UNIT 4: PROGRAMMING

Learning Aim A: Examine the computational thinking skills and principles of computer programming

BY

ARIYA BAYAT

YEAR 12

WALWORTH ACADEMY

Unit 4: Programming

Name: Ariya Bayat
Coursework: Concepts of Programming

Contents

Name: Ariya Bayat
Coursework: Concepts of Programming

List of Figures

Name: Ariya Bayat
Coursework: Concepts of Programming

Chapter 1

Background and Motivation

Programming is seen as an essential tool for the development of technology, especially, in our modern-day era, using languages to lay out algorithms to achieve specific results. This skill has brought technology to life with its creations of mind-blowing but intimidating software and hardware.

The history of programming begins in the 9th century, when a programmable flute was invented by the Persian Banu Musa Brothers, as described in the 'Book of Ingenious Devices', along with many other programmable instruments. Next, before data storages existed, the concept of storing data on external punched cards was invented by Herman Hollerith in the 1880s for the use of 'programming' cloth-making machinery. Finally, evolving to the modern era, Computer programming languages advanced from low-level to high-level: machine code (1883), a computer language based on machine language like binary code which can execute tasks directly by the CPU, to Assembly language (1940s), which uses text formats with abbreviations and intelligible names for keywords, to high-level languages (1950s) making the development of programs simpler and more understandable.

Today, computer programming can be seen used everywhere in the modern society, shaping up our world with useful technology such as personal computers like mobile phones, laptops and software like Windows, MacOS. As programmers from different industries develop and share their creations to the world, programming software and languages continue to advance along with new software designing concepts being used. It is important to understand how these methods work for a programmer to improve the development and overall quality of their software. The principles of computational thinking and software design are examples of skills that are used by developers to make problem and process solving in software creation more efficient and effective.

Name: Ariya Bayat
Coursework: Concepts of Programming

Chapter 2

P1 | Computational Thinking Skills

2.1 Introduction

Computational thinking has proved to be an essential skill for tackling big problems and processes in the development of software applications. This method is made up of four main stages, including: decomposition, pattern recognition, abstraction, and algorithm design. In this chapter, we will be looking into these principles of computational thinking in detail to understand how to use these processes with key steps and how they are applied in finding solutions to problems during software development.

2.1.1 Decomposition

Decomposition is the process of breaking down complicated problems or processes into smaller and simpler parts. This is a technique used in everyday life activities. For example, when you go to school, you decompose the big task into smaller tasks in order: firstly, you wear your school uniform and leave your house, next you take public transport to your school, then you attend to your lessons, and etc.

In software engineering, this is used to break down big problems into minor problems for a more manageable and effective approach to solving the whole problem. This process contains four key steps that programmers use to help them understand the problem more clearly and plan out how they would go about building the program and solve the issues in the 'problem domain', the area where the problem is located.

1. 'Identifying and Describing Problems or Processes'
2. 'Breaking Down Problems and Processes into Distinct Steps'
3. 'Describing Problems or Processes as a Set of Structured Steps'
4. 'Communicating the Key Features of Problems or Processes'

2.1.2 Pattern Recognition

Pattern recognition is the process of spotting things that are common between decomposed problems or processes. Patterns are seen everywhere in our daily life. For instance, we see patterns of how our days are organised in school: school day starts at a specific time, our school day is broken up into several lessons, each lesson consists of one or two periods, etc.

In software engineering, we use pattern recognition to solve and predict future problems. This method is based on five key steps. These key steps help software developers to find solutions for repeating patterns.

1. 'Identifying Common Elements in Problems or Processes'
2. 'Identifying and Interpreting Common Differences in Problems or Processes'
3. 'Identifying Individual Elements within Problems or Processes'
4. 'Describing Patterns that have been Identified'
5. 'Making Predictions based on Identified Patterns'

2.1.3 Abstraction

Abstraction is the process of hiding what happens behind the scenes of patterns found in problems or processes. We see this done with everyday objects, like ATM machines where all the complexity of the process and problem solving are hidden from the user.

