An Analysis of Some Characteristics of Different Programming Paradigms

Zoran Putnik

It has been defined that: "A programming paradigm is a collection of conceptual patterns that together mold the design process and ultimately determine a program's structure [2]". Because of the fact that programs are usually executed on a Von Neumann style based computers, not best suited for programming styles different than procedural, it was not possible to correctly compare those paradigms for a long time.

With development of computers, systems allowing programming in different programming styles, or even combining different programming styles in a single program, became available. One of them is programming package *Leda*. Availability of such a system, leads to a set of interesting questions. Should we reconsider our opinions about "most efficient" programming style. Which one is the best? The "fast" one, the "readable" one, or the one which can be proven correct via mathematical formalisms.

At the beginning of development of programming paradigms, it was possible to recognize languages strictly connected to a certain paradigm. Example of *imperative* paradigm was *FORTRAN*, of *functional* paradigm was *LISP*, of *logic* paradigm was *PROLOG*. Later, it became more complicated. Natural need for combining the best features from different programming paradigms, induced programming languages that borrow from many paradigms and support more than one.

Along the way, some other programming paradigms emerged, more or less successful. Object-oriented, parallel (asynchronous or synchronous), transformational, form-based, dataflow, constraint, demonstrational and so on [2]. All of them can still be separated into two main different categories: *operational* or *demonstrational* approach, describing step-by-step how to construct a solution, and *definitional* approach, stating properties about solution, without describing how to compute it. In our analysis, imperative - as operational, and logical - as definitional paradigms are included. Functional paradigm, having characteristic of both categories is included as a fine transition from one to the other.

While discussion on readability, or correctness may be a matter of opinion, discussion on speed of programs may be checked rather agreeably. In order to gain correct and valuable results, several different kind of problems have to be included, best suited for both operational and definitional approach, i.e. for each of mentioned programming styles. This includes, for example: "Declarative" problem - mathematical problem that can be stated as a set of facts and rules, i.e. as a program written in a logic programming style, "Iterative" problem - mathematical problem, naturally suited for imperative style and procedural programming, that can still be programmed in both functional and logical manner, and "Recursive" problem - mathematical problem in which a new result is gained from a previous results, best suited for a functional programming style. All these problems are programmed in each paradigm and execution times are measured. My conclusion is: logical and procedural paradigms had almost the same execution times. Functional paradigm had slower execution time, but still very competitive one.

This paper is aimed to help in better understanding of some of the existing paradigms, oldest and most spreaded, and in changing our opinions about some "facts" about programming styles that may not be true. I presume that some kind of procedural programming (object-oriented, for example [1]) will be mostly used in the future, for which there are lots of reasons, beginning with the computer's architecture and programmers long-time habits. My main interest was to show that given a proper programming package - supporting different programming styles, a proper knowledge about the problem we have to solve - allowing combinations of programming styles and proper computer and software architecture - allowing execution of different programming paradigm, we would be able to achieve better, more efficient, more readable and mathematically correct programs.

References

[1] Rine, C.D., Bhargava, B., Object-Oriented Computing, Computer, October, 1992., pp. 6-10.

[2] Ambler, L.A., Burnett, M.M., Zimmerman, A.B., Operational Versus Definitional: A Perspective on Programming Paradigms, Computer, September, 1992., pp. 28-43.