Communicative approach to teaching programming

Václav Vrbík

Department of Computing and Instructional Technologies,
Faculty of Education, University of West Bohemia,
Klatovská 51, Pilsen 306 19, Czech Republic
E-mail: vrbik@kvd.zcu.cz

Abstract: There are many different theoretical researches, which deal with the content and quality of teaching. Their common aim is to improve the quality of teaching both for students and pedagogues. The aim of the paper is to give the basic information about methods of the communicative approach to teaching, in teaching programming. The communicative approach to teaching (CAT) is based on the optimisation of the amount and retention of knowledge, and the character of subject matter achieved by the higher motivation of students', and applying the knowledge in a real context. The communicative approach to teaching exploits a broad spectrum of methods, which support the general development of instructional ability of students, independent and cooperative solving of problems, and also object teaching. The centre of teaching is created by methods which use mutual communication, together with solving problems in pairs and in groups, and processes elaborating the independent solving of problems. The purpose of this paper is to suggest applying the CAT method mainly used in teaching foreign languages, to teaching programming by means of the Pascal programming language.

Keywords: communicative approach; teaching; programming.

Reference to this paper should be made as follows: Vrbík, V. (2005) 'Communicative approach to teaching programming', *Int. J. Cont. Engineering Education and Lifelong Learning*, Vol. 15, Nos. 3/4, pp.148–161.

Biographical notes: Václav Vrbík works as a senior lecturer and the Head of the Department of Computing and Didactical Technologies, at the University of West Bohemia in Pilsen. He holds a MSc in Technical Cybernetics from the same university, and holds a PhD in Technical Cybernetics from the Czech Technical University, Prague. His research interests concentrate on the development of educational software, programming languages, cybernetic education, and the pedagogical aspects of teaching programming languages.

1 Introduction

American linguist Noam Chomsky was involved in the foundation of this method at the beginning of the 1950s. He outlined the main ideas (Neuner et al., 1981) based on the fact that a language is not only a system of lingual constructions, but that its structure is richer. The authors who followed his work, added the subject of relationship between communication and society to the observation of abstract abilities of individuals. The purpose of teaching languages is the creation of the so-called communication capability – the ability to successfully transfer information (while considering the given knowledge and the situation) among the participants in communication, i.e. the ability to use the given lingual system reasonably and efficiently (Nunan, 1991). A communicatively qualified speaker gains further knowledge and capabilities while acquiring language constructions, which enable him to realise whether the message is suitable for a particular situation, how it is possible to comprehend the message, and what the consequences of the comprehension of the message are Brumfit and Johnson (1979).

A language is created not only by formal syntactical entries collated to single constructions and description of their meaning; the information about using the given construction during representation, as well as its inclusion into the process of communication, are also its indivisible parts. That is why special attention is paid to motivation, support of students' creativity, their cooperation, and guiding them towards independent thinking. The preparation of textbooks, instructional materials and the selection of suitable teaching methods are less important than driving the creativity of students. Special emphasis is put on the superior pedagogic-psychological education of the teaching staff. In addition, the function of pedagogues in the teaching process is definitely one of the most important factors in education. Traditional approaches as well as alternative approaches emphasise the key role of the teaching staff in educational programs. It is possible, however, to find out the differences between the CAT and traditional approaches when considering the role of pedagogues, especially in the fields in which the approaches mentioned, differ strongly.

One of the fundamental differences between traditional and alternative approaches consists in the method of imparting new bits and pieces of knowledge. The teaching model called ‘jug and empty glass’ is applied in traditional teaching approaches. This model is based on ‘filling’ students with completed information. Bits and pieces of knowledge are presented as definite and they do not usually initiate any discussions. As a result, it worsens the students’ interest and deepens their passivity. In this case, the role of the teacher is defined as the guarantee of the truth – the source of knowledge that students are forced to adopt. The alternative approaches prefer the model called ‘principle of a hunter’, in which the students are considered to be hunters trying to trace and catch a bag (represented by bits and pieces of knowledge). Pedagogues provide students with the means (tools) for hunting, and thus, help them to determine the way of hunting. Therefore, the pedagogue is considered the guarantee of the method. This model improves the students’ activities and their involvement in the teaching process. The bits and pieces of knowledge are accessible to critical appreciation, and students have sufficient space to express their opinions in a constructive discussion. The utilisation of this model develops the independence and positive motivation of the students better.