In software engineering, we use abstraction to hide irrelevant information from users. This method concerns with only two key steps used by software developers to get rid of any unnecessary parts of a problem.

1. 'Identifying Information Required to Solve the Problem or Process'.
2. 'Filtering out Unnecessary Information'

2.1.4 Algorithm Design

Algorithm design is the process of laying out a detailed step-by-step set of instructions on how to solve problems or processes. We follow algorithms every day without thinking, like when you go to the gym: firstly, you wear your gym outfit, next you take public transport or walk to the gym, then you use your membership card to have access to the gym, and etc.

In software engineering, we use algorithm design to describe solutions to problems in our software application. This method consists of three key steps used by developers to have a clear understanding of how solutions to problems work.

1. 'Understanding the Problem or Process'
2. 'Identify the Inputs, Processes, Data Storages, and Outputs'
3. 'Collect your Notes'

Name: Ariya Bayat
Coursework: Concepts of Programming

Chapter 3

P2 | Programming Languages

3.1 Introduction

Computer programming is a way of giving computers instructions that is used for developing software. Every software application requires some form of programming. There are many different programming languages (C#, Python, Java) that are used for different types of applications. In this chapter we investigate the principles and the elements of programming, including: the different types and styles of programming languages. We will also be comparing these programming languages to see which one is best for developing achieving different types of software and tasks.

3.1.1 Programming Language Types

There are two different types of programming languages, low-level and high-level languages.

Low-level languages are easily understood by the computer, as some use binary notation. This type of programming language requires the developer to know more about the structure of the processor. It is used for creating programs that concerns mostly with the architecture and hardware of a computer. Examples of low-level programming languages include assembly, machine code, etc.

High-level languages, however, use more natural language, making it easily understood by developers. This type of programming language requires the developer to know more about the keywords of the language. It is mostly used for creating problem-orientated programs. Examples of high-level programming languages include C#, Java, Python, etc.

3.1.2 Programming Language Styles

There are also three different styles of programming: object-orientated, procedural, and event-driven that are suitable for specific types of programs.

3.1.2.1 Object-Orientated Programming

Object-orientated programming concerns with programs that use concepts like objects, classes, and abstraction. Objects have its own properties that can be tampered with in its code. For instance, you can change a car's: colour, scale, position, etc. OOP is used for applications that implement real-world entities. Examples of OOP languages include C#, Java, C++, etc.

3.1.2.1.1 C#

C# is a modern, high-level programming language developed in 2000 by Anders Hejlsberg at Microsoft as a rival to Java. It was developed because Sun Microsystems did not want Microsoft to make changes to Java, so Microsoft developed their own language instead. C Sharp's popularity has grown quickly since it was first created and is now conquering as one of the most popular programming languages in the world. This language can be used to create almost anything but is mostly strong at building Windows desktop applications and games on game engines, like Unity Engine and Unreal Engine.

3.1.2.2 Procedural Programming

Procedural programming is more for programs that follows, in order, a set of commands. Procedural programming also uses concepts of procedure calls, such as functions which changes the selection of the program (see content 4.1.3 for more information on selection). Examples of procedural programming languages include: Python, Java, C, etc.

3.1.2.2.1 Python

Python is a high-level programming language developed in 1991 by Guido Van Rossum. Its first released version was Python 0.9.0, which included features like classes, lists and strings. In 2000, Python 2 came out with cycle-detecting garbage collector for memory management and Unicode support. Then, in 2008, Python 3 was released with the most noticeable change being the way print statements works, as in Python 3 the print statement has been replaced with 'print ()'. This language is mostly used in developing desktop GUI applications, websites and web applications.

3.1.2.3 Event-Driven Programming

Event-driven programming is suitable for programs that uses events to determine the flow of the program. Events such as user inputs, like mouse clicks and key presses are considered. Examples of event-driven programming languages include: Visual Basic, Visual C++, Java, etc.

