Definition of Programming

Table of Content

1. Introduction

Programming has been living in innate human curiosity for centuries. As far as the 9th century was the first programmable device created by the Persian Banu Musa brothers (Meier, 2016). They described it as an automated mechanical flute player in their book, Book of Ingenious Devices (‘Book of Ingenious Devices’, 2021). The first computer program was written by Ada Lovelace (Puiu, 2021) working on Charles Babbage’s (O’Connor and Robertson, 1988) analytical engine which is generally considered as the first computer. Ever since, programming and computers have had achieved immense progress in technology, affecting all most all the functionalities of living on earth and beyond.

Computer programming is writing instructions to a computer using a language a computer can understand, to implement different functionalities and operations to perform a specific task. The following report includes discussions and descriptions about several areas related to the programming and software development process.

2. What is an Algorithm?

An algorithm is a well-defined and finite number of step-by-step solutions or a series of instructions to solve a well-defined problem. In computer programming context an algorithm is a procedure used to solve problems. This procedure should take an input, apply some actions on them and transform them into an output. The applications could be mathematical or logical yet the most significant parts of an algorithm are input, procedure, and output.

3. Process of building an application

3.1 Outline

Software development is the procedure to make software using computers and programming. This procedure generally involves six steps. Although the titles of the steps and the number of steps can be slightly varying according to the context, these six steps contain all the actions that need to be taken to create executable software from scratch. The steps are as below.

1. Analysis – A detailed analysis of the required software is performed here to identify the requirements of the client. Sometimes if the application is an innovative product, a brainstorm session would be held to identify out-of-the-box solutions and creative ideas of the team. Product managers and developers sit together and they all contribute their ideas and discuss to select the most appropriate. If the product is a client requirement, analysis of the requirement specification, defining use cases, project plan, timelines, the scope of the project, and the costs estimations will be made in this step.

2. Design – This is the step of building the architecture of the project. Based on the agreements made in the previous step, developing and validating the user interface and the prototype, representation of the requirements through diagrams and use cases, elaboration on the design, development of the prototype with the test model is done in this step.

3. Develop – Based on the finalized design models, the actual implementation is done by software developers. As the most time-consuming process it would be, project managers

manage this process and keep it under the quality required. The whole system is broken into pieces and every developer has their parts to implement. Developers have to do the developer testing at the implementation.

4. Integrate and Test – Once the development is completed and developer testing is done, quality assurance engineers test the system. There are various tests involved at different stages of the integration of the system unit testing, smoke testing, stress testing, sanity testing, load testing, etc. Integration testing is performed to verify the overall functionality of the system after integrating is done. QA uses different frameworks and techniques for testing and writes tests cases. They report the bugs to developers to fix.
5. Deployment – Once the testing is passed it is time to deploy the system to production. After the customer accepts the product never versions might come with cyclic upgrading to the project.
6. Maintenance – Once a product is deployed to a customer environment, a maintenance team is assigned to fix the bugs that arise in use and do the customizations and feature developments according to customer requirements.

3.2 From writing the code to Execution

The following steps take place from the start of the writing of the code to the execution of it in a production environment.

- Understand the problem that needs to solve is a major part of producing the right software at the end as this understanding becomes the basis for all the designs and code.
- Design an algorithm - Once the inputs and outputs are identified it is needed to design an algorithm to solve the problem. An algorithm is a plan to solve a problem and the plan can be of several levels. Therefore, first, design a high-level solution and refine it again and again.
- Draw a flow chart - A flow chart outlines the flow of information and processes visually hence it helps to create a coded solution. Using the flow chart developer can identify variables, errors, and finetune ideas. Based on the algorithm designed in the above step, a flow chart can be drawn using tools and accepted symbols.
- Write pseudo-code – A pseudo-code is a representation of how to implement the algorithm written in annotations and plain English without no programming syntax. It is a step-by-step outline of how the code should be, which can gradually convert into the programming language.
- Write code – Based on the written pseudo-code, the code is written using selected language and tools. Since a program is mostly in plain text a developer can use any text editor to write the code. But there are many tools and technologies built to elevate the coding and testing processes. Hence, we can choose a suitable integrated development environment (IDE) for this task rather than a text editor. There are so many IDEs in use, and generally, a development team sticks into similar tools within the team so in a team environment it is better to use the IDE and tools the team usually uses.
- Test and debug – Once the code is written, testing for issues takes place. Before quality assurance teams tests for any issues, developers do developer testing in their particular implementations. Once the code is pushed to QA environments, QAEs start testing. There are many types of QA testing and they all fall into two major categories as functional and non-functional. Within those, unit, integration, system, sanity, smoke, interface,

regression, beta/acceptance testing are considered functional testing. Performance, load, stress, volume, security, compatibility, install, recovery, reliability, usability, compliance, and localization testing are considered non-functional testing. Among all these testing, user acceptance testing (UAT) and beta testing have importance as they are done by the clients. The difference between UAT and beta testing is, UAT is carried out by one or two users while beta testing is done in the wild. Beta testing is the testing that happens in the real world with many input loads and so many user actions so it is testing whether the software is used in a mass environment.

