VERSITA
10.2478/cris-2013-0011

# PROGRAMMING LANGUAGE PARADIGMS & THE MAIN PRINCIPLES OF OBJECT-ORIENTED PROGRAMMING

NIKOLETTA MINAROVA

## INTRODUCTION

Since the first design concept of computers came into the world, many different methods of communication with these machines have been developed (Wu, 2010, p. 2).

### FIRST GENERATION LANGUAGE

A computer's architecture is based on the binary number system, and this is why the CPU, or Central Processing Unit, can only understand one type of language, known as machine language (Davis, 2010, p. 17). Davis (2010, p. 17) says that this "first generation", low-level language is represented "as a sequence of numbers written either in binary or hexadecimal" and goes on to imply that the system of 1's and 0's is virtually incomprehensible to humans.

### SECOND GENERATION LANGUAGE

According to Brookshear (2012, p. 240), writing programmes in machine code is slow and leads to many errors which requires debugging, and as a result, a slightly higher-level language was developed in the 1940s. A notational system where instructions can be displayed in mnemonic form is featured in this generation of programming, which collectively is known as assembly language (Gabrrielli and Martini, 2010, p. 5). Wu (2010, p. 11) mentioned that the CPU is not able to recognise this form of communication and needs a special assembler to translate it into the machine language equivalent.

### THIRD GENERATION LANGUAGES

A series of third generation languages was devised in the 1950s (Davis, 2010, p. 18). These high-level programming languages were developed with more abstraction, basing their grammar on human language with the use of command words (Davis, 2010, p. 18). Programmes written in high level languages are not recognised by the CPU, and a compiler must be used to translate them into the assembly language equivalents (Wu, 2010).

### PARADIGMS

Many distinct concepts of programming, known as paradigms, have been devised since the development of third generation languages (Brookshear, 2012, p. 244). Some of these, including the procedural paradigm, will be discussed in this paper and further on, will be compared against the object-oriented paradigm.

## 1. THE PROCEDURAL PARADIGM

*One of the most common and understood paradigms in use today is procedural programming, which is also known as imperative programming (Deirbach, 2013, p. 17). Brookshear (2012, p. 244) even goes so far as to say that the procedural paradigm "represents the traditional approach to the programming process".*

### 1.1 DESCRIPTION

Procedural programming can somewhat be described as a flow chart where a linear sequence of statements is clearly defined for the programme to complete (Yevick, 2012, p. 35). When this sequence is followed, the programme manipulates data to create the desired execution, for example the change of a value of a location in its memory (Brookshear, 2012, p. 244).

#### 1.1.1 CHARACTERISTICS AND PRINCIPLES

There are small sections of code in the procedural paradigm known as functions or procedures, which are able to act as small programmes within their own scope (Weisfeld, 2009, p. 6). An example of this could be a programme which has a main input load where the programme goes around in circles waiting for the input of a key on a keyboard. These procedures are able to be called upon from other parts of the programme in the same way that functions can be called on within functions (Brookshear, 2012, p. 244). The main characteristic of the procedural paradigm is that it runs on sequential logic, where the logical flow of a programme can be followed from start to end even through various calls to procedures (Kedar, 2011, p. 139).

### 1.2 HISTORY

The best known early language of this type was Fortran (FORmula TRANSlating system) which was developed in the 1950s by an IBM team led by John Backus (Beebe, 1992, p. 1). It was a compiled language that was initially used for scientific calculations and engineering purposes (Brookshear, 2012, p. 241).

### 1.3 MODERN EXAMPLES OF PROCEDURAL PROGRAMMING

A good example of the procedural paradigm is seen in C, a programming language that was devised by Dennis Ritchie of Bell Laboratories in the 1970s (Ben-Ari, 2006, p. 5).

Ben-Ari (2006, p. 6) mentions that "C was designed to be close to assembly language so it is extremely flexible." Many object oriented programming languages are characterised on the style and principles of the procedural paradigm including C++, which is based on C (Brookshear, 2012, p. 248). That being said, most programming languages allow for different paradigms to be used in conjunction when developing a programme (Weisfeld, 2009, p. 5).

### 1.4 STRUCTURED PROGRAMMING

The structured programming paradigm is a subset of procedural programming which was developed to manage larger programmes and to improve accuracy, quality and development life cycle (Gabbrielli and Martini, 2010, p. 150). The programme structure is usually broken down into small comprehensible sized portions of code that are easy to manipulate and change later on (Gabbrielli and Martini, 2010, p. 150).

