Annales Mathematicae et Informaticae
***35** (2008) pp. 163–172*
http://www.ektf.hu/ami

ICT teaching methods – Programming languages

Zsuzsanna Papp-Varga, Péter Szlávi, László Zsakó

Department of Media and Educational Informatics
Eötvös Loránd University, Budapest, Hungary

Submitted 7 March 2008; Accepted 9 December 2008

Abstract

Today the important ICT topics are taught with the help of various methods. Some of them are unsuitable for successful teaching-learning whereas others may bring about success in certain age groups and class types.

Programming languages were first taught shortly after the appearance of high-level programming languages. First it was done rather as an “art”, but later more and more consciously and systematically. However, it should be stated that the methods used in teaching programming languages, as “languages”, are far from being near to those of natural languages with respect to their elaborateness, quality and, unfortunately, efficiency.

1. Introduction

The most important teaching methods of the various fields of information and communication technology (ICT) have already been developed [15]. As ICT teaching cannot boast with a long history, in most cases they have not been clearly formulated, and their formation has not been so conscious but rather instinctive, which results in the fact that most teachers do not use one single method but a sort of blend of methods, where one of them is represented dominantly.

This methodological “uncertainty” also ensues that there are teachers who are capable of teaching successfully even when they use a method labelled below as being negative. The negative label can be principally explained by the fact that these methods do not “automatically” ensure good teaching; what is more, it is fairly easy to teach very badly when one relies on them.

Hereinafter the most widespread programming language teaching methods are listed and reviewed:

Statement-oriented (the language is seen as a set of statements, and the individual elements of the set are taught in a certain order).

Using as a tool (when teaching programming and database management, it considers the aspects of database-teaching to be of primary concern, and thus introduces language tools in the necessary extent).

Software technology-oriented (a programming language teaching method adapted to a software developing methodology and technology, where the methodology motivates the choice of a language or even languages).

Task type-oriented (the method is identical with the one discussed at the programming teaching method; it introduces new programming language knowledge in a way that the problems to be solved necessitate).

Language-oriented (the method sees a language as a structural unit, bringing the logic of the language to front, and introducing the concrete elements of a language in the necessary extent and order).

Action-oriented (the statements of a language are taught in a way that it traces them back to an implementation in another language – formerly to assembly statements, now rather to other high-level languages).

Sample task-based (the method presents a language through an analysis of sample tasks).

2. Statement-oriented

The statement-oriented method defines a programming language as a set of statements [1]. It conceives teaching a language as teaching the elements of a set. (And to top it all, in alphabetical order, in the worst case.[1]) The idea of the set also refers to the fact that each element of the set, i.e. each statement of a language must be taught (which leads to a common ICT teaching delusion[2]). Nevertheless, it is easy to foresee the depth of acquiring a language: it is only a mere set of lexical elements lacking any connections among statements or with the modus operandi.

Neither does this notion promote deciding which elements are important and which are not. Moreover, there is no guarantee that one will ever make any use of the elements learned.

If one pictures a language this way, one can claim that a language is an unstructured unity of elements and thus there is no need for any further knowledge to construct programs from the elements: it will develop by “itself”.

[1]This idea evolved based on books on programming languages. These books have not yet reached the advanced state of those on natural languages. In most cases the same book is meant to be the manual of a given language (which is practically in alphabetical order), the language coursebook, the dictionary etc.

[2]The notion of popular delusion see in [16].

3. Applying as a tool

There are many programming teaching methods where program writing is more or less an automated activity, and can be done with the help of coding rules and coding conventions. In this case the programming language appears as a result of the coding process. One always needs only that amount that one needs for coding one's algorithms [17, 13].

Let us see some examples for the above (Pascal encoding rules):

Algorithmic statement (with Hungarian keywords)	**Pascal code equivalents**
Be: variables [conditions]	**Repeat** write('question?'); readln(variables) **until** conditions;
Ha conditions **akkor** statements **különben** statements **Elágazás vége**	**If** condition **then begin** statements **end** **else begin** statements **end;**
Ciklus amíg condition statement **Ciklus vége**	**While** condition **do begin** statements **end;**

When using this method, it is guaranteed that the acquired language elements will later be used again. Since the structures, algorithmic elements and data types recur regularly in programming craft, one can also state that the acquired elements have to be often used.

4. Software technology-oriented

Relying on the above principle, Tibor Temesvári has constructed object-oriented programming (OOP) and its implementation in the Pascal and C++ programming languages. First, he discusses object-oriented programming in general (1. Characteristics of OOP, 1.1. Classes and Objects, 1.2. Encapsulation, 1.3. Inheritance, 1.4. Implementation of Inheritance, 1.5. Using Inheritance, 1.6. Multiple Inheritance, 1.7. Type Compatibility, 1.8. Polimorphism, 1.9. Dynamic Binding, 1.10. Virtual Method, 1.11. Execution of Methods 1.12. Object-oriented Programming Languages), in which he does not touch upon concrete programming language knowledge, but only deals with the object-oriented technology.