- Release the software – Once all the testing is passed, the software is released to production. Building and deploying the code to production should be a managed process and all the development and testing teams should be ready to work on any issues that arise while the dev-ops usually taking care of servers and configuration management tasks.
- Maintenance – After successfully release the software to production, a maintenance team is assigned to fix bugs and develop customized featured required by the customer.

4. The Algorithm, code, and the programming language

4.1 Relationship between an algorithm and a code

An algorithm is a procedure to solve a problem. A program or a code expresses an algorithm. The same algorithm can be expressed in many different ways using many different programming languages. An algorithm outlines inputs, outputs, restrictions, doable and undoable tasks. The programmer's job is to understand the restrictions and create a path to solve the problem using the preferred programming language and tools. An algorithm is conceptual while a code is practical.

An algorithm is like a food recipe.

For example, imagine baking a cake. A baking student knows what she needs to make the cake but she has no idea how to do it as she's just started learning baking. So there comes the recipe. The recipe includes all the instructions, what to use in what amounts, how to mix them and when to put them in the oven. The recipe outlines the possibilities and loopholes so the baker can learn what to do and what not to while executing the recipe. Once in the implementation, the baker student executes the recipe as she prefers, replacing vanilla flavor with chocolate, and mixing chocolate chips instead of candied fruit.

Similarly, different programming languages can implement the same algorithm in different ways yet the outline, the inputs, and outputs are similar.

The following example shows a simple algorithm and its implementation using Java programming language.

1. Start
2. Read a, b
3. c = a+b
4. Display c
5. Stop

Algorithm

```
import java.util.Scanner;

public class Add{
    public static void main(String [] args){

    Scanner Ob1 = new Scanner(System.in);

    System.out.println("Enter value of a: ");
    int a = Ob1.nextInt();

    System.out.println("Enter value of b: ");
    int b = Ob1.nextInt();

    int c = a+b;
    System.out.println("Sum of given two numbers is: " +c);
    }
}
```

Program

4.2 How to choose a programming language

The selection of a programming language for a particular project depends on many factors. Two of the most important are development speed and execution speed. Following is a list of many such factors to consider when choosing a programming language.

1. Development speed.
2. Execution speed.
3. Robustness.
4. Developer availability.
5. Ecosystem and community.
6. The demand and industry trends.
7. Connections with other languages and ecosystems.
8. The age of the market.
9. The opportunity for self-improvement.

The discussion below is an analysis of the above factors of how to choose a programing language.

The time required to achieve the deadline of the project is an important fact when deciding the programming language. Some languages are easy to learn, faster when coding but takes time when debugging. Also, some languages have extensive support from tools and technologies while some are not. Memory-intensive languages are heavy on some systems while some languages are lightweight and quickly get executed. Tools offered by the languages can be open source or commercial thus affecting the budget of the solution. Also, if the software depends on a third-party application, execution speed is uncontrollable in that phase. Using cloud-based services makes every minute count in dollars. That implies how important is speed in choosing a language.

Robustness means strength. A program is correct if it accomplishes the task that it was designed to perform. It is robust if it can handle illegal inputs and other unexpected situations in a reasonable way(Eck, 2020). In simple words, if a program does not crash during execution and less possibility to show errors, it is called a robust program. The level of robustness depends on the situation and language must be chosen according to that.

When working on a large project with a team of developers it should always be a consideration to use a programming language the programmers are fluent in. So, the composition of available skills is also a factor.

Some languages have a huge ecosystem of supported libraries, tools, technologies, and a community of developers and researchers who continually work on them to make more and more sophisticated technologies using the language. When a new language is built upon an older ecosystem, it has support and the learning curve is smaller; while a brand-new language lacks both. Therefore, connection with other programming languages and ecosystems is also a big concern here. When choosing a language, big community support is always a plus point as it would support the speed development and troubleshooting with immensely helpful forums and discussions.

