Methods of teaching programming

Article · January 2003

DOI: 10.5485/TMCS.2003.0023

CITATIONS
12

READS
7,886

2 authors:

Péter Szlávi
Eötvös Loránd University
40 PUBLICATIONS **82** CITATIONS

SEE PROFILE

László Zsakó
Eötvös Loránd University
46 PUBLICATIONS **90** CITATIONS

SEE PROFILE

Methods of teaching programming

Szlávi Péter – Zsakó László

Department of Informatics Methodology, Eötvös Loránd University of Budapest
e-mail: szlavi@ludens.elte.hu, zsako@ludens.elte.hu

Abstract

Programming methodology is one of the oldest fields of IS education, and thus various methods have evolved for its teaching. While some of them could be used effectively in primary or secondary education, others are more suited for students in higher education. The methods themselves determine the structure and curricula of courses such as *Programming methodology*, *Data types and algorithms*, *Programming technology*.

Teaching programming is virtually as old as information technology itself. In the beginning, (before 1950) it meant the understanding of how computers worked and how they could be used. In this and the following era programming was an 'artistic' activity, programmers had to learn different tricks and gimmicks, and furthermore they had to find out new algorithms for each problem. The knowledge of programming was spread by word of mouth even in the 1950s. They used programming languages which were based on the machine, thus they were forced to spend more time working out the details than focusing on the essence of the problem. Meanwhile, they had to realize that the computers were slow and did not have enough memory capacity.

It became really important to teach programming after programming languages of higher level had appeared. On the influence of the previous era, however, it was based on teaching one particular programming language. The exercises were to understand the instructions of the programming language and to introduce how it could be applied.

Actually, teaching programming as we mean it today, dates back to the 1970s. The big break in the methodology of programming was brought by the book *Structures in Programming* (Dahl, Dijkstra, Hoare [DDH,D], Wirth [W]). Its valuable invention was that it used a few but well defined algorithm and data structures, it divided the problems into subproblems according to the principle "Divide et impera", it created abstract algorithms and data types independent of the programming language to solve programming problems.

At the end of the '70s the methodology of teaching programming was approached from another point of view as well, based on the way how children think. This method was developed mainly by S. Papert [P]. J. Hvorecky and J. Kelemen [HK] writes about a similar method although it is based on different problem classes.

Several methods have been developed for teaching programming since then. A few of them have already become out-of-date, most of them, however, are still in use at a certain level of education. Although some of them are related to methodology of program design, the majority of the methods are independent from them.

Here is a list of the most widespread methods in teaching programming:

- methodical, algorithmic oriented
- data oriented

- specification oriented
- problem type-oriented
- language oriented
- instruction oriented
- mathematics oriented
- hardware oriented
- model oriented.

We would like to note that the majority of teachers prefer to use several methods at the same time rather than stick to one single method.

1. Methodical, algorithmic oriented

This method, similarly to many others, covers the whole process of programming:

- problem definition, specification
  - algorithm and data structure planning, comprehension of the correctness of the algorithm
  - coding
  - testing
  - error detection, correction
  - efficiency control, quality control
  - documentation.

Each activity is to be dealt with separately. The methods and tools connected with the topic should be considered in each case. Algorithm elaboration is considered to be of primary importance in this method, thus most of the emphasis is laid upon this during teaching. There are algorithmic oriented elements in later phases as well.

The basic idea of algorithm elaboration is systematic arrangement. The first step includes general problem types and their general solution models, i.e. the programming theorems. (Although it is formally provable [SzZs1] that the general solution models of certain problem types are correct solutions to the relevant problems, this method substitutes formal proof for informal one.)

The second step is to reduce the given problems to programming theorems, i.e. to determine how to apply certain programming theorems. The speciality of this method lies in the fact that it examines where and what needs to be actualized both in specification and algorithm at the same time.

As the third step, the programming theorems should be combined so that more theorems could be applied at the same time (not lineally, one after the other). It is important to give the relevant (program transformation) rules not only for the specification but for the combination of algorithms as well.[HSzZs].

Data structure planning (a part of the subject traditionally called *Data Structures and Algorithms*) is to be processed here on the basis of the methodology of programming, i.e. data structure is a certain type with specification, structure representation and operation implementation [PSzZs].

A great number of decisions can be made during the coding phase depending on the definite or more general programming language. This method observes the vast majority of these decisions from algorithmic point of view. Thus the methods of conversion between iterative and recursive algorithms [SzZs2], the realisation of the usual programming structures

(branches, iterations, functions, operators, etc.) are dealt with this way as well. Apart from this, the method discusses the tools of user-friendly programming (e.g. the realisation of menu) in this phase. As regards the methods of testing, it handles the methods based on specification (known also as black box) and the ones based on algorithm (known as white box) as well [SzZsT].