In Chapter 2 the above are followed by teaching the implementation possibilities i.e. programming language skills (2. OOP in Pascal, 2.1. Planning, 2.2. Defining a Class 2.3. Interface Part, 2.4. Implementation Part, 2.5. Self, 2.6. The Declaration of Objects, 2.7. Using Objects, 2.8. Inheritance, 2.9. Procedure Calls Defined in the Ancestor, 2.10. Redefined Methods, 2.11. Virtual Method Table – VMT, 2.12. Constructors (procedures), 2.13. Dynamic Methods, 2.14. Dynamic Method Table (DMT), 2.15. Type Compatibility, 2.16. Dynamic Objects, 2.17. Cleaning up Dynamic Objects, Destructors). [18]

Similarly, some software technology (the OOP, database management, COM- and web-programming) denotes the guideline of Delphi language processing in a book by Marco Cantù: [8]. There is also a good example for this in the topic of web design in a book by Kris Jamsa et al. [4]. New paradigms including aspect-oriented and generic programming may also affect teaching programming languages. [10]

5. Task type-oriented

In this case the elements of a programming language are introduced because they are needed in the process of problem solving. The various elements do not turn up because some educational objective requires them, but because the next task cannot be solved without them. [12, 19]

The task below comes from a class introducing PROLOG that we developed (relying on Turbo PROLOG system):

Step 1: facts

```
one'sfather(father,child).
one'smother(mother,child).
```

Step 2: clauses

```
one'sparent(X,Y) if one'smother(X,Y).
one'sparent(X,Y) if one'sfather(X,Y).
```

Step 3: **or** operation in clauses

```
one'sparent(X,Y) if one'smother(X,Y) or one'sfather(X,Y).
```

Step 4: **and** operation in clauses

```
one'sgrandparent(X,Y) if one'sparent(X,Z) and one'sparent(Z,Y).
```

Step 5: recursion in clauses

```
one'sancestor(X,Y) if one'sparent(X,Y)
                   or one'sparent(X,Z) and one'sancestor(Z,Y).
```

Step 6: "`any`" value in the place of parameters

```
parent(X) if one'sparent(X,_).
```

Step 7: "`not`" operation i.e. negation in clauses

```
notparent(X) if one'sparent(_,X) and not (one'sparent(X,_)).
```

Step 8: cut operation in clauses

```
oneparent(X) if parent(X) and !.
```

Step 9: display, and equally false formula in clauses

```
allparent if parent(X) and write(X) or fail or nl.
```

Step 10: equivalency check in clauses

```
twochilded(X) if one'sparent(X,Y) and one'sparent(X,Z) and
                 not (Y=Z).
```

Step 11: new programming skill without new language element

```
onechilded(X) if one'sparent(X,_) and not (twochilded(X)).
```

Similar examples can be found in the syllabus on teaching Logo programming language developed at Eötvös Loránd University. Its subjects and the new language elements to be learned in brackets are as follows:

• Drawing elementary shapes	(forward, back, right, left, repeat)
• Constructing from shapes	(learn, penup, pendown)
• Principles of making complex figures	
• Circles, arcs	(setpencolor, setpenwidth!)
• Recursion, trees	(if)
• Line patterns, shape patterns	(fill, setfillcolor, setfillpattern)
• Logo and the frame of reference	(setx, sety, setheading)
• Fractals	

An important development in teaching programming languages is that the statement-oriented method is often merged with this one [20], since the abstract, "crystal clear" know-how of the previous notion, which is free from programming problems, are completed with the real-life experience of statements. It is the way that makes the level of language teaching rise significantly!

6. Language-oriented

The language-oriented variant regards the language as a *structured unit.* It examines the calculation model belonging to the language [2] (in primary and secondary education only von Neumann-principled, automaton-principled, functional, and logical languages can be present). Then it reviews the main framework of the build-up of programs e.g. in the Pascal language:

```
Program name;
  declarations
begin
  statements
end.
Declarations: label definitions
              constant definitions
              type definitions
              variable declarations
              procedure and function definitions
```

Becoming familiar with the basic concepts used in a programming language and their possible implementations in that language (e.g. compilation unit, program unit, block structure, memory management, declaration evaluation, concepts regarding variables, concepts regarding type, parameter pass etc.) also tightly belong to the build-up of a program [18].