When considering the above-mentioned, the CAT can become one of the alternatives in teaching programming languages that can enforce students’ interests, and independence in this field. However, applying of the CAT methods used in teaching foreign languages, to the teaching of programming languages is difficult, due to the many different interpretations of the CAT, resulting in difficult specifications.

Thus it is necessary to consider the differences between programming languages and natural languages (e.g., the English language), find out the analogy between them, analyse them and state their basic equal and different properties. *These properties define the possibilities of the application of particular methods*. This process is necessary for the preparation of instructional materials, and defining the educational program with regard to the CAT. The English language and Pascal programming language were chosen for description purposes in this paper.

2 Communicative approach to teaching Pascal

What are the bases in experiments using the CAT in teaching the Pascal programming language? The Pascal programming language was designed to provide a tool for teaching programmers. As Pascal was designed as a didactic tool, it was required to use objective and comprehensive constructions. The English language was used to design lexical elements (keywords, etc.) of Pascal. Therefore, Pascal is similar to the English language in particular fields.

It is possible to find out *similarities* or even coincidences with regard to lexicology, syntax and semantics. Pascal contains many lingual constructions directly derived from the English language. Some lexical elements (e.g., key words, standard identifiers) consist of one-word expressions designed by the direct transcription of the English expressions or the adjustments of the English words (e.g., by the abbreviation of the key word *var*). Such adjustments improve the arrangement of the source code and make the understanding of the code easier, especially when students can speak English at least at the elementary level. These constructions take the semantic content from their equivalents in the natural language as well. Some more complicated constructions taken from English correspond to the English language constructions, such as the structure and meaning of the, *if…then…else*, conditional expressions.

A natural language uses standardised lingual constructions for the expression of different functions (e.g., requesting information or placing an order in a restaurant), and for the communication control. It is possible to create constructions used for solving particular kinds of problems using Pascal (e.g., acquiring data from an I/O device or computing the *n*-th root of an integer number) as well as recognise the constructions used for communication control (the number of such constructions is significantly lower, in comparison to the English language).

A program in an arbitrary programming language can be considered as a customised writing of lingual constructions originally created in a natural language. It is possible to find out syntax and semantics in any programming language. The syntax defines the set of allowed symbol combinations that determines formal writing rules in a programming language. The semantics describes the meanings of the syntactic structures. As a result, it is possible to recognise the particular form of vocabulary and grammar of the programming language that may be analogous to its counterparts in natural languages.

Therefore, we can expect that it is possible to apply the CAT methods used in teaching English to teaching the Pascal programming language.

On the other hand, it is necessary to consider the *differences* resulting from the focused speciality of Pascal in expression options, and the specific relationship of the programming languages and real life.

There is an apparent difference in the origin of both the languages when comparing English and Pascal. The English language is a natural language with its real social and cultural background. It has been developing for a long time and the development is supposed to continue in the future. On the contrary, the Pascal programming language is the so-called artificial language – a language created by humans in order to fulfill specific requirements (similar to Esperanto).

A significant difference between these two languages is the number of lexical elements and syntactic constructions. The English language has significantly more extensive vocabulary and grammar due to its origin. On the contrary, the Pascal programming language was designed in order to meet special requirements. Hence, its facilities of expression are limited; it is not designed to express emotions. The role of a computer in social communication is important when determining the distinctive Pascal teaching properties. The basic premise is that the computer is not a 'live entity' that is able to be conscious of itself. It does not have its own personality and trains of thought. A characteristic property of the computer as a participant in communication is passivity. As a result, Pascal acts as a 'unidirectional' language.