3.1.2.3.1 Java

Java is a high-level programming language developed in 1995 by James Gosling and released by Sun Microsystems. Java is known as the most popular programming language with over 3 billion devices running java programs. Since the first version of Java (JDK Beta), there were many updates with features, making the programming language more and more useful. The current version is Java 13 released on September 2019 with important features such as Text Blocks, Switch Expressions Enhancements, and Socket API Reimplementation. Java is mainly used for creating Android applications. However, this language can also be implemented well in game applications, as seen with Minecraft.

3.1.3 Comparing Programming Languages

When choosing a programming language, it is important to consider many factors to ensure that it can achieve what you want when developing your program.

One major factor to consider is its strengths and weaknesses, like its ability to handle data or create websites. If you want to be a data analysist, then may want to learn Python as it is a procedural programming language, which is important for going through tables and databases like in BigQuery to look for specific data. Python also supports graphs where you visually display your data on.

If you want to be a game developer, then C# would be a good choice as it is an OOP language, which is crucial when making video games. Also, Unity, the most popular game engine only supports C#, making it the centre of attention for games development.

And, if you want to be an app developer, then Java would probably be the best choice as it is an event-driven programming language, which is key for triggering events by the user's interaction with the GUI. It is also the official language for Android development in Android studio.

Another important factor to take into consideration when choosing a suitable language is its usability. Out of top ten of the easiest programming languages to learn, HTML is ranked number one with 13.3% of developers designating them so, Go being number ten with only 3.6%.

This is probably the case since HTML is a mark-up language, meaning that it consists of easily understood keywords, names, or tag that format the visual view of a page and the data it contains, like when setting an objects 'width', 'height' or 'color'.

Go on the other hand is a bit more complicated, as it isn't a mark-up language and is used for large distributed systems and highly scalable network servers.

Name: Ariya Bayat
Coursework: Concepts of Programming

Chapter 4

P2 | Using Computer Programming

4.1 Introduction

Once you have chosen your desired programming language, it is now important to understand how we can use it to make our software function. In this chapter, we will be diving deeper into programming languages to learn how we can use them to our advantage to create our software applications with constructs and techniques, including: data types, variables, constants. We will also be looking into logic, specifically propositional logic to see how we can use to conclude statements in programming.

4.1.1 Data types

While programming our software application, we mess around with values to do calculations. So, it is important to know the different types of values we may come across or want to store as variables or constants. The five main data types are: integers (int), floats (float), strings (string), characters (char), and booleans (bool).

Integers are whole number values like 1, 2, 3, etc. We use integers to solve mathematical problems and processes in programming that strictly involves only whole numbers.

Floats can be whole number as well as decimal values like 0.5, 1, 1.5, etc. We use floats for solving more complex mathematical problems and processes in programming.

Strings are text values, like “Hello world!”, “How are you?”, “He left the room.”, etc. We use strings to represent text in general, it can also contain spaces and numbers too like “He jumped 5 times”.

Characters are single character values, like ‘A’, ‘!’, ‘7’, etc. We use characters to identify a single character from a text of letters, symbols, or numbers.

Booleans are values that can represent 2 outcomes, like 1 or 2. We use booleans as a true or false value to conform whether a condition or statement in programming is true or false to trigger events.

Figure 4.1 below shows an example of an integer named “year” set to 12, float named “age” set to 16.67, string named “name” set to “Ariya”, character named “initials” set to ‘A’ and boolean data type named “alive” set to true in C#.

```
// Integer example
int year = 12;
// Float example
float age = 16.67;
// String example
string name = "Ariya";
// Character example
char initials = 'A';
// Boolean example
bool alive = true;
```

Figure 4.1: Example of an integer, float, string, character and boolean data type (C#)

4.1.2 Variables and Constants

When handling software and processes, we often want to keep certain data for later use, this is when data storing can come in handy. You can use data storages to hold data that you can later refer to by its addressable name. There are two types of data storages, variables and constants.