The next step might be that certain elements of the language are examined and it is given how the programming structures are implemented in that given language. For example as for Pascal, one might claim the following about conditional loops (before setting the concrete syntactic rules):

The Pascal language can have pre-test or post-test conditional loops. For pre-test loops the condition is first evaluated – if the condition is true, the code within the block is then executed. This repeats until the condition becomes false. On the other hand, for post-test loops the exit condition must be set. The core of a pre-test loop can be one single statement. If more statements are necessary, they must be surrounded by statement brackets. Contrarily, the core of a post-test loop can contain any number of statements.

Finally, only after the above can one give the syntax and semantics of the statements. As opposed to higher education, in primary and secondary education it is usually not a formal method that is used but a demonstration via examples. To define syntax, only the format of the statement is given (pl. **while** `condition` **do** `statement`). For semantics, however, smaller programs are used, through which the operation of the given statement can be understood (with the help of the method described in the next chapter).

It should be noted that in higher education this method is becoming more and more widespread in demonstrating the possible elements of a programming language, bringing examples parallelly from several languages [9, 18]. For instance, the course *Functional languages*, taught by Zoltán Horváth at bachelor's degree courses for programmers at the Faculty of Informatics, Eötvös Loránd University, follows the same structure. Of course, both the objective and the presupposed basics are different from those in primary and secondary education.

7. Action-oriented

Here the primary criterion is to understand how the statements operate i.e. to make students able to visualize what happens when the statements are being executed. In the simplest case one can give the statements of the language in another known language, perhaps in assembly language.

The example explaining `DO` statement below comes from a classical FORTRAN coursebook [6]:

```
    K=1                               DO 20 K=1,N
17  T=0.0                             T=0.0
    J=1                               J=1
18  T=T+A(I,J)*B(J,K)             18  T=T+A(I,J)*B(J,K)
    J=J+1                             J=J+1
    IF(J-N)18,18,20                   IF(J-N)18,18,20
20  C(I,K)=T                      20  C(I,K)=T
    K=K+1                         21
    IF(K-N)17,17,21
21
```

In a book on C# one can read the following explanation about the ++ operator [3]:

```
a = ++b;   // ⇔ b = b+1; a = b;

a = b++;   // ⇔ a = b; b = b+1;
```

The above examples show that one should not necessarily go back to another language to describe the operation of a statement, but one can define it with the help of other elements of the same language.

In a sense, a possible solution belonging here is when the semantics of the elements of a new programming language is defined with the elements of a "well-known" algorithmic language. This results in an extra educational profit: while the elements of a language are demonstrated, the students can practise programming in an algorithmic language, as well.

8. Sample task-based

According to this notion, if students are shown quite a lot of examples, they will be able to acquire a programming language well [7]. Here is a quotation from a book by Zsuzsanna Márkusz called "It is Easy to Write PROLOG Programs":

"*My PROLOG teaching experience has convinced me that the easiest way to learn programming is through sample programs. Therefore, instead of any scientific introduction (notations, definitions, theorems) the book foreshows twelve sample programs, which are explained in great detail.*"

Although it shows some resemblance to the task type-oriented method, their basic principles are different. There the root of the matter is that the set of tasks makes it necessary to introduce new language elements. As for this method, it is just the opposite: the language elements are given in the tasks and their build-up follows the language elements. That is why it is not certain that the acquired language elements will need to be used in the future and they might be forgotten if not practised.

8.1. A short evaluation of the above methods

We think that the *statement-oriented* method is unsuitable for teaching programming languages because a programming language is not equal to a set of statements. Behind a programming language there is always an idea, and in order to apply a programming language properly, it is inevitable to learn it[3]. Programming languages use basic language concepts like type, block structure, parameter pass etc., which might be different in various language types, or even in languages and their knowledge is connected primarily not to statements, but to languages. In each programming language a program has some structure, some constructing rule.

The *"Applying as a tool" approach* is the one that is needed in algorithm- and data-oriented teaching of programming, and thus this method can be used parallelly with the above programming teaching methods, that is with teenage students considering ICT as a carrier.

The *software technology-oriented* method is, actually, an improvement of the previous one (applying as a tool) for higher education, ICT specialists' education; so it can be a very powerful method there. On the other hand, in primary and secondary education it could have a role maximum in ICT vocational training.

The *task type-oriented* method is the only one that can be used in each level of primary and secondary education, where the main objective is the implementation and try-out of algorithms, and not a thorough knowledge of a programming language (quotation from the justification in the Hungarian National Curriculum: *It is enough to teach a programming language to that depth that is necessary for implementing and trying out algorithms. The language itself is not a crucial part of the ICT curriculum.* [21]).

