Unit 3 – Learning Outcomes: Programming Languages: History, Concepts & Design

1. **Describe some critical milestones in the development of programming languages.**

How we create and interact with code has changed due to several significant turning points in the evolution of programming languages (Juillet, 2022).

In today's world, computer programming is required to keep the systems and devices we use daily operating smoothly. Programming languages enable humans to interact with machines and make them perform necessary operations. Humans and machines process information differently, and programming languages are the key to bridging the gap between people and computers (HP, 2018).

In addition, HP (2018) lists the following significant advancements in computer programming languages:

- **1883:** The first programming language was developed in 1883 by Ada Lovelace and Charles Babbage. Lovelace wrote the algorithm for the Analytical Engine, the first computer program that computed Bernoulli numbers. She realised the importance of numbers, as they could represent more than just numerical values of things.
- **1949:** Assembly language was created to simplify machine code language, making it easier to program computers.
- **1952:** Alick Glennie created Autocode, considered the first compiled programming language that could be directly translated into machine code.
- **1957:** John Backus developed FORTRAN, a programming language for scientific, mathematical, and statistical projects.
- **1958:** Algol was developed as an algorithmic language and precursor to modern programming languages like Java and C.
- **1959:** Dr. Grace Murray Hopper created COBOL as a language capable of operating on all computer types.
- **1959:** John McCarthy created LISP, a programming language for artificial intelligence research. It is still in use today and can be used with Python and Ruby.
- **1964:** John G. Kemeny and Thomas E. Kurtz created BASIC, a programming language designed for students with limited technical and mathematical backgrounds, making computers accessible to a broader audience.

- **1970:** Pascal was created by Niklaus Wirth and named in honour of Blaise Pascal. The primary language used by Apple to construct its early software is simple to learn.
- **1972:** Smalltalk was developed by Alan Kay, Adele Goldberg, and Dan Ingalls to enable quick code changes for programmers.
- **1972:** Dennis Ritchie is credited with creating C, considered the first high-level programming language. Unlike machine code, it is closer to human language.
- **1972:** SQL was created for IBM by Donald D. Chamberlin and Raymond F. Boyce. This language was used to examine and modify database-stored data.
- **1978:** The math programming language MATLAB was created by Cleve Moler. Research and teaching are conducted using this language.
- **1983:** Objective-C was developed by Brad Cox and Tom Love as the primary language for creating Apple software.
- **1983:** Bjarne Stroustrup created C++, an extension of C, widely used.
- **1987:** Perl was created by Larry Wall as a scripting language for text editing to streamline the processing of reports.
- **1990:** Haskell was created as a functional programming language for handling challenging mathematical operations.
- **1991:** Python was created by Guido Van Rossum and is a condensed, simple-to-read programming language.
- **1991:** Visual Basic, developed by Microsoft, allowed programmers to select and modify code using a drag-and-drop process easily.
- **1993:** R was developed by Ross Ihaka and Robert Gentleman for statisticians to perform data analysis.
- **1995:** Java was developed by Sun Microsystems specifically for use with handheld devices.
- **1995:** Rasmus Lerdorf created PHP for Web development, which remains popular today.
- **1995:** Yukihiro Matsumoto created Ruby as a versatile programming language suitable for various tasks. It is extensively utilised in the creation of web applications.
- **1995:** JavaScript was developed by Brendan Eich to improve interactions on Web browsers.
- **2000:** Microsoft developed C# as a modern, object-oriented programming language with features inspired by C++ and Visual Basic. C# shares similarities with Java.
- **2003:** Scala was developed by Martin Odersky as a programming language that fuses elements of functional programming.
- **2003:** Groovy was developed as a Java offshoot by James Strachan and Bob McWhirter.
- **2009:** Go was created by Google to address the common problems in large software systems.
- **2014:** Swift was created by Apple as a replacement for C, C++, and Objective-C.

Today's computer programming languages are built on earlier ideas, some of which are still used or are the basis for new ideas. Due to the ever-expanding technology, modern languages will continue to be crucial to everyday life, making programmers' jobs easier (HP, 2018).

2. Outline some fundamental paradigms that define the different types of languages.

Bhumika_Rani (2018) states that an approach to problem-solving that makes use of programming languages, tools, and methodologies is known as the programming paradigm.

A programming paradigm represents the core coding approach and methodology, providing a conceptual foundation for creating and organising code in a language. It impacts how tasks are split, data structures are employed, operations are carried out, and activities are coordinated (Joy, 2022).

Programming languages are categorised into paradigms based on their characteristics. These paradigms include code structures and execution models. **Imperative** and **Declarative** are the two main programming paradigms. They further divide into additional paradigms with even more specific traits (Garcia, 2021).