A variable is an addressable value that can be tampered with overtime. A real-life example of a variable that changes overtime is our age, which increases by one every year.

A constant is an addressable value that will remain the same and cannot be changed overtime. A real-life example of a constant that never changes overtime are the components that a plant needs to survive such as water, air, nutrients, etc.

Figure 4.2 below shows an example of a public integer variable named “age” that is set to 16 and a private string constant named “plantNeeds” that is set to “water, air, nutrients, sunlight, and soil” in C#.

```
// Variable example
public int age = 16;
// Constant example
private const string plantNeeds = "water, air, nutrients, sunlight, and soil";
```

Figure 4.2: Example of a variable and constant (C#)

4.1.3 Programming Constructs

All programs are designed using common building blocks, known as programming constructs. These constructs form the pattern of all software. The three main programming constructs are sequence, selection and iteration.

Sequence is the order in which statements are executed. The sequence of a program is important as it is crucial that the tasks are handled in the correct order, otherwise the software would not function properly, which would fail the user requirements (see content 5.1.3.1 for more information on user requirements). For example, if two values are adding to each other, then the total value should be given after, otherwise the user wouldn’t see the result.

Selection is the path that the program is running based on conditions. The selection of a program is important as it is used to trigger events with conditional statements such as ‘if’ or ‘else’. For example, if the user presses a button on the application, then the path of the program should switch to where interaction with the button is handled, or the user may not get their response.

Figure 4.3 below shows an example of conditional statements in C#, including an ‘if’ statement checking if variable ‘age’ is over 17. If this is true, then a message will be displayed to the user: “You can drink!”, else a different message will be displayed: “You can’t drink!”.

```
// Conditional statements example
if (age > 17)
    Console.WriteLine("You can drink!");
else
    Console.WriteLine("You can't drink!");
```

Figure 4.3: Example of an if and else statement (C#)

Iteration is the repeated execution of section of code when a program is running. The iteration of a program is used to execute repetitive statements in a loop with loop statements such as 'while' or 'while not' to make processes in scripts more robust and efficient. For example, if the user sends files to another person with a software, then the program would check every single file for signs of harmful content before sending them altogether to the recipient repetitively until all files are searched. There are two types of iteration: count-controlled and condition-controlled iteration.

Count-controlled iteration are loops that are repeated for specific number of times. Count-controlled iteration loops are declared with 'for' loops.

Condition-controlled iteration are loops that are repeated until a condition is met. Condition-controlled iteration loops are declared with 'while' loops.

Figure 4.4 below shows an example of a for and while loop statement which both will keep displaying the message "You can't drink!" to the user until the integer 'age' is 18 or greater.

```
// For loop example
for (int i = age; i < 18; i++)
    Console.WriteLine("You can't drink!");
// While loop example
while (age < 18)
{
    Console.WriteLine("You can't drink!");
    age++;
}
```

Figure 4.4: Example of a for loop and while loop (C#)

4.1.4 Functions and Arrays

Like mathematics, when solving a problem in programming there are many different solutions you could use. There are many other techniques you can use to make your solutions much simpler and more robust including functions and arrays.

A function is a selection of code with an addressable name that can be referred to and called out, much like a variable or constant but instead with a group of code. We use functions to carry out specific tasks in a more manageable manner. We can also use functions to create local variables in the parameters section, which can be accessed only within the function while global variables created outside of the function can be referenced inside and outside of the function. Functions can make programming easier as we can simply use the name of the function to call out instead of constantly entering it to execute a chunk of code.

An array is a list of individual variables, also known as elements, with the same data type. We use arrays to prevent repetitive data storages. We can also use 2D and 3D arrays (vectors) to represent

the x, y and z axis of an object to mess around with its coordination. Arrays can make programming easier as we can simply refer to an element or axis from an array or vector other than referring from a long list of messy variables.