The *language-oriented* concept may be excellent to summarize the elements of a language as completing language learning. For those considering information science as a carrier, it is also possible to introduce a new language that is fairly similar to the ones learnt before (e.g. after Pascal Delphi, C++ can be taught this way; or after C++, C#, ...), as in this case the students' previous language knowledge can be used effectively.

The *action-oriented* idea greatly resembles to the statement-oriented one, since it teaches statements, as well. On the other hand, here the definitions are given on the level of the "operation" of statement (i.e. how they work) instead of their "specification" level (i.e. what they should do). If used exclusively for beginners, it will not bring any success.

Teaching with the help of *sample tasks* is a "medieval" concept. This way one can train "artists" of programming and not its conscious doers.

[3]It is the "cost" of how to discover this world of ideas that qualifies the language itself. That is why it has such special importance in education. [14]

Note. The presentation of language teaching methods is somehow dangerous when relying on books on programming languages. The reason for this is that today the methodological background of books on programming languages is much weaker than that of those on natural languages. That is why the same book discusses a language in several ways from several aspects.[4]

References

[1] Alcock, D., Illustrating BASIC! *Cambridge University Press*, 1977. (Hungarian translation: Ismerd meg a BASIC nyelvet! *Műszaki Könyvkiadó*, 1984.)

[2] Horowitz, E., Fundamentals of programming languages, *Springer Verlag*, 1983. (Hungarian translation: Magasszintű programnyelvek, *Műszaki Könyvkiadó*, 1987.)

[3] Illés Z., Programozás C# nyelven, *Jedlik Oktatási Stúdió*, 2004.

[4] Jamsa, K., Lalani, S., Weakley, S., Web-programming, *Jamsa Press*, 1996. (Hungarian translation: A Web programozása, *Kossuth Kiadó*, 1997.)

[5] Lischner, R., Delphi in Nutshell, *O'Reilly & Associates*, Inc., 2000.

[6] Lőcs Gy., Vigassy J., A FORTRAN programozási nyelv, *Műszaki Könyvkiadó*, 1973.

[7] Márkusz Zs., PROLOG-ban programozni könnyű, *Novotrade*, 1988.

[8] Cantù, M., Mastering Delphi 5, *Sybex*, 1999.

[9] Nyékiné Gaizler J. (szerk.), Programozási nyelvek, *Kiskapu Kft.*, 2003.

[10] Porkoláb Z., Kozsik T., Zsók V., Új szoftverparadigmák nyelvi támogatása: a jelen oktatása – a holnap technológiája, *Informatika a felsőoktatásban'05. Debrecen*, aug. 24–26, 2005. `http://aszt.inf.elte.hu/~fun_ver/2005/papers/if05_paper_zsv.pdf`

[11] Stroustrup, What is "Object-Oriented Programming"? (1991 revised version), *Proc. 1st European Software Festival*, February, 1991. `http://www.research.att.com/~bs/whatis.pdf`

[12] Szentpéteriné Király T., Comenius Logo teknőcgrafika, *Kossuth Kiadó*, 2000.

[13] Szlávi P., A programkészítés didaktikai kérdései, *PhD thesis*, 2004.

[14] Szlávi P., Programozási nyelvek értékelése, *electronic script*, 1999. `http://digo.inf.elte.hu/~szlavi/InfoOkt/SzoftErt/Szoftverek%20értékelése.pdf`

[15] Szlávi P., Zsakó L., Methods of teaching programming, *Teaching Mathematics and Computer Science 1*, No. 2 (2003) 247–258.

[16] Szlávi P., Zsakó L., Delusions in informatics education, *Teaching Mathematics and Computer Science 2*, No. 1 (2004) 151–152.

[17] Szlávi P., Zsakó L., Temesvári T., Módszeres programozás: A programkészítés technológiája, *ELTE IK*, 2007.

[4]In most cases the same book can be used as a manual and a coursebook of a programming language. Moreover, it often contains programming know-how. Now let us compare them with books on natural languages, where one can find monolingual dictionaries, bilingual dictionaries, coursebooks, workbooks, books to develop speaking skills etc.

[18] Szlávi P., Zsakó L., Temesvári T., Programozási nyelvi alapfogalmak, *ELTE IK*, 2005.

[19] Turcsányiné Szabó M., Zsakó L., Comenius Logo gyakorlatok, *Kossuth Kiadó*, 1997.

[20] Végh Cs., Juhász I., Java – start! *Logos* 1999, 2000.

[21] Zsakó L., Az informatika ismeretkörei, *ELTE IK*, 2005.

Zsuzsanna Papp-Varga
Péter Szlávi
László Zsakó
Department of Media and Educational Informatics
Eötvös Loránd University
Budapest
Hungary

e-mail:
`vzsuzsa@elte.hu`
`szlavi@ludens.elte.hu`
`zsako@ludens.elte.hu`