## 2. THE OBJECT-ORIENTED PARADIGM

The prominent object oriented paradigm is sometimes said to be an extension of procedural programming (Weisfeld, 2009, p. 6). It utilises a high level of abstraction with sets of objects which contain information describing how those objects should respond and communicate with different stimuli (Brookshear, 2012, p. 276).

### 2.1 DESCRIPTION

The main intention of object-oriented programming is to be able to develop programmes that are flexible and maintainable (Gabbrielli and Martini, 2010, p. 277). Programmes in this language are described as having high modularity, and therefore they can be simply developed and easily comprehended (Stroustrup, 2013, p. 51).

#### 2.1.1 CHARACTERISTICS AND PRINCIPLES

To have a clear understanding of the object-oriented paradigm, the idea of an "object" must be understood. Weisfeld (2009, p. 6) defines an object as being an "entity that contains both data and behavior". These attributes and behaviors are usually separated in other paradigms (Lipmann et al., 2012). Object-oriented programming combines various objects which interact with one another to form a complete programme (Wu, 2010). Objects have many abilities including receiving and passing messages, processing entries and detecting information (Stroustrup, 2013, p. 12). Every object must be made up of a "class" which is formu-

lated by a set of instructions that specifies its data and functions (Purdum, 2013, p. 21). Weisfeld (2009, p. 31) suggests that a class can be considered a "blueprint for an object" and that once it is defined, many objects belonging to the class are able to be created. Wu (2010, p. 23) also identifies another mechanism in object-oriented programming called inheritance, which he says is used to "design two or more entities that are different but share many common features". It utilises class hierarchy, including superclasses and subclasses which allow for reusability and the extension of existing classes (Wu, 2010, p. 23). The polymorphism feature allows for same messages to be sent to objects from different classes (Purdum, 2013, p. 492). Another important feature of object-oriented programming is called encapsulation, which is a process of binding data in an object so that it cannot be accessed by external code which has been defined outside the class (Brookshear, 2012, p. 282).

### 2.2 HISTORY

Two of the earliest languages based on the formal programming theory of objects were developed by Ole-John Dahl and Kristen Nygaard from Norway, and were called SIMULA 1 and Simula 67 (Purdum, 2013, p. 4). Despite having the advantages that this paradigm offers, object-oriented programming did not become popular until Bjarne Stroustrups's development of C++ which addressed many of the dilemmas found in procedural programming (Stroustrup, 2013, p. 11).

### 2.3 MODERN EXAMPLES OF OBJECT-ORIENTED PROGRAMMING

The programming language Java is one of the newest object-oriented languages and was designed by James Gosling in cooperation with Sun Microsystems in the early 1990s (Wu, 2010, p. 12). Java was based on C and C++ and was initially developed to connect household machines (Wu, 2010, p. 12). Java is commonly used to write Web applications and is therefore is often described as a Web programming language (Wu, 2010, p. 12).

### 2.4 VON NEUMANN ARCHITECTURE

Both of the above mentioned paradigms are based on the programming style of the von Neumann architecture (Ben-Ari, 2006, p. 274). This architecture features updatable storage with a sequence of memory cells whose contents are modified when an instruction is executed (Ben-Ari, 2006, p. 274).

Functional programming is not natively based on the von Neumann architecture, but on a model of computation based on the mathematical concept of functions (Ben-Ari, 2006, p. 274).

## 3. THE FUNCTIONAL PARADIGM

### 3.1 DESCRIPTION

Turner (2004, p. 251) describes the functional paradigm as programming which is "closely related to mathematics". A programme of this type consists of an expression and is characterised by values, functions and functional forms (Turner, 2004, p. 255).

#### 3.1.1 CHARACTERISTICS AND PRINCIPLES

A functional paradigm uses special functions that can accept inputs and produces outputs (Brookshear, 2012, p. 245). Programmes are developed by putting predefined functions together until a desired input/output relationship is created (Brookshear, 2012, p. 245). Common features found in functional languages include first class functions, pure functions, recursion, and pattern matching (Ben- Ari, 2006, p. 274). Smith (2010, p. xv) mentions that the functional paradigm focuses on what a programme should do and not how the programme is meant to work overall.

### 3.2 HISTORY

One of the earliest programming languages which used the functional paradigm in its design is Lisp (Ben-Ari, 2006, p. 274). It was designed in 1956 by John McCarthy and used computation based on the lambda calculus model (Ben-Ari, 2006, p. 274). Although Lisp can be described as a functional language, it uses many features from other paradigms (Ben-Ari, 2006, p. 274).