Figure 4.5 below shows an example of a three-dimensional string array created inside the parameters of a function that adds three sections of the message: “you “, “can “, “drink!” to the three elements of the array, adding up and outputting the full message “you can drink!”. This function is called out below the function.

```
// Function and array example
void DisplayMessage(string[] message = new string[3])
{
    message[0] = "You ";
    message[1] = "can ";
    message[2] = "drink!";

    Console.WriteLine(message[0] + message[1] + message[2]);
}

DisplayMessage();
```

Figure 4.5: Example of a function and array (C#)

4.1.5 Logic in Programming

Logic is when you declare statements to then allow the machine to come up with the outcome of those sentences to whether it is valid (true) or invalid (false) with a set of rules. This is used by programmers to come up with a conclusion for a statement to figure out if it is simply true or false. There are many types of logic that can be used for problem solving in programming, including propositional logic.

4.1.5.1 Propositional Logic

Propositional logic is logic that uses propositions and conditions to come up with a conclusion of if a statement is true or false. Propositions are statements that can either be true or false and only have one value. Propositional logic can be seen used in programming with conditions such as if and else statements.

When using propositional logic, you can use variables to represent statements, combine statements with logical operators like ‘and’, ‘or’, and compare statements with comparison operators like ‘>’, ‘<’, ‘=’ and ‘≠’. Once you have created your condition, you can evaluate your statements with truth tables which consists of different types of outcomes based on what exactly you are trying to evaluate.

There are also two different types of propositional logic, soundness and completeness: soundness is when you can’t prove that anything is invalid (false), while completeness is when you can prove anything that is valid (true).

Table 4.5 below shows an example of a truth table that evaluates whether statement “A” and “B” are both (“A^B”) true.

Name: Ariya Bayat
Coursework: Concepts of Programming

A	B	A^B
T	T	T
T	F	F
F	T	F
F	F	F

Table 4.1: Example of a truth table

Name: Ariya Bayat
Coursework: Concepts of Programming

Chapter 5

P3 | Software Design

5.1 Introduction

It is important that a software application is designed to meet the user's requirements for their experience. In this chapter, we will cover the principles of software design to see how they are implemented to create high-quality applications. We will also be looking into the uses of software applications to understand for what purposes users use them.

5.1.1 Purpose of Software Applications

There are many reasons for why people use software applications, like for entertainment or productivity. Applications in stores are categorised into different purposes like in the Google Play Store: 'Shopping' apps like Amazon or 'Social' apps like In.

Online shoppers use Amazon because it known as the biggest online shopping app in the world. It offers a wide range of products, ranging from 'Home & Garden' and 'Office Products' to 'Video Games' and 'Music'. In 2005, Amazon announced the creation of the 'Amazon Prime' subscription. This membership offers members features such as fast shipping, streaming of movies, TV shows and music, unlimited reading, etc.

Social media users use Facebook because it known as the biggest social media app in the world. You can follow people to communicate and share with them more easily. Facebook also released 'Facebook Messenger', a messaging app where you link chat and call with your friends linked to your Facebook. You can also use the Facebook Messenger app for creating group chats where you can invite groups of friends to have conversations in a single chat room.

5.1.2 Principles of Software Design

There are four main principles that designers use to handle complex problems in software designing, including: problem partitioning, abstraction, architecture, and modularity. We will be looking into these rules in detail to understand how they are useful for tackling significant problems in software applications.

5.1.2.1 Problem Partitioning

Problem partitioning is a method used by designers in software design to divide a big problem into smaller problems, like decomposition. This technique has many of its benefits when used in the design of software applications; Problem partitioning can make software applications easier to understand, debug, maintain, adjust, and develop. (See content 2.1.1 for more information on decomposition in computational thinking.)

