Evolutionary Trend In The Development Of Computer Programming Languages And Concepts

Dr.Onu Fergus.U.
Department of Computer Science, Ebonyi State University,
Abakaliki, Nigeria.

Asogwa Tochukwu Chijindu
Department of Computer Engineering, Enugu State University of Science and Technology(ESUT),
Enugu, Nigeria.

Mba Calister Nnenna
Department of Computer Engineering, Caritas University,
Enugu, Nigeria

Edeh Eucharia Ujunwa
Information Communication Technology Unit, Projects Development Institute (PRODA),
Enugu, Nigeria.

***Abstract*—Programming languages are the tools that allow communication between the Computer and Computer application developers. Apart from being a static tool, programming languages evolve ie they are created and constantly change over the course of their use. Some issues led to this evolution. Such issues include speed, memory management, reusability, efficiency, maintainability, portability and compatibility issues. This paper traced the history, taxonomy, evolutionary path, characteristics, usage and the future of popular Computer programming languages in use today. Experts' inputs, experiences from Software Engineering practice and review of some reliable documents were the sources of facts for this paper.**

Keywords—Programming Languages, Computer Programming, Evolution, Taxonomy, Software Engineering, reusability, portability.

Introduction

Background of Study

The term Computer language is sometimes used interchangeably with programming language.[1] However, the usage of both terms varies among authors, including the exact scope of each. One usage describes programming languages as a subset of Computer languages.[2] In this vein, languages used in computing that have a different goal than expressing Computer programs are generically designated Computer languages. For instance, markup languages are sometimes referred to as Computer languages to emphasize that they are not meant to be used for programming.[3] Another usage regards programming languages as theoretical constructs for programming abstract machines, and Computer languages as the subset thereof that runs on physical Computers, which have finite hardware resources.[4] John C. Reynolds emphasized that formal specification languages are just as much programming languages as are the languages intended for execution. He also argued that textual and even graphical input formats that affect the behavior of a Computer are programming languages, despite the fact they are commonly not turing-complete, and remarked that ignorance of programming language concepts is the reason for many flaws in input formats. Programming language is a formal constructed language designed to communicate instructions to a machine, particularly a Computer. Programming languages can be used to create programs to control the behavior of a machine or to express algorithms.

The earliest programming languages preceded the invention of the digital Computer and were used to direct the behavior of machines such as Jacquard looms and player pianos.[5] Thousands of different programming languages have been created, mainly in the Computer field, and many more are still being created every year. Many programming languages require computation to be specified in an imperative form (i.e. as a sequence of operations to perform), while other languages utilize other forms of program specification such as the declarative form (i.e. the desired result is specified, not how to achieve it).The description of a programming language is usually split into the two components of syntax (form) and semantics (meaning). Some languages are defined by a specification document (for example, the C programming language is specified by an ISO Standard), while other languages (such as Perl) have a dominant implementation that is treated as a reference.

Aim and Objectives of the Research

The aim of this research is to really find out the evolutionary trend in the development of Computer Programming languages and concepts.
The objectives of the research are as follows:

- to examine the trends in the development of Computer Programming languages.
- to study the history of Computer Programming Languages.
- to find out the future development of Computer Programming Languages.

Literature Review

Programming Language Defined:

A programming language is a notation for writing programs, which are specifications of a computation or algorithm.[6] Some authors restrict the term "programming language" to those languages that can express all possible algorithms.[6][7] A Computer programming language is a language used to write Computer program, which involves a Computer performing some kind of computations[8] or algorithm and possibly control external devices such as printers, disk drives, robots,[9] and so on. For example, Post script programs are frequently created by another program to control a Computer printer or display. More generally, a programming language may describe computation on some, possibly abstract, machines. Programming languages differ from natural languages in that natural languages are only used for interaction between people, while programming languages also allow humans to communicate instructions to machines.

Characteristics of a Programming Language

Some characteristics of programming language include:

- Readability: A good high-level language will allow programs to be written in some ways that resemble a quite-English description of the underlying algorithms. If care is taken, the coding may be done in a way that is essentially self-documenting.
- Portability: High-level languages, being essentially machine independent, should be able to develop portable software.
- Generality: Most high-level languages allow the writing of a wide variety of programs, thus relieving the programmer of the need to become expert in many diverse languages.
- Brevity: Language should have the ability to implement the algorithm with less amount of code. Programs expressed in high-level languages are often considerably shorter than their low-level equivalents.
- Error checking: Being human, a Programmer is likely to make many mistakes in the development of a Computer program. Many high-level languages enforce a great deal of error checking both at compile-time and at run-time.
- Efficiency: It should permit the generation of efficient object code.
- Modularity: It is desirable that programs can be developed in the language as a collection of separately compiled modules, with appropriate mechanisms for ensuring self-consistency between these modules.
- Widely available: Language should be widely available and it should be possible to provide translators for all the major machines and for all the major operating systems.