As Garcia (2021) explains, the Imperative Paradigm is the oldest programming paradigm, focused on sequences of instructions that modify computer system states. It's based on von Neumann's architecture, which includes a processing unit, control unit, and memory. Imperative programming languages manipulate data in memory using statements and a sequential order for executing instructions. It has two popular derived paradigms: procedural and object-oriented. Furthermore, the Declarative Paradigm defines tasks for computer systems without specifying how to process them.

It provides "guides" on what tasks should be accomplished, resulting in a higher abstraction level and improved maintainability. The functional and logic paradigms stem from this approach.

Although there are more paradigms, the following are the most well-known and widely applied ones in programming, according to Garcia (2021):

1. **Procedural Programming:** Procedural programming breaks programs into subroutines for easier maintenance and modularity. Subroutines have names and parameters and can be called to perform a specific task. Some subroutines need to return results.
2. **Object-Oriented Programming:** Object-oriented programming is based on modular programming but models the real world. Classes represent real-world objects and can be instantiated as objects. Attributes store information about a class, and methods are procedures associated with a class that can produce internal changes or side effects.
3. **Functional Programming:** Functional programming involves using functions as data types, assigning them to variables and passing them as arguments to other functions. There are different categories of functions, such as first-class, pure, and high-order functions. First-class functions are elementary and can be assigned, passed, and returned. Pure functions have transparency and lack side effects. High-order functions receive or return functions and include examples like the map function.
4. **Logic Programming:** Logic programming uses formal logic to solve problems. Programs rely on a knowledge base of logic and rules. Facts are assertions about objects, while rules describe logical relations among facts. Queries consist of logic expressions and operators. Programs use an inference system to search for proofs and return solutions to questions. If there are no solutions, the program produces a false value.

3. Explain the key concepts that determine the operation of programming languages.

Programming languages are artificial languages used to control machines, especially computers. They use syntactic and semantic rules to determine structure and meaning. These languages facilitate communication for organising and manipulating information and expressing algorithms. There are thousands of programming languages, and new ones are created every year (cs.mcgill.ca, N.D.).

Computer programming is creating instructions that tell a computer what to do. All software adheres to certain programming principles, and skills can be improved by grasping different concepts (Indeed, 2023).

The essential principles of programming, according to Educative (N.D.), include the following:

1. **Variable Declaration:** Variables are containers that store data values in memory and are created using a declaration or keyword that varies across programming languages. Variable names are typically alphanumeric but can include special characters like underscores or dollar signs. They can hold values of any data type supported by the programming language, which may change during program execution.
2. **Basic Syntax:** Programmers must master the basic syntax of the programming language they are learning. Each programming language has its syntax. The principles defining a language's structure are called its syntax. It is nearly difficult to read or comprehend a computer language without syntax.
3. **Data Type and Structures:** Data types are categories of data. The most common ones are strings, Booleans (true/false), numbers (integers and decimals), characters, and arrays. Data structures are collections with operations for efficiently managing, organising and storing data. Common data structures include stacks, heaps, trees, linked lists, queues, arrays, tables, and graphs.
4. **Flow Control Structures (Conditionals and loops):** Flow Control Structures are vital components of computer programs, allowing them to make decisions. There are three types: sequential, selection, and iteration. Sequential is the fundamental execution of code statements, selection involves the computer deciding action based on test results, and iteration allows for code to be run repeatedly until a particular condition is no longer valid.
5. **Functional Programming:** Functions are containers that take in inputs and return outputs. Pure functions always produce the same output for the same information. Functional Programming utilises pure functions and avoids data mutation and side effects.
6. **Object-Oriented Programming:** Object-oriented programming (OOP) is a programming paradigm that uses objects to store data and methods to manipulate it. There are four fundamental principles of OOP: Inheritance, Polymorphism, Abstraction, and Encapsulation.
7. **Debugging:** The ability to debug is essential. It entails finding and fixing any problems, flaws, or 'loopholes' in one's code, both current and potential.
8. **IDEs and Coding Environments:** IDE stands for Integrated Development Environment. Programmers use it to write code and organise text groups. It includes code completion, compilation, debugging, and syntax highlighting. Some popular IDEs are Visual Studio Code, IntelliJ IDEA, NetBeans, and Eclipse.

4. Discuss key programming challenges and recommended best practices.

Coding is crucial in today's digital world for technology and software development. Skilled coders are in high demand due to rapid technological advancements. Anyone can become an experienced coder with the right mentality and essential abilities. (Geeks of Gurukul, 2023).

Novice programmers face various challenges, including lack of motivation, limited scientific knowledge, difficulty understanding abstract concepts, and tracking complex discussions. Previous research has shown that these challenges are in addition to coping with new programming languages and understanding instructions while coding (Saha & Mitra Thakur, 2022).