Let us have a look at problems resulting from the differences, which should be considered when using the CAT in teaching Pascal. The usage ratio of the natural language and the target language (the English language and Pascal) is different when using the CAT. The English language teaching supports communication efforts from the very beginning. The translation to the mother tongue is used wherever or whenever it is useful for teaching, but its usage is limited to a minimum. Communication during teaching programming, in the Pascal language, is held in the native language although it always contains a certain part of the target language. The Pascal compilers are not able to decode information given in a natural language. Thus, students are forced to use the target language during programming. The aims of the CAT in teaching Pascal should be the increase of programming language representation during the interaction within a user group (making the communication with computers more intensive), and enforcement of the programming language as the main communication means achieved by the utilisation of activities used in communicative English language teaching.

There is a question that considers the possibility of using a programming language for mutual communication among students. The Pascal language has different specifications. For example, the entry task defined in a natural language is more suitable. It is theoretically possible to stimulate mutual communication among students in a programming language by means of simple task entries during students' activities, that could be used for syntax and semantics training without using a computer. This method has two main problems. The problem of creating a sufficiently motivating situation that would force students to communicate in a programming language, and the problem of sufficient Pascal knowledge that guarantees the instruction processing, with such a low level of errors that it doesn't make the tuition slower due to misunderstanding.

Communicative foreign language tuition also tries to apply the target language during tuition organisation. Thus, for e.g., instructions for work with instructional materials (opening books, etc.) or division of students into groups are given in the target language. Frequently repeated instructions (e.g., creating work groups with a particular number of members) can be transferred using non-verbal communication by means of specially prepared symbols (pictures, gestures, etc.). These symbols are often more persuasive and less time-consuming than verbal instructions. They can also be used as a supplement to the verbal instructions. The lesson management performed by means of the Pascal language does not seem to be effective. The non-verbal communication usage is directed by the same rules as the one used in teaching foreign languages. The most important is the utilisation of the same symbols for the same instructions in order to prevent students from getting confused. It is necessary to follow the directions that make passing instructions to students and pedagogues as easy and short as possible when using alternative tuition management methods.

The specific way of communication with computers in tuition, coheres with another difference between the communicative and traditional approach to teaching; students' mistakes. The traditional approach requires absolute correctness. Errors are strongly undesirable and they are usually immediately corrected. The CAT does not consider errors in an absolute way. It considers them in a particular context. Attention is paid to comprehensible and fluent speech during foreign language tuition. For example, giving instructions has to be correct, but the discussion should be focused on giving correct information. Grammatically correct expression is not the most important thing. This approach to teaching Pascal has to be adjusted to different characters of students and different properties of computers, when considering the communication point of view. The students have the opportunity to correct errors, or complete the meaning of a wrong or incomplete piece of information during their interaction. The programming language compiler is able to detect lexical and syntactic errors only when parsing the source code. It can suggest the method of correcting the error (e.g., 'semicolon expected'). Of course, it does not correct the wrong part of the given information (e.g., wrong syntactical source code structure) and it is not able to correct semantically wrong constructions at all.

The compiler requested correction of all of the formal errors present in the source code (actually the negative response to students' effort) can have a negative influence on the students' motivation; it can even frustrate the students and result in the loss of interest in communication. At this point, the teacher's approach plays the key role. Correction of the formal errors by the student himself (herself), can also have a positive influence, due to the feeling of success that the student experiences on being able to cope with the formal writing of the source code. Moreover, if the program does not work correctly, there is the confidence that errors occur in the semantic structure of the code only.

The application of the CAT in teaching programming brings many changes to the tuition concept, in both methodology and social aspects. The author's knowledge obtained by applying the CAT in teaching programming proves the high efficiency of the CAT when considering students' activity, their independent individual initiative, and improved task solution results. The diversity of such methods helps to upgrade the natural recognition capabilities of the students and thus develop their positive motivation. There is a significant mutual interaction growth. The students unwittingly acquire the pieces of knowledge necessary for social communication, they also learn how to argue and critically evaluate their own ideas. Close cooperation during the task solutions strengthens the social bindings within the groups of students. Dialogues and discussions used in the tuition process are a positive contribution to the global level of knowledge, and allow the poor students to participate better in the problem solution.