In addition, the method of efficiency control is discussed with the help of algorithm as well. This means that it defines general problem classes according to execution time, reservation, and complexity (similarly to programming theorems), it gives the algorithmic schemes and ideas for the enhancement of effectiveness (for example, the principle of sequence segmentation can be used for the algorithms of logarithmic search, quicksorting, parallel maximum and minimum choosing as well as for determining roots by 'bisection method') [Zs,B].

Since the programming language does not play a primary role either in this or in the following two methods, it has little influence on the structure of programming knowledge, thus programmers instructed by this method will not be bound to one particular programming language.

One of the main principles of algorithm-oriented conception is that the designer can put himself in the executor's position thus acquiring informal ideas about the correctness of the algorithm. This way individual experience can be advantageous to sequential operation conceptions and experience of group operation and behaviour is beneficial to object oriented or parallel models.

2. Data-oriented

This method is similar to the previous one with the exception that it regards data structure and type refinement as primary. Its basic principle is that problem identification is regarded as type specification and it combines type refinement with algorithmic structures: [J1, J2]

- Cartesian product – sequence
- union, alternative data structures – branching
- multitude (set, sequence, hierarchical and network structures) – iteration
- recursively defined sequence (data recursion or recursive type) – recursion.

In the clear variety of the method the connection with the user can be regarded as type refinement as well. We receive the input (forms) and output (reports) formats from this.

While the stress was laid upon *programming theorems* in the previous method, here data processing standard problems (data input, listing, listing with hierarchical totals, copying, up-dating, etc.) are put forward, which join input data structures and output data structures. It is possible to give general data structures and algorithms for these just like for programming theorems [SzZs3].

This method emphasizes other problem types than the previous one. One characteristic of the algorithmic oriented method is that first it expands algorithmic structures, that is its main principle is the principle of *simple data structure – complex algorithmic structure.* As a contrast, data oriented conception follows the principle of *complex data structure – simple algorithmic structure*. The programs here can be based on the theory of 'one reading – processing – one writing' for a long time [BV].

3. Specification oriented

The notion of the method has a lot in common with the previous two ones. However, it considers formal specification to be the most significant part of the program development

process, the algorithm is derived from the specification automatically, then the code is created with the help of rigid coding instructions. Similarly to the first method, there are programming theorems here as well, but the algorithms of these are derived from the given specification [F].

While it is the transformation of the algorithm that is emphasized in the algorithmic oriented method, here more stress is laid upon the transformation of the specification. As a result, data structure and algorithm subjects may contain more theoretical knowledge, may be based on theorems and on their proof necessary for effective realisation. Since each phase involves mathematical elaboration, it is advisable not to begin this method without profound theoretical knowledge and being able to understand thet it requires abstraction skills [FNH].

4. Problem type-oriented

This method is fundamentally different from the previous three ones. Here programming is seen as a global activity, it cannot be divided into separate parts, and compared to the previous method, this one has got one essential feature, which is the fact that we always deal with the whole program. (This will be true for the following methods as well.) That is why it advances through each part one after the other while discovering new methods.

We start from a definite problem class here, which belongs to classical mathematics, usually with problems from number theory (divisibility, prime numbers, prime factor expansion), but this method is most successfully applied (especially as regards primary and secondary education) in completely different fields :

- graphics
- word processing
- common algorithms.

The essence of all of them is that a series of problems based on one another has to be solved. To solve the single problem we need new programming notions, elements and they are invented because they are needed for solving a specific problem. It has the advantage that new knowledge is derived from natural demand and not as an allegation. Besides, it is used to solve the problem at the same time and it is widely known that the highest level of understanding is the ability to use the newly acquired knowledge.

The method based on common algorithms is especially suitable for getting to know programming at an early age (in the first four years of primary education, or even in kindergarten), since it is built upon our innate knowledge, and programming notions, methods are derived from it. It is important to note that algorithms have long been taught in kindergartens and primary schools only it did not have anything to do with teaching informatics algorithmization [K].

We should note that this concept (i.e. algorithms work the same way as we would do it by hand) is applied to a number of methods of teaching programming.

5. Language oriented

This is one of the oldest methods, where the aim, just like in the previous case, is to produce an effective program. Another vital feature is that it is closely related to one particular programming language as it is reflected in the following examples:

One can often hear conversations like this:

- ’What are you studying in programming?’