Code Signing Store (N.D.) explains that programming best practices exist to ensure codes are safe from attackers and easy to read, test, use, and maintain for authorised users. We write code not just for ourselves but for others who will need to read it.

Software development can be challenging due to project complexity, time constraints, and resource limitations. Tran (2021) has compiled a list of common challenges and solutions as follows:

1. **Lack of Management:** Clear guidance from project managers is crucial for effective software development. A concise project plan with outlined critical tasks and responsibilities and a timeline helps developers stay on track and avoid delays. Regular updates to the project plan ensure everyone is on the same page.

2. **Difficulty Estimating Time and Resources:** Estimating time and resources for software projects takes much work. Realistic timelines are essential for budget and time constraints. Breaking tasks into smaller chunks helps. Use tools like Toggl or Harvest to track effort. Set deadlines and have backup plans for potential problems.

3. **Lack of Resources for the Software Development Process:** Software development can be challenging due to limited budgets and resources. Projects are becoming more complex, requiring more time and money. Developers may

also face obstacles like limited access to testing computers, a shortage of engineers, or outdated technology. One way to overcome these challenges is by using free assets available online. Teams should communicate their resource needs to ensure they have the tools to produce quality products. It's also essential to prioritise features and look for cost-cutting opportunities to save time and money.

4. **Defining the Requirements of the Software Development Projects:** Defining software requirements is time-consuming and challenging. Clear and concise requirements are essential to ensure clarity and timely progress in the project. Holding discussions with customers and creating prototypes for feedback can provide transparency and efficiency in the development process.

5. **Miscommunication with Customers/Stakeholders:** Software developers face miscommunication among customers and stakeholders, leading to delays and a poor final product. To prevent this, establish regular communication channels, be clear about requirements, document everything, and ask questions for clarification.

6. **Strict Time Constraints:** Software developers often face strict time constraints and pressure to meet deadlines. To manage this challenge, companies should prioritise good time management by setting clear expectations and realistic deadlines from the start. Teams should also take breaks and build in time for unexpected events to avoid putting the project at risk.

7. **The Complexity of Software Projects:** Developers often face the challenge of complex projects with numerous dependencies and potential problems. To overcome this, they should break down the project into manageable tasks, establish a plan for handling issues, communicate closely with their team, and devise strategies for false starts to stay organised and on track.

8. **Finding Qualified Talents:** Finding qualified software development talent can be challenging due to a shortage of workers and high hiring costs. Companies can be more proactive by identifying needed skills, offering attractive job packages, promoting their company culture, and considering outsourcing or intern opportunities. Recruitment agencies and online platforms can also provide access to ready-made resumes for skilled professionals.

9. **Testing and Debugging:** Software testing teams often face challenges with testing and debugging code, including identifying and fixing errors. Developers can overcome these challenges by meticulous work, understanding the system and code they're working on, using automated debugging tools, and establishing a thorough testing plan before releasing code to production. This approach saves time and effort while ensuring error-free code.

10. **Maintaining the Competitive Edge:** To stay ahead in software development, companies must continually innovate and improve, monitor industry

developments, explore new technologies, and invest in R&D initiatives. This will help them stay competitive and better serve their customers.

5. **Explain what design patterns are and when to use them.**

Design patterns are pre-made blueprints to solve common problems in software design. They're not specific code but a concept for solving a problem. Patterns need clarification with algorithms, which define a set of actions to achieve a goal. A pattern is more like a blueprint with the exact implementation up to the programmer (Refactoring.guru, 2014). Therefore, using design patterns can improve software's readability, extensibility, and maintenance. They can also increase the effectiveness of the code and aid in avoiding typical mistakes (Silk, 2022).

Design patterns require creativity, attention to detail, and technical knowledge. A successful design pattern visualises and translates the finished product into a precise and accurate pattern. Design patterns provide reusable solutions for everyday problems in software design. They speed up development by giving well-tested paradigms that are flexible, reusable, and maintainable (GeeksforGeeks, 2015). As a result, Gang of Four (GoF) design patterns, divided into Creational, Structural, and Behavioural groups, are the basis for all other patterns, according to Dofactory (2018).

The three primary categories of design patterns, as outlined by Silk (2022), are as follows:

1. Creational patterns are used to create objects in a controlled way that suits the situation. These patterns help reduce complexity, increase stability, and improve code reusability. Examples of creational design patterns include Singleton, Factory Method, Prototype, and Builder.

2. Structural patterns help connect application components, particularly in more complex applications. Examples of structural designs include Adapters, Bridge, Decorator, and Facade.

3. Behavioural patterns facilitate effective interaction between components by defining responsibilities. Examples include Chain of Responsibility, Iterator, Mediator, Observer, and Strategy design patterns. The aim is to simplify procedures.

Design patterns can speed development, prevent issues, simplify maintenance, improve code quality, ease readability, and facilitate communication and collaboration between team members (Awad, 2022).