### 3.3 MULTI-PARADIGM PROGRAMMING LANGUAGES

As Brookshear (2012, p. 257) said, some programming languages combine various features of different paradigms. F# is an open source programming language that was originally developed by Microsoft Research, Cambridge (Smith, 2010, p. xiii). It supports several different programming paradigms including functional, object-oriented, and procedural, which makes it a multi-paradigm programming language (Smith, 2010, p. xv).

## 4. COMPARISON OF PARADIGMS

### 4.1 THE OBJECT-ORIENTED PARADIGM

#### 4.1.1 ADVANTAGES

The modularity of the object-oriented paradigm is a major advantage of this type of programming (Stroustrup, 2013, p. 51). It has the ability to easily define abstract data types whose implementation details are hidden (Ben-Ari, 2006, p. 227). Another important advantage is the ability to use inheritance which allows programmers to maintain and reuse codes and extensions without having to make any big changes any existing source codes (Wu, 2010, p. 23). Sets of pre-defined objects and code can be reused which allows for faster development (Wu, 2010, p. 199). Encapsulation is another benefit for programmers when using the object-oriented paradigm (Brookshear, 2012, p. 282).

#### 4.1.2 DISADVANTAGES

A major drawback of this paradigm is that it can cause decreased system performance as it has large memory requirements (Chen, 2004, p. 70). This type of programming typically results in larger programme sizes as it includes more lines of code and planning, which, along with the demand for more system resources, can slow the programme down (Chen, 2004, p. 70). Another disadvantage could be unanticipated complex interaction between objects which may sometimes create a problem in the code (Weisfeld, 2009, p. 70). For some programmers, the idea behind creating programmes based on the interaction of objects is unnatural and it may be difficult to get used to characteristics such as inheritance and polymorphism (Weisfeld, 2009, p. 2).

### 4.2 THE PROCEDURAL PARADIGM

#### 4.2.1 ADVANTAGES

Brookshear (2010, p. 244) refers to the procedural paradigm as "traditional programming" which implies the advantage that it is very well-known and researched. It allows for each procedure to be programmed separately, which means if there is an error in the code, the problem area can be quickly isolated and repaired (Chen, 2004, p. 71). It can be advantageous to use procedural programming when developing smaller programmes because it is written in sequential steps (Brookshear, 2012, p. 244). Since it is used for specific uses, the programmes tend to be very efficient (Chen, 2004, p. 71).

#### 4.2.2 DISADVANTAGES

The big disadvantage to procedural programming is that if an error is repaired, every line that corresponds to the original change in code also needs to be edited (Ben-Ari, 2006, p. 282). This can become particularly troublesome when dealing with the maintenance of programmes which have large amounts of code (Gabbrielli and Martini, 2010, p. 358). Another problem with procedural languages is that it is difficult to relate to real world objects and forces programmers to view problems as a set of steps (Chen, 2004, p. 71).

#### 4.2.3 COMPARISON WITH THE OBJECT-ORIENTED PARADIGM

Procedural programming focuses on the purpose of the programme and commonly has global variables and functions whereas object-oriented programming is based on objects and usually does not feature global variables (Chen, 2004, p. 72). The procedural paradigm is a representation of computational processes in a sequential order which is difficult to compare with the real world, whilst the object-oriented paradigm views the world as a system of interacting objects (Brookshear, 2010, p. 249). Whereas object-oriented programming allows for reuse of code and uses inheritance, many portions of procedural code are so interdependent that they are not able to be used in other applications (Brookshear, 2010, p. 247). Programmes written in an object-oriented language typically are bigger and slower than in a procedural language (Chen, 2004, p. 70). However, bigger and more complicated programmes can be maintained and altered much easier when using object-oriented programming (Brookshear, 2010, p. 247).

### 4.3 THE FUNCTIONAL PARADIGM

#### 4.3.1 ADVANTAGES

One of the biggest advantages of the functional paradigm is that programmers are able to compose functions and can build larger abstractions from smaller ones (Brookshear, 2010, p. 246). Hughs (1990, p. 22) refers to the functional paradigm as having better modularity and adds that "functional programming languages provide two new kinds of glue - higher-order functions and lazy evaluation".

Another advantage to functional programming is that it is relatively easy to test and debug code (Hughs, 1990, p. 3). These functions are very efficient as they are usually devised to have only one specific task and can be easily read and maintained (Ben-Ari, 2006, p. 275).