- ’Pascal'

In other words: There are ’Pascal’ programmers, ’C’-programmers, etc.

The basis of the method is that it teaches one particular programming language and it introduces programming knowledge with the help of this language. Because the centre of this method is the programming language itself, it contains a number of language-dependent information, which might be remembered as general programming terms (an example is the knowledge connected to the use of DATA-READ-RESTORE instructions in BASIC language). This is the reason why programmers who learned one particular programming language have difficulty changing over to another language.

Another source of danger is that the complexity of a certain programming language element has hardly anything to do with the complexity of its application in programming and when teaching with the emphasis on the language it is not highlighted appropriately. Excellent examples of this are branching and pretesting conditional iteration instructions (in Pascal they are IF and WHILE), which are of about the same complexity from linguistic point of view, however, we deal with problems that require iteration in the solution much more than with the ones that include branching only.

Furthermore, there are several programming conceptions, activities (stack, queue, sorting methods, etc.) that are not related to a certain programming language element (at least not today concerning the programming languages taught), thus they might be excluded from the process of teaching.

In spite of this there are a certain number of successful examples. These are usually related to programming languages different from traditional programming based on the Neumann principle [NF].

6. Instruction-oriented

This method is similar to language-oriented method with the exception that it is based on a general language type instead of one particular language. This is the most important difference between the two methods, which only means that the problems caused by the use of one special language are solved, the ones in connection with generalization still exist.

The method defines general language elements, according to the Neumann principle:

- assignment, expressions;
- reading, writing;
- branching (IF-statement, Case-statement);
- iterations (counting, conditional pre- and post-testing);
- procedures;
- functions, operators;
- modules.

Apart from the difficulties of being bound to the language, there are all the disadvantages of the previous method as well, therefore this method might be considered dangerous.

7. Mathematics oriented

This concept is founded on the notions of another subject (which is mathematics in our example, but it could be another one as well). The problems to be solved are taken from mathematics, where the individual problems are based on each other in accordance with the principles of mathematics. Unfortunately, there is no guarantee that the structure will be either logical or complete from programming point of view.

Real temptation lies in mathematics itself with its 'sovereign' topics, special inner logic and proportions. There is no guarantee that these inner proportions can be synchronized with the real aim, which is programming. It is interesting to scan the teachers' book written by Miklós Simonovits and Margit Gémes [SG], as an early, yet of a very high standard, example of this tendency.

8. Hardware oriented

This method assumes that algorithmic knowledge cannot be understood without high-level programming language knowledge; programming language knowledge cannot be understood without assembly or machine code knowledge, respectively; assembly knowledge cannot be understood without the understanding of how the processor works; etc.

Originating from this appealing but incorrect reasoning, this methods tries to build up programming knowledge from the bottom, claiming that: "To be able to understand the structure of assembly instructions and to execute an assembly program, one should know how processors work. Having acquired assembly language it is possible to understand how the instructions of higher-level languages work. With the knowledge of the instructions of higher-level languages it is easier to understand how certain algorithms work."

As a consequence, this method states that there is no need for programming knowledge (in general: algorithmic and data modelling knowledge) except at university or college. This completely contradicts the fact that everyone needs to possess executive ability and algorithmic knowledge, as emphasized in public informatics.

9. Based on a model

In this method models are introduced to the students (algorithms, program codes) and they get information about programming by studying them. They can produce new programs by modifying the existing ones.

Experimenting plays a very important role here, pushing programming knowledge into the background. Students modify the programs that they have been introduced evaluating the received results and in case it meets their expectations the modification has been successful; if the result is not satisfactory, they have to continue experimenting.

Brief evaluation of the methods

In our opinion, the first two methods (algorithmic and data oriented) can be used towards the end of secondary education (for those preparing to work in informatics business), in informatics professional training and in higher education. Regarding motivation, algorithmic oriented method is more valuable because a greater variety of more interesting problems can be used.

Specification oriented method is to be used for students whose major is informatics at university (so-called elite training) as it can only be successful if the students have profound mathematical knowledge. Concerning mass higher education it might be introduced during the 4th or 5th year of university education.

The only method that is advisable to be used in itself at all levels of public education (primary and secondary), where the aim is to develop algorithmic way of thinking and not professional training is problem oriented method.

Language and instruction oriented methods are considered to be out-of-date and their application might cause too much danger while they might be less useful than others.