Programmers can use design patterns to improve code readability, maintainability, and extensibility, eliminate errors, and create more efficient code for common software design issues (Gill, 2023).

Smith (2023) explained that design patterns are essential for efficient, maintainable, and high-quality software solutions. They provide reusable solutions to common problems, promote good design principles, and establish a common language for developers to collaborate effectively. Using design patterns reduces effort and improves scalability, performance, and code readability. Naming conventions also help maintain consistency and prevent confusion in larger teams or codebases.

References:

Juillet, R. (2022). *The evolution of programming languages: from 1843 to today*. [online] Available at: https://www.bocasay.com/evolutions-trends-programming-languages/.

HP (2018). *Computer History: A Timeline of Computer Programming Languages | HP® Tech Takes*. [online] Available at: https://www.hp.com/us-en/shop/tech-takes/computer-history-programming-languages.

Bhumika_Rani (2018). *Introduction of Programming Paradigms - GeeksforGeeks*. [online] GeeksforGeeks. Available at: https://www.geeksforgeeks.org/introduction-of-programming-paradigms/..

Joy, A. (2022). *Types of Programming Paradigms*. [online] Pythonista Planet. Available at: https://pythonistaplanet.com/types-of-programming-paradigms/#:~:text=Let [Accessed 23 Aug. 2023].

Cocca, G. (2022). *Programming Paradigms – Paradigm Examples for Beginners*. [online] Available at: https://www.freecodecamp.org/news/an-introduction-to-programming-paradigms/#popular-programming-paradigms.

Garcia, V.F. (2021). *Imperative and Declarative Programming Paradigms | Baeldung on Computer Science*. [online] www.baeldung.com. Available at: https://www.baeldung.com/cs/imperative-vs-declarative-programming.

cs.mcgill.ca. (N.D.). *Programming language*. [online] Available at: https://www.cs.mcgill.ca/~rwest/wikispeedia/wpcd/wp/p/Programming_language.htm#:~:text=A%20programming%20language%20is%20an.

Indeed (2023). *6 Fundamental Programming Concepts (With Tips to Improve)*. [online] Available at: https://ca.indeed.com/career-advice/career-development/fundamental-programming-concepts#

Educative (N.D.). *What are the basic fundamental concepts of programming?* [online] Available at: https://www.educative.io/answers/what-are-the-basic-fundamental-concepts-of-programming.

Geeks of Gurukul (2023). *Skills Needed to Become a Competent Coder*. [online] Available at: https://www.linkedin.com/pulse/skills-needed-become-competent-coder-geeks-of-gurukul/ [Accessed 23 Aug. 2023].

Saha, B. & Mitra Thakur, G.S. (2022). Learning Programming: Challenges and Remedies. *SSRN Electronic Journal*. doi:https://doi.org/10.2139/ssrn.4213000.

Code Signing Store. (N.D.). *The Ultimate Programming Best Practices Guide*. [online] Available at: https://codesigningstore.com/ultimate-programming-best-practices-guide.

Unadkat, J. (2022). *Key Software Testing Challenges and Solutions*. [online] Available at: https://www.browserstack.com/guide/software-testing-challenges.

Tran, T. (2021). *10 Most Common Software Development Challenges*. [online] Available at: https://www.orientsoftware.com/blog/software-development-challenges/.

Refactoring.guru. (2014). *What's a design pattern?* [online] Available at: https://refactoring.guru/design-patterns/what-is-pattern.

Silk, J. (2022). *Why Software Development Design Patterns Matter For Your Business*. [online] Startechup Inc. Available at: https://www.startechup.com/blog/software-development-design-patterns/.

GeeksforGeeks (2015). *Introduction to Pattern Designing*. [online] Available at: https://www.geeksforgeeks.org/introduction-to-pattern-designing/.

Dofactory (2018). *.NET Design Patterns in C# and VB.NET - Gang of Four (GOF) - doFactory.com*. [online] Available at: https://www.dofactory.com/net/design-patterns.

Awad, H. (2022). *4 Things to Consider When Applying Design Patterns*. [online] Medium. Available at: https://betterprogramming.pub/4-things-to-consider-when-applying-design-patterns-46b9fcee4a59.

Gill, D.S. (2023). *Mastering Design Patterns: A Guide to Writing Cleaner Code*. [online] Medium. Available at: https://medium.com/@dawinderapps/mastering-design-patterns-a-guide-to-writing-cleaner-code-92a634313ba9 [Accessed 24 Aug. 2023].

Smith, G. (2023). *Best Software Development Practices - Comprehensive Guide*. [online] Mobile & Web App Development Company | USA, UK, Norway. Available at: https://itcraftapps.com/blog/mastering-software-development-a-guide-to-excellence-in-coding/ [Accessed 24 Aug. 2023].