5.1.2.2 Abstraction

Abstraction is another process in computational thinking, used by designers in software design to hide any internal processes from the user, only presenting relevant information, the user should only be

able to have access to the interface. There are two common types of abstraction, data, and functional abstraction. (See content 2.1.3 for more information on abstraction in computational thinking.)

Data abstraction is the process of hiding how data is handled, such as how inputs are handled in the program as the user does not need to know about the process.

Functional abstraction is the process of hiding the functionality and how tasks are done, such as how buttons on the GUI can direct the user to a different page as the user should not know 'how' things function but just that it just does.

5.1.2.3 Architecture

Architecture is a skill used by designers in software design to focus on the components of a software's structure. This technique is very useful is software designing as you can create diagrams of the structure to highlight the relationships between elements and their properties. Architectural design can also be used to illustrate any inputs, processes, storages, and outputs that would exist in the program. This is an important principle in software design as it can help you with decision-making on how you can deal with any problems or risks that may occur in the system.

5.1.2.4 Modularity

Modularity is a routine used by designers in software design to divide a system into smaller addressable parts known as modules to be developed individually. It is important that these modules: are well defined and useable in other applications, are easier to use than to build, and can be compiled and saved in the library separately. Once the modules are created, they are then integrated together as a whole program to meet the user's requirements.

5.1.3 User Requirements

It is essential to meet with the user's needs to produce high-quality software applications, including maintainability, portability, reliability, robustness and usability.

5.1.3.1 Maintainability

Maintainability is how maintainable a software application is. This refers to the ease of the program being modified by developers at any time. The simplicity of the software code and structure may increase maintainability. Software applications need to be maintainable so that they are stable when new implementations are added by developers.

The principles of software design can prove to be useful for ensuring software maintainability: problem partitioning can be used to simplify the whole problem down into smaller and more manageable parts like where the problem domains are located, abstraction can be used to hide processes like how updates are implemented into the application, architecture can be used to plan out how and where these implementations will be taking place, and modularity can be used for dividing the whole program into multiple modules like one that ensures the software's maintainability during any changes made by developers.

5.1.3.2 Portability

Portability is how portable a software application is. This is the measure of how many different computer platforms this application is targeting. API level is essential for android platforms as it identifies the framework of the device. Software applications need to be portable so that it is seen by a wider range of audience.

The principles of software design can prove to be useful for ensuring software portability: problem partitioning can be used to simplify the whole problem down into smaller and more manageable problems like the difficulties that might be faced when targeting certain platforms, abstraction can be used to hide the processes like how the API level of the android device that the program is running on is detected, architecture is used for planning the different interfaces of different computer platforms, and modularity can be used for dividing the whole program into multiple modules like one that detects the API level of the android device the program is running on for a more effective approach.

5.1.3.3 Reliability

Reliability is how reliable a software application is. For instance, downtime can make software unreliable as users can lose progress as the application becomes unavailable. Cloud storages are known to be very useful for backing up data, which can restore a user's progress. Software applications need to be reliable so that users can use them without worrying about losing their work.

The principles of software design can prove to be useful for ensuring software reliability. For example, for downtime issues: problem partitioning can be used to simplify the whole downtime problem into smaller tasks, abstraction can be used to hide processes like how the program connects to the cloud storage for preparation of downtime, architecture can be used for planning the connections between elements like the software and the cloud server, and modularity can be used for dividing the whole program into multiple modules like one that can connect to the cloud server for a more effective approach.

5.1.3.4 Robustness

Robustness is how robust a software application is. This is the measure of the program's code quality to ensure that tasks are executed as effectively and efficiently as possible. Pseudocode is a skill that acts like a sketch to help plan out the structure of the program's code. Software applications need to be robust so that there is a lower chance of users experiencing software failures.

The principles of software design can prove to be useful for ensuring software robustness: problem partitioning can be used to simplify the whole problem into smaller tasks like checking for repeating lines of code, abstraction can be used to hide processes like the whole functionality of the program, architecture can be used for planning the links between codes, and modularity can be used for dividing the whole program into multiple modules like one that runs when the users interacts with the UI.