Practical utilisation of the CAT has higher requirements for the teachers' preparation than the utilisation of traditional approaches. The main difference is in instructional materials preparation. The materials usually have to be adapted to the CAT, or have to be created from scratch if the tuition uses methodologies not used by traditional approaches (e.g., didactic games). However, some materials (tests, examples, etc.) are easily adaptable and their usage makes the preparation less time-consuming.

3 Methods of the CAT in teaching programming

3.1 Dialogic methods

A discussion, a chat, or alternatively a dialogue belongs to these methods and they suppose a cooperative involvement of a minimum of two participants. The intention of these methods is to bring about a situation, in which students will solve an assigned problem by mutual communication. Students are activated by suitably chosen questions or by formulating a problem so that they can combine the subject matter with their actual knowledge; they judge the problem, design its solution and evaluate these solutions again together. The task of the teacher is to get the activity ready (stimulation to communication, placement of the students in the classroom, materials and so on.) and monitor its course. It is important for the students to respect the opinions of the others and for each student to have a possibility to present his or her thoughts. If necessary, teachers help students to formulate suitable questions, which support independent thinking and the development of creativity (‘how…?’, ‘why…?’). During activities teachers ask questions only if it is necessary for maintaining communication or introducing all the aspects of the problem; the main role in the activities is played by students. These methods fit for a common solution of certain questions (e.g., searching the most effective algorithm for the given task), clearing doubtful areas of the subject matter, and they are often used to evaluate single activities, to evaluate work in lessons, etc.

Examples of the application of the dialogic method

A well-arranged record of a source text enables fast orientation and makes understanding the function of the program easier. Fundamental habits of the production of well-arranged source texts of programs must be demanded from students from the very beginning, because they create an important component of a practical application of the language. The following three examples show the same program from function views (Vrbík, 2000) in different stages of the design lucidity.

Example 1: Very low lucidity (minimum division into lines, slack descriptive identifiers of variables, missing communication under program run):

```
program MaxI;

const k = 5; var d : Real; c, a : Integer; b : array [1..k] of Real; begin

for c : =1 to k do Read(b[c]); d := b[1]; a := 1;

for c := 2 to k do if b[c] > d then begin

d := b[c]; a := c end; Writeln(d, a) end.
```

Example 2: Lower lucidity (relatively good division into lines, bad or missing indenting of a text, identifiers of variables are descriptive, but upper cases are badly distinguished, missing communication under program run):

```
program MAXI;

const NMAX = 5;

var BMAX : REAL;

I, IMAX : INTEGER;

B : array [1..NMAX] of REAL;

begin

  for I := 1 to NMAX do

  READ(B[I]);
  BMAX := B[1];
  IMAX := 1;
  for I := 2 to NMAX do
  if B[I] > BMAX then begin
  BMAX := B[I]; IMAX := I end;
  WRITELN(BMAX, IMAX)
end.
```

Example 3: Well-arranged text (division into lines, indenting of a text, descriptive identifiers of variables, lower cases, efficient use of commentaries, good communication with the user):

```
program Max1;
{It searches maximum from setting sequence of numbers}
const Nmax = 5;
var Maximum: Real;
    I, Imax: Integer;
    Posl: array [1..Nmax] of Real;
begin
  WriteLn ('Setj ', Nmax,' of numbers, divide them by a key <ENTER>:');
  for I := 1 to Nmax do
    Read (Posl[I]);
  Maximum := Posl[1];
  Imax := 1;
  for I := 2 to Nmax do
    if Posl[I] > Maximum then
    begin
      Maximum := Posl[I];
      Imax := I
    end;
  WriteLn('Maximmum number is', Maximum:7:3);
  WriteLn('It was set like', Imax, '.')
end.
```

These examples can be used very well as stimulation for dialogic methods of teaching. According to the number of students in the group, the teacher prepares a different number of examples of the same source text, from which only one will be digestedly modified. The condition is, that they should distinguish identifiers of variables in single examples, and if need be, also, further used identifiers. The teacher divides students into work groups; each group receives one example. The students' task is to judge the function of the program in a certain time-limit (for the presented instances, one or two minutes). After the time-limit is over, they interchange the examples with another group, and the activity is repeated.