Uses of Programming Languages

A programming language provides a structured mechanism for defining pieces of data, and the operations or transformations that may be carried out automatically on that data. A programmer uses the abstraction present in the language to represent the concepts involved in a computation. These concepts are represented as a collection of the simplest elements available (called primitives). Programming is the process by which programmers combine these primitives to compose new programs, or adapt existing ones to new uses or a changing environment. Programs for a Computer might be executed in a batch process without human interaction, or a user might type commands in an interactive session of an interpreter. In this case the "commands" are simply programs, whose execution is chained together. When a language can run its commands through an interpreter (such as a UNIX shell or other command-line interface), without compiling, it is called a scripting language. It is difficult to determine which programming languages are most widely used, and what usage means varies by context. One language may occupy the greater number of programmer hours, a different one have more lines of code, and a third utilize the most CPU time. Some languages are very popular for particular kinds of applications. For example, COBOL is still strong in the corporate data center, often on large mainframes.[8][9] FOTRAN in Scientific and Engineering applications; Ada in aerospace, transportation, military, real-time and embedded applications; and C in embedded applications and operating systems. Other languages are regularly used to write many different kinds of applications. Combining and averaging information from various internet sites, langpop.com claims that in 2013 the ten most popular programming languages are (in descending order by overall popularity): C, Java, PHP, JavaScript, C++, Python, Shell, Ruby, Objective-C and C#.

A. *Taxonomy of Computer Programming Language*

There is no all-embracing classification for Programming languages. A given programming language does not usually have a single ancestor language. Languages commonly arise by combining the elements of several predecessor languages with new ideas in circulation at the time. Ideas that originate in one language will diffuse throughout a family of related languages, and then leap suddenly across familial gaps to appear in an entirely different family. The task is further complicated by the fact that languages can be classified along multiple axes. For example, Java is both an object-oriented language (because it encourages object-oriented organization) and a concurrent language (because it contains built-in constructs for running multiple threads in parallel). Python is an object-oriented scripting language.

In broad strokes, programming languages divide into programming paradigms and a classification by intended domain of use, with general-purpose programming languages distinguished from domain-specific programming languages. Traditionally, programming languages have been regarded as describing computation in terms of imperative sentences, i.e. issuing commands. These are generally called imperative programming languages. A great deal of research in programming languages has been aimed at blurring the distinction between a program as a set of instructions and a program as an assertion about the desired answer, which is the main feature of declarative programming. More refined paradigms include procedural programming, object-oriented programming, functional programming, and logic programming; some languages are hybrids of paradigms or multi-paradigmatic. An assembly language is not so much a paradigm as a direct model of underlying machine architecture. By purpose, programming languages might be considered general purpose, system programming languages, scripting languages, domain-specific languages, or concurrent/distributed languages (or a combination of these). Some general purpose languages were designed largely with educational goals. A programming language may also be classified by factors unrelated to programming paradigm. For instance, most programming languages use English language keywords, while a minority do not. Other languages may be classified as being deliberately esoteric or not.

Historical Development of Computer Programming Languages

Ever since the invention of Charles Babbage's difference engine in 1822, Computers have required a means of instructing them to perform a specific task. This means is known as a programming language. In 1945, John Von Neumann at the Institute for Advanced Study developed two important concepts that directly affected the path of Computer programming languages. The first was known as "Shared-Program Technique". This technique stated that the actual computer hardware should be simple and not need to be hard-wired for each program. Instead, complex instructions should be used to control the simple hardware, allowing it to be reprogrammed much faster. The second concept was also extremely important to the development of programming languages. Von Neumann called it "Conditional Control Transfer". This idea gave rise to the notion of subroutines, or small blocks of code that could be jumped to in any order, instead of a single set of chronologically ordered steps for the Computer to take. The second part of the idea stated that Computer code should be able to branch based on logical statements such as IF (expression) THEN, and looped such as with a FOR statement. "Conditional Control Transfer" gave rise to the idea of "libraries," which are blocks of code that can be reused. In 1949, a few years after Von Neumann's work, the language Short Code appeared. It was the first Computer language for electronic devices and it required the programmer to change its statements into 0's and 1's by hand. Still, it was the first step towards the complex languages of today. In 1951, Grace Hopper wrote the first compiler, A-0. A compiler is a program that turns the language's statements into 0's and 1's for the Computer to understand. This led to faster programming, as the programmer no longer had to do the work by hand.