5.1.3.5 Usability

Usability is how usable a software application is. This is the measure of how simple and easy it is for the users to use the application. Accessibility may be a factor to consider attracting more users with accessibility features like keyboard shortcuts and text-to-speech.

Unit 4: Programming

Name: Ariya Bayat
Coursework: Concepts of Programming

The principles of software design can prove to be useful for ensuring software usability: problem partitioning can be used to simplify the whole problem into smaller tasks like adding specific accessibility features, abstraction can be used to hide processes like how text-to-speech works, architecture can be used for planning where these features will be located in the application, and modularity can be used for dividing the whole program into multiple modules like on that handles all of the accessibility features.

Name: Ariya Bayat
Coursework: Concepts of Programming

Chapter 6

M1 | Impact on Software Design & Quality

6.1 Introduction

In this chapter we will be discussing how computational thinking skills can impact software design and its overall quality, including examples of high-quality software applications, analysing how they function with the four main steps of computation thinking.

6.1.1 Examples of Software

Computational thinking is a skill used by software developers to make the process of developing their software more easily and to initially produce a high-quality software by understanding how a computer thinks. The examples of software that is based on this way of thinking that we will be looking into is YouTube and Instagram.

6.1.1.1 YouTube

An example of a high-quality software application that is based on computational thinking is YouTube, the biggest video-sharing service where users can watch, like, share, comment, and upload videos.

When watching a YouTube video, decomposition is used to break down big problems, like adding to the 'like' counter when a viewer likes the video: first, the program checks if a connection between the software and the internet is established, next, the program looks for a matching like counter of the video in the database, then, the like counter is added up by one. This is when abstraction comes into play: this whole internal process is hidden from the user, instead showing them relevant information like highlighting the like button and displaying a message saying that the user has successfully liked the video.

When searching for a YouTube video, pattern recognition is used to identify common elements between problems or processes, like comparing the user's current input in the search box with a relevant and popular topic. For instance, if the user types the letter 'p' in the input field, a drop box will be shown below with a list of relevant topics that start with the letter 'p', like the most followed YouTuber on the platform, 'PewDiePie'. Algorithm design is considered here when the user searches for a video, they must: first, click on the search box, next, input the desired topic, then, click search or displayed relevant topic from the drop box.

6.1.1.2 Instagram

Another example of a high-quality software application that uses computational thinking is Instagram, an American photo and video-sharing social media networking service.

When looking at someone's profile on Instagram, decomposition can be seen used when the user follows other people, adding to the user's 'following' counter: first, the program checks if a connection between the software and the internet is established, next, the program looks for a matching account of the person they are following in the database, then, the follower's following counter is added up by one. This is when abstraction comes into play: this whole internal process is hidden from the user,

instead showing them relevant information like adding the followed account to the user's following list and highlighting the following button.

Once the user has followed their desired account, any recent activity from that account will be listed and displayed on your home page, where you can view all photos and videos of the people you follow. Pattern recognition is used here to identify common elements, like comparing the viewers of a photo or video who liked the content and the people the user is following. If a followed user likes a photo or video that you view, it will display their account on top of the 'likes list'. Algorithm design lays out a sequence of steps to carry out tasks, like sharing a video: first, the user presses the share button, next, write a caption that will be sent with the video, then, the user presses 'send' buttons of desired people and presses 'done' to share the video.

Name: Ariya Bayat
Coursework: Concepts of Programming

Chapter 7

D1 | Evaluation and Conclusion

7.1 Evaluation

We can see the endless potential of programming as programmers aim to reach new ground-breaking heights in technology as seen with AI. The principles of computational thinking and software design helps developers to improve their software development process and outcome, satisfying the users as their requirements are met. It is also important to know what type and style of programming language you will need and how we can use them to create your programs.