After ending the mutual handing of examples, a discussion follows, which can be in progress within single work groups or in the whole class. The conclusion is formulated jointly for the whole class. The purpose of the discussion is to find out in which instances students were doing (estimating the function) best, in which badly, and why. Concrete pieces of knowledge are then generalised by a determination of fundamental rules for writing well-arranged source texts.

Another possibility of using dialogic methods is a discussion over an incomplete source text. Students are divided into groups and a section of the source text is introduced to them. Particular program sections are divided so that they, together, create a full source text. Students in groups discuss a probable object of their section, confront other groups with their records and subsequently they make up a program.

Another modification of this activity is a discussion on the function of a section of the source text in groups only. Then the rest of the program based on the supposed function is written in the groups. Subsequently, newly risen programs are contrasted in the whole class. During this variant, it is possible as well, to use sections of different source texts, so several programs arise and their function can be explained by single groups, for example, by the so called mutual teaching. During this method, after creating the program, one or two members of a group shift into a further group, where they discuss the programs of both groups. The shifting of students continues until all groups have had a chance to discuss the arisen programs.

3.2 Heuristic methods

It rises from dialogic methods, but it is not based only on a discussion during which actual pieces of knowledge are applied to a new subject matter together with the subsequent conclusions drawn. For a correct formulation of a solution to the problem, it requires searching, and mainly verification of new pieces of knowledge by students themselves. Teachers motivate students by a suitable formulation of the problem, similar to using dialogic methods, and monitor the running activity. The choice of the topic is important, students should not know the solution in advance, but the majority of students should be able to solve the problem. On the basis of the results of the work of single students, a final statement is formulated. This method is used for detecting new common regularities and concrete qualities of phenomena. Its implementation is more time-consuming than in dialogic methods, but the pieces of knowledge adopted during its use are more permanent, thanks to the involvement of more complex trains of thought. In combination with inductive and deductive techniques they belong to the most used methods of the communicative approach to teaching in general.

We illustrate the application of a heuristic method on the following example. Obviously, there are two approaches to the analysis of constructions in a programming language. The first approach deals with findings on what will happen if something is set (in a natural language–what will happen if something is said). The second approach is based on the question of what has to be set if something has to be reached (in a natural language–what to say if something has to be reached). Some students tend to take the first approach, i.e. they strangle different commands and watch what will happen. They should be led, rather, to the second way, i.e. first to learn to manage fundamental situations and then, as a supplement, use the method of attempt and error, to the specification or location of a better variant of the solution.

Examples of the application of the heuristic method

Watch the following program. What is its function? Can you identify something new in it? Write and let the program run.

```
program Numbers;

var A, B, Total, Difference, Product: Integer;

begin

  Write('Set the first complete number: ');

  ReadLn(A);

  Write('Set the second complete number: ');

  ReadLn(B);

  Total := A + B;

  Rozdil := A – B;

  Product := A * B;

  WriteLn(Total);

  WriteLn(Difference);

  WriteLn(Product)

end.
```