When software is being built, 99% time it's for business. So, the demand in the market and industry trend means a lot. With new waves of technological advancements, different languages are in the limelight from time to time.

Age of the market and opportunity for development is not much of a concern in an enterprise development team, but if someone decides to work on their project, these two factors are a must to consider because the developer's future career depends on these factors.

So according to all the factors described above, a software development team must first collect all the relevant data about the market, demand, execution speed, available resources (human and hardware/software), community support, future of the language, age of the market, etc. and analyze them to identify what language suits the best for their project.

5. Programming paradigms

- Procedural paradigm (PP) – Procedural programming divides the large program into smaller programs called functions. Each function has a specific task to carry out and any function can be called at any time. A function or a sub-program is a set of step-by-step actions to perform a task at hand. These functions work on the program data. Following are the key characteristics.

  Top-down approach – The main program is broken down into sub-programs.
  Data and functions are two different entities

Data flows freely among functions
One function can access the data of the other function by calling that function.

- Object-oriented programming – OOP maps the real-world objects to objects of the program. These objects make up the whole system. Key characteristics are as follows.

  Maps real-world objects into programming objects.
  Data and actions are in a one-unit call object.
  Encapsulation – making a self-contained module of data and functions which outside functions cannot access.
  Inheritance – Passing down the information inside classes through inheritance. Child classes inherit methods and data of parent classes under this.
  Abstraction - Ability to represent data at a very conceptual level without any details.
  Polymorphism – Procedures of objects are unknown of the type until runtime and can get the form of different classes.

- Event-driven programming (EDP) - is a programming paradigm in which the flow of the program is determined by events such as mouse clicks, keypresses, sensor outputs, or message passing from other programs or threads. Some of the characteristics are below.

  Service-Oriented – they are made for services and do not slow down the computer.
  Time-Driven – includes code that is triggered after some time limit.
  Event Handlers - code that runs when a specific event occurs is a key function in this paradigm.

In EDP, classes should be created from pre-defined classes while in OOP, a developer can create any class according to instructions. EDP and OOP can stay together in one system and EDP is supporting other paradigms. Fundamentally OOP and EDP are different but united they work perfectly well. OOP and PP are different ways of solving the same problem. In OOP objects do not share data but PP shared data among functions. The way OOP and PP look at real-world scenarios are fundamentally different. EDP contains different components listening to the various event and is used to make the system responsive to events. PP and EDP are also conceptually different but both have different components assigned to specific tasks.

06. Three features in an IDE

Text editor - Every IDE has a text editor inbuilt. Developer can write and manipulate code in that. Some IDEs have drag and drop components as well.

Debugger – Debugger assists the programmer in locating the errors, the specific lines, remote debugging, connecting to clients, simulate real-world scenarios, and even suggests simple fixes to remedy the bugs.

Intelligent Code completion – Known as IntelliSense, IDEs have this feature to intelligently suggests developers code components while writing the code. It is a great time saver and mostly keeps the code away from typos.

7. Conclusion

The above report included a definition of Algorithms in the first section. The second section contained a description of the process of building an application from algorithm to execution. The third section is about how to choose a programming language and the connection between an algorithm and a program. The fourth section described three programming paradigms Procedural, Event-Driven and Object-oriented. The last section is a brief description of some key features in an IDE.

8. References

‘Book of Ingenious Devices’ (2021) *Wikipedia*. Available at: https://en.wikipedia.org/w/index.php?title=Book_of_Ingenious_Devices&oldid=1032142113 (Accessed: 23 July 2021).

Eck, D. J. (2020) *Introduction to Programming Using Java, Eighth Edition*. Available at: https://math.hws.edu/javanotes/ (Accessed: 24 July 2021).

Meier, A. (2016) *The 9th-Century Islamic ‘Instrument Which Plays by Itself’*, *Hyperallergic*. Available at: http://hyperallergic.com/285064/the-9th-century-islamic-instrument-which-plays-by-itself/ (Accessed: 25 July 2021).

O’Connor, J. J. and Robertson, E. F. (1988) *Charles Babbage - Biography*, *Maths History*. Available at: https://mathshistory.st-andrews.ac.uk/Biographies/Babbage/ (Accessed: 25 July 2021).

Puiu, T. (2021) *Celebrating Ada Lovelace: the first computer programmer (XIXth century)*. Available at: https://www.zmescience.com/science/news-science/ada-lovelace-first-programmer-452542/ (Accessed: 25 July 2021).