In 1957, the first of the major languages appeared in the form of FORTRAN. Its name stands for FORmula TRANslating system. The language was designed at IBM for scientific computing. The components were very simple, and provided the programmer with low-level access to the Computers innards. Today, this language would be considered restrictive as it only included IF, DO, and GOTO statements, but at the time, these commands were a big step forward. The basic types of data in use today got their start in FORTRAN, these included logical variables (TRUE or FALSE), and integer, real, and double-precision numbers.

In 1958, John McCarthy of MIT created the LISt Processing (or LISP) language. It was designed for Artificial Intelligence (AI) research. The Algol language was created by a committee for scientific use in 1958. Its major contribution is being the root of the tree that has led to such languages as Pascal, C, C++, and Java. It was also the first language with a formal grammar, known as Backus-Naar Form or BNF. Pascal began in1968 and its development was mainly out of necessity for a good teaching tool. C was developed in 1972 by Dennis Ritchie while working at Bell Labs in New Jersey. The transition in usage from the first major languages to the major languages of today occurred with the transition between Pascal and C. Its direct ancestors are B and BCPL, but its similarities to Pascal are quite obvious. C uses pointers extensively and was built to be fast and powerful at the expense of being hard to read. But because it fixed most of the mistakes Pascal had, it won over former-Pascal users quite rapidly.

In the late 1970's and early 1980's, a new programming method was developed. It was known as Object Oriented Programming(OOP). Objects are pieces of data that can be packaged and manipulated by the programmer. Bjarne Stroustroup liked this method and developed extensions to C known as "C with Classes." This set of extensions developed into the full-featured language C++, which was released in 1983. C++ was designed to organize the raw power of C using OOP, but maintain the speed of C and be able to run on many different types of Computers. In the early 1990's, interactive TV was the technology of the future. Sun Microsystems decided that interactive TV needed a special, portable (can run on many types of machines), language. This language eventually became Java. In 1994, the Java project team changed their focus to the web, which was becoming "the cool thing" after interactive TV failed. The next year, Netscape licensed Java for use in their internet browser, Navigator. At this point, Java became the language of the future and several companies announced applications which would be written in Java, none of which came into use. Java had serious optimization problems but may wind up as the instructional language of tomorrow as it is truly object-oriented and implements advanced techniques such as true portability of code and garbage collection.

Discussions

So far, technological upgrade has been the major reason for the advances in the software engineering industry. Such upgrades are observed in the areas of hardware components, software components, users' needs, networks as well as distributed processing. These areas have direct implications on the issues that really contribute to software development trend and evolution. Such issues include but not limited to the following: Microprocessor's speed/access time, memory management issues, efficiency issues, compatibility issues, reusability and portability issues. Earlier software programs occupied a lot of memory spaces, so running such programs was not fast and as such was not efficient by any means. Code reuse, which is a common feature of the object oriented languages, does not apply in the procedural languages.

Though FORTAN was good at handling numbers, it was not so good at handling input and output, which mattered most to business computing. Business computing took off in 1959, and because of this, COBOL (Common Business Oriented Languages) was developed. It was designed from the ground up as the language for businessmen. Its only data types were numbers and strings of texts. A LISP list is denoted by a sequence of items enclosed by parentheses. LISP programs themselves are written as a set of lists, so that LISP has the unique ability to modify itself, and hence grow on its own. The LISP syntax was known as "Cambridge Polish," as it was very different from standard Boolean logic. Though Algol implemented some novel concepts, such as recursive calling of functions, the next version of the language, Algol 68, became bloated and difficult to use. This led to the adoption of smaller and more compact languages, such as Pascal. Pascal was designed in a very orderly approach; it combined many of the best features of the languages in use at the time, COBOL, FORTRAN, and ALGOL. While doing so, many of the irregularities and oddball statements of these languages were cleaned up, which helped it gain users. The combination of features, input/output and solid mathematical features, made it a highly successful language. Pascal also improved the "pointer" data type, a very powerful feature of any language that implements it. Pascal also helped the development of dynamic variables, which could be created while a program was being run, through the NEW and DISPOSE commands. However, Pascal did not implement dynamic arrays, or groups of variables, which proved to be needed and led to its downfall (Bergin, 101-102). Wirth later created a successor to Pascal, Modula-2, but by the time it appeared, C was gaining popularity and users at a rapid pace.

In procedural languages, programs are made up of modules, which are parts of a program that can be coded and tested separately, and then assembled to form a complete program. These modules are procedures, where a procedure is a sequence of statements. These procedures are functions, which map arguments to return statements. An alternative to procedural programming is Object Oriented Programming. Object Oriented Programming is meant to address the difficulties with procedural programming. In object oriented programming, the modules in a program are classes, rather than procedures. The object-oriented approach lets you create classes and objects that model real world objects. One common trend in the development of programming languages has been to add more ability to solve problems using a higher level of abstraction. The earliest programming languages were tied very closely to the underlying hardware of the Computer. As new programming languages have developed, features have been added that let programmers express ideas that are more remote from simple translation into the underlying hardware instructions. A language's designers and users must construct a number of artifacts that govern and enable the practice of programming. The most important of these artifacts are the language specification and implementation.

