	Facilitates communication between computers and other devices over networks.

Hardware Components of a Computer System

6. Exercice : EXERCISE 4EX

Analyze the advantages and disadvantages of different data structure designs for a given problem.

7. Exercice : EXERCISE 5EX

Create a detailed flowchart for a software application that involves multiple functionalities such as input processing, data storage, and output generation.