Mathematics oriented method is not thought to be effective in teaching programming but we must note that teaching mathematics with the help of programming can be very useful for those possessing programming knowledge because, as it has already been mentioned, the highest level of understanding mathematics is the ability to apply it (e.g. in programming).

Hardware oriented method (especially in its final form) can also be regarded out-of-date, which generates more interesting questions as well: e.g. Is it good to teach subjects connected with computer architecture in the first terms? Or should we teach certain parts of it later and more profoundly?

The method based on a model is an idea from 'the Middle Ages' and only artists of programming can be educated this way, professional programmers cannot. This method is useful for a number of geniuses, others can only potter using it.

In conclusion, here at the informatics teachers' training courses of University of Eötvös Loránd we have chosen algorithmic oriented method within the subjects *Programming Methodology* and *Data Structures and Algorithms.* (Naturally, we adjust teaching certain programming steps to the expected level of abstraction at the university; therefore we have to include a lot of formal knowledge for students studying to be informatics teachers.) Moreover, there are other subjects (*Computer Graphics, Methodology of Informatics Application*) as well which are based on this conception. We are doing this although we are aware of the fact that the vast majority of our graduated students will not use this method when teaching informatics.

References

[DDH] O.J. Dahl-E.W. Dijkstra-C.A.R. Hoare: *Structured Programming*, New York, Academic Press, 1972.

[D] E.W. Dijkstra: *A Discipline of Programming*, Prentice-Hall, 1976.

[W] N. Wirth: *Systematic Programming: An Introduction*, Prentice-Hall, 1973.

[P] S. Papert: *Mindstorms. Children, Computers and Powerful Ideas*, Basic Books, Inc, Harper Colophon Books, 1981.

[HK] J. Hvorecky-J. Kelemen: *Algoritmizácia, elementárny úvod*, ALFA, Bratislava, 1983.

[K] C.H.A. Koster: *Systematisch leren programmeren*, Educaboek, 1984.

[NF] A.E. Nicholson-K.M. Fraser: *Methodologies for teaching new programming languages: A case study teaching LISP*, CACM, 1997.

[SzZs1] Szlávi Péter, Zsakó László: *Módszeres programozás: Programozási tételek*, ELTE TTK Informatikai Tanszékcsoport, 1996.

[HSzZs] Harangozó É.-Szlávi P.-Zsakó L.: *Joining Programming Theorems, a Practical Approach to Program Building*, Annales Universitatis Scientiarum Budapestinensis. Sectio Computatorica 17, 155-172, 1998.

[PSzZs] Pap Gáborné, Szlávi Péter, Zsakó László: *Módszeres programozás: Adattípusok,* ELTE TTK Informatikai Tanszékcsoport, 1998.

[SzZs2] Szlávi Péter, Zsakó László: *Módszeres programozás: Rekurzió,* ELTE TTK Informatikai Tanszékcsoport, 1995.

[SzZsT] Szlávi Péter, Zsakó László, Temesvári Tibor: *Módszeres programozás: A programkészítés technológiája,* ELTE TTK Informatikai Tanszékcsoport, 1995.

[Zs] Zsakó László: *Módszeres programozás: Hatékonyság,* ELTE TTK Informatikai Tanszékcsoport, 1996.

[B] J.L. Bentley*: Programming pearls*. Communications ACM, 1984. feb.-nov., Vol.27, No.2-11.

[J1] M.A. Jackson: *Principles of Program Design*, Academic Press, 1976.

[J2] M.A. Jackson: *Structure-oriented programming*, in Program Transformation and Programming Environment, 169-180., Springer, 1984.

[SzZs3] Szlávi Péter, Zsakó László: *Módszeres programozás: Adatfeldolgozás,* ELTE TTK Informatikai Tanszékcsoport, 1995.

[BV] Bánné Varga Gabriella: *Programtervezési gyakorlatok*, SZÁMALK, 1989.

[F] Fóthi Ákos: *Bevezetés a programozáshoz,* Tankönyvkiadó, 1983.

[FNH] Fóthi Ákos-Nyékyné Gaizler Judit-Harangozó Éva: *Abstraction strategies in practice*, Annales Universitatis Scientiarum Budapestinensis. Sectio Computatorica 19, 75-92, 2000.

[SG] Simonovits Miklós, Gémes Margit: *Tanári segédkönyv Simonovits Miklós „Számítástechnika" tankönyvéhez,* OKKFT TS 4/1, 1991.

[KS] R. Kempf-M. Stelzner: *Teaching object-oriented programming with the KEE system*, OOPSLA'87 Proceedings, 1987.