#### 4.3.2 DISADVANTAGES

A major disadvantage to functional programming is that it is very hard to learn and because of this, programmes written in the language by beginners could be unnecessarily slow (Ben-Ari, 2006, p. 274). The syntax and style of the functional paradigm is completely different from others and computations may be slower (Hughs, 1990, p. 25). Another drawback is that modern computers are based on von Neumann architecture and this kind of programming does not match the hardware as well as the procedural or object-oriented paradigms (Ben-Ari, 2006, p. 274).

#### 4.3.3 COMPARISON WITH THE OBJECT-ORIENTED PARADIGM

As was previously mentioned, the object-oriented paradigm is based on the programming style of the von Neumann architecture, which could give it an advantage over functional programming (Ben-Ari, 2006, p. 274). In object-oriented programming, when a programme involves many variables, they can be easily associated with a particular class which could make it easier to use in comparison to functional programming (Purdum, 2013, p. 21).

## CONCLUSION

Although there are many advantages and disadvantages of using various paradigms, there is no single programming style which fit all problems well. Many programmers favour certain languages because of the benefits they provide for the particular work they need it for, and dislike others due to certain difficulties in using them. While it is true that the object oriented paradigm is becoming very popular in the world of programming because of the enormous benefits it provides, there are still programmers who prefer to use other languages. ■

## REFERENCES


Beebe, N. F. (1992) *A Summary of Fortran.* Center for Scientific Computing. University of Utah. [Online]. Available at: http://www.math.utah.edu/~beebe/software/fortran-documentation/ ftnsum.pdf (Accessed: 17 October 2013).

Ben-Ari, M. (2006) *Understanding Programming Languages.* Chichester: John Wiley & Sons.

Brookshear, J. G. (2012) *Computer Science: An Overview.* 11th edn. Boston: Addison-Wesley.

Chen, K. C. (2004) ' Comparison of Object-Oriented and Procedure-Based Computer Languages: Case Study of C++ Programming', *Journal of Computer Information Systems,* 5(1), pp. 70-76, IACIS. [Online]. Available at: http://iacis.org/iis/2004/Chen.pdf (Accessed: 15 October 2013).

Davis, S. R. (2010) *Beginning Programming with C++ For Dummies.* IT eBbooks [Online]. Available at: http://www.it-ebooks.info (Accessed: 2 October 2013).

Dierbach, C. (2013) *Introduction to Computer Science Using Python.* IT eBbooks [Online]. Available at: http://www.it-ebooks.info (Accessed: 9 October 2013).

Gabbrielli, M. and Martini, S. (2010) *Programming Languages: Principles and Paradigms.* Bologna: Springer.

Hughs, J. (1990) *Why Functional Programming Matters* [Online]. Available at: http:// www.cse.chalmers.se/~rjmh/Papers/whyfp.pdf (Accessed: 15 October 2013).

Kedar, S. (2011) *Principles of Programming Languages.* 2nd edn. Pune: Technical Publications.

Lipmann, S. B., Lajoie, J. and Moo, B. E. (2012) *C++ Primer*. IT eBbooks [Online]. Available at: http://www.it-ebooks.info (Accessed: 3 October 2013).

Purdum, J. (2013) *Beginning Object-Oriented Programming with C#*. IT eBbooks [Online]. Available at: http://www.it-ebooks.info (Accessed: 4 October 2013).

Smith, C. (2010) *Programming F#.* IT eBbooks [Online]. Available at: http://www.it- ebooks.info (Accessed: 18 October 2013).

Stroustrup, B. (2013) *The C++ Programming Language.* 4th edn. New Jersey: Addison-Wesley.

Turner, D. A. (2004) ' Total Functional Programming', *Journal of Universal Computer Science*, 10(7), pp. 751-768, J.UCS [Online]. Available at: http://www.jucs.org/jucs_10_7/ total_functional_programming (Accessed: 16 October 2013).

Weisfeld, M. (2009) *The Object-Oriented Thought Process.* IT eBbooks [Online] Available at: http://www.it-ebooks.info (Accessed: 4 October 2013).

Wu, C. T. (2010) *An Introduction to Object-Oriented Programming with Java.* IT eBooks [Online]. Available at: http://www.it-ebooks.info (Accessed: 4 October 2013).

Yevick, D. (2012) *A Short Course in Computational Science and Engineering.* Cambridge University Press eBooks [Online]. Available at: http://www.ebooks.cambridge.org/jsf? bid=CBO9781139022262 (Accessed: 15 October 2013).