Computational thinking is a skill that enables us to tackle problems and processes in a more effective and efficient way. This skill is made up four main stages, starting with decomposition, which is when you break down big problems into smaller problems, helping us understand every bit of the problem. The second stage is pattern recognition, a method used for identifying any common elements between problems or processes, this can help us predict and solve future problems. The third stage is abstraction, a method used to hide any unnecessary information from the user like internal processes and calculations and only displaying the UI. And finally, the fourth stage of the principles of computational thinking is algorithm design, used for laying out a sequence of step-by-step instructions, allowing us to be fully clear on how to solve these problems or processes.

The principles of software design are also essential for handling problems or processes while designing a software. These principles consist of four main stages as well. The first two stages are problem partitioning and abstraction, which is much like decomposition and abstraction used in computational thinking. The third stage is architecture, important for analysing the components of the software's structure. And finally, the fourth stage of the principles of software design is modularity, where you divide a system into smaller addressable modules and develop them individually and integrate them when finalizing.

We also need to understand what makes a high-quality software meet with the user's requirements. There are five main factors to consider when designing a software, starting with maintainability, making sure that the system remains stable while developers make changes to the software. The second factor is portability, the more portable and accessible the application is, the wider the audience the app is targeting. The third factor is reliability, this is key for gaining the user's trust for them to make progress and use off the software. The fourth factor is usability, important for ensuring that the application is easy and simple to use, especially for people who need accessibility features. And finally, the fifth main factor to be considered when creating a high-quality software when meeting with the user's needs is robustness, checking if the software completes tasks as effective and efficiently as possible to prevent bugs or crashes in the program.

We use the principles of computational thinking and software design in the process of designing and finalizing our software, but it is also important to know what type and style of programming language we will need to use when writing our scripts. If your application interacts with the device's hardware structure, including the CPU then you would need to use a low-level language like assembly, but, if your application is a problem-based program then you should be more interested in using a high-level language like Java. Once you have picked the type of programming language you want to use, you must now choose its style to decide on your language. Object-orientated programming involves real

world entities with properties that can be tampered with. Procedural programming follows in a structured and ordered style of code execution. And event-driven programming concerns with events, changing the flow of the program. However, when choosing your initial programming language, you must also make sure that it can achieve what you want based on your speciality. For instance, if your goal is to be a data scientist then Python would be useful to learn due to its programming features like data graphs. If you want to be a game developer, then C# would be a good choice as it is a OOP language, which would be useful in game developing as games involve a lot of objects. And, if your goal is to be a mobile software developer, then you may want to use Java as it is suitable for event-triggering via the user's interaction with the GUI.

Once you have chosen your language; you must now dive deeper into software development and learn how to use it to start creating your software. A program is made up of statements and conditions, so it is important for a developer to learn the concept of programming constructs so that they can understand how processes are handled in programming: sequences, in what order statements are executed, selections, how the path of a program is changed based on conditions, and iterations, how repetitive tasks are looped. Data types are also used for solving problems and processes and they can also be stored into variables or constants for future use. Techniques like loops, arrays and functions can be quite useful for making code more neat, preventing repetitive statements and having the program running effectively and efficiently to fully meet with the user requirements.

7.2 Conclusion

The capabilities of technology with software and hardware continues to grow limitless as programming languages evolve, making development processes easier and yet making outcomes more extraordinary. This makes us wonder what the future would look like as creators satisfy users from different industries with bigger products than ever like the upcoming PS5 from the games industry.

However, technology has also become a major problem in our society as it makes us adapt to lazy habits. Thanks to technology, a person can spend days in their homes with loads of entertainment: Spotify providing over 50 million songs for us to listen to, Netflix streaming thousands of movies and TV shows, and hours of gaming with endless amounts of console, PC, and mobile games. Shopping and food deliveries can also be done online on websites, apps, and calls. This shows that technology has also impacted our lives negatively as these unhealthy activities can make us less engaged with the outside world and rely on our devices too much.