Specification of Programming Languages

The specification of a programming language is an artifact that the language users and the implementers can use to agree upon whether a piece of source code is a valid program in that language, and if so, what is its behavior. A programming language specification can take several forms including the following:

- An explicit definition of the syntax, static semantics, and execution semantics of the language. While syntax is commonly specified using a formal grammar, semantic definitions may be written in natural language (eg as in C language), or a formal semantics (eg as in standard ML and scheme specifications).
- A description of the behavior of a translator for the language (eg the C++ and FORTRAN specifications). The syntax and semantics of

the language have to be inferred from this description, which may be written in natural or a formal language.

- A reference or model implementation, sometimes written in the language being specified (eg Prolog or ANSI REXX). The syntax and semantics of the language are explicit in the behavior of the reference implementations.

Implementation of Programming Languages

An implementation of a programming language provides a way to write programs in that language and execute them on one or more configurations of hardware and software. There are broadly two approaches to programming language implementation viz: compilation and interpretation. It is generally possible to implement a language using either technique. The output of a compiler may be executed by hardware or a program called an interpreter. In some implementations that make use of the interpreter approach, there is no distinct boundary between compiling and interpreting. For instance, some implementations of BASIC compile and then execute the source a line at a time. Programs that are executed directly on the hardware usually run faster than those interpreted in software. One technique for improving the performance of interpreted programs is just-in time compilation. Here, the virtual machine, just before execution, translates the blocks of byte code which are to be used to machine code, for direct execution on the hardware.

Conclusion

As the developers' needs have evolved, so have the abilities of programming languages evolved. If a programming language is not expressive enough, then it must evolve to allow its users the ability to articulate their abstractions or it will become extinct. At one time, many believed that a single multi-purpose programming language would allow developers to standardize. However, the wide variety of problems that need solving and the diverse philosophies of developers have appropriately led to different languages for different purposes. Programmers do not need to use complex languages; there is enough complexity in the world for them already. During all these evolutions, the basic role of a programming language will not change – allowing the developer to easily express abstract ideas in a language that a machine can execute. Future advances in programming languages will only be made possible by the evolutionary advances being made today. In the near future, the general evolutionary trends of increasing machine independence, increasing programming language interoperability, and increasing modularity will continue.

References

1 Robert A. Edmunds, The Prentice-Hall standard glossary of computer terminology, Prentice-Hall, 1985, p. 91
2 Pascal Lando, Anne Lapujade, Gilles Kassel, and Frédéric Fürst, TOWARDS a General Ontology of Computer Programs, ICSOFT 2007, pp. 163-170
3 S.K. Bajpai, INTRODUCTION To Computers And C Programming, New Age International, 2007, ISBN 81-224-1379-X p. 346
4 John C. Reynolds, SOME thoughts on teaching programming and programming languages,SIGPLAN Notices, Volume 43, Issue 11, November 2008, p.109
5 Ettinger, James (2004), Jacquard's Web, Oxford University Press
6 Aaby, Anthony (2004). INTRODUCTION to programming Languages.
7 Bruce J. (1987). PRINCIPLES of Programming Languages. Oxford University Press. p. 1.ISBN 0-19-511306-3
8 David A. Schmidt, THE structure of typed programming languages, MIT Press, 1994, ISBN 0-262-19349-3 , p. 32
9 Pierce, Benjamin (2002). TYPES and Programming Languages. MIT Press. p. 339.ISBN 0-262-16209-1
10 Ben Ari, Mordechai (1996). UNDERSTANDING Programming Languages. John Wiley and Sons.
11 David A. Schmidt, THE structure of typed programming languages, MIT Press, 1994, ISBN 0-262-19349-3 , p. 32
12 Steven R. Fischer, A HISTORY of language, Reaktion Books, 2003, ISBN 1-86189-080-X , p. 205
13 Éric Lévénez (2011). "COMPUTER Language History".
14 Jing Huang (2012). ARTIFICIAL Languages vs Natural Languages
15 Luca Cardelli and Peter Wegner. "ON UNDERSTANDING Types, Data Abstraction and Polymorphism". Manuscript (1985). Retrieved 19 July 2015.
16 Dijkstra, Edsger W."ON THE FOOLISHNESS of natural language programming" EWD667.
17 Perlis, Alan (September 1982). "Epigram on programming" SIGPLAN Notices Vol. 17, No. 9. pp. 7–13.