Before the startup of the program, students in groups analyse its supposed function and output. Some identifiers give marked help. Innovations are assignment statements, fundamental mathematical operations, the use of several variables in the section of the declaration of variables *var*. Students compare their judgment with actual records after startup. Adjust the program output so that every result is stated, for example, in the form ‘13 + 42 = 55’ or ‘the product of numbers 13 and 42 is 546’. It is possible to use further variants, e.g., print-out of all results on one line. Suggest also values of input variables (*A*, *B*) so that the results are out of range of the type *Integer*. Set also negative ness of input variables, and, also values so as to hold *A* < *B*. In the program *Numbers,* consider setting a fractional expression at least into one of the input variables *A* or *B*. Determine whether it is necessary before startup to change the program so that it can work with fractions. Provided students do not consider the change necessary, let them start the program (using Turbo Pascal, after setting the fraction, the computer announces an error ‘Invalid numerical shape’, which indicates that this shape of input is inadmissible). Bring in the type *Real*. Change only the declaration of variables *A* and *B* into type *Real*, output variables are kept behind the type *Integer*. Ask whether it is necessary to perform further changes besides the type of input variables. Provided students do not consider further changes necessary, let them start (at using Turbo Pascal the translator reports an error of the 'Type mismatch', which indicates incompatibility of types). Change the type output variables also into *Real*. Emphasise that the concept 'real number' is not used for integral numbers in Pascal. Pay attention to low lucidity of the exponential shape of the output. Consider carefully whether to use the concept of 'fractional' or 'integral' number for values of the type REAL at the beginning of teaching.

3.3 Problem methods

Their usage (including the production of projects) is directed especially to solving more extensive and complex exercises. The extent of projects however can be different - from shorter projects, whose solving does not overrun several hours, up to yearlong projects. For achievement purposes it is usually necessary to solve several partial exercises, from which the problem is assembled. If it is not an individual project, the key role is to divide students into cooperative and positively motivated work groups (into two or three groups maximum, each of which can process their own project or participate in the common project), where all members of the group are responsible for the final result of the project.

Examples of the application of the problem method

Here we demonstrate an exercise for the calculation of factorial, which is often found in different phases of teaching, e.g., during the introduction of the concept cycle with firm arithmetic repetition, during the introduction of the concept user' s function or during the introduction of recursive programs. It is also possible to illustrate here the using of further integral and real types of definers in Turbo Pascal, which 'magnify' the range of the types *Integer* and *Real*.

```
program Faktorial;

var N, I, Fakt: Integer;

begin

  Write('Set a natural number: ');

  Read(N);

  Fakt := 1;

  for I := 1 to N do

    Fakt := Fakt * I;

  WriteLn(N, '! = ', Fakt)

end.
```

The program is an accurate transcription of the correct algorithm, so we could expect that it will solve a given exercise correctly. However we obtain the following results:

6! = 720

7! = 5040

8! = –25216

This case, when the ‘computer does not count correctly’, can be used for a more detailed illustration of the concept of data type. The type *Integer* is characterised partly as a set of admissible values, and partly as a set of acceptable operations, which can be performed with the data of the given type. In contrast to mathematics, the set of values of the type *Integer* is final (it represents the interval –32,768–32,767). Value 8! = 40,320 lies out of the mentioned interval and therefore the computer ‘cannot display’ it. Students will no longer regard a computer as an unmistakable calculator and they will begin to be interested in how to solve such a simple task as the calculation of factorial. The first possibility, which they usually choose, is the ‘expansion’ of the interval of viewable values. Then we can acquaint students with the type *Word*, whose extent is 0–65,535, but preferably with the type *Longint*, which works with integral numbers in extent –2,147 483,646–2,147,483,647. In this case we receive the following results:

8! = 40320	9! = 362880	10! = 3628800	11! = 39916800
12! = 479001600	13! = 1932053504	14! = 1278945280	15! = 2004310016
16! = 2004189184	17! = –288522240	…	34! = 0

The results up to 12! are correct, results 13! to 16! are ‘suspicious’ (a small number of valid figures), 17! is evidently mistaken, from 34! on the result is zero. We did not get very far. Then students can be persuaded that not even real types mean progress forward, at the most it is possible to obtain exactly 18!, then the rounding begins.

It is necessary to explain to students that the mistake is not in the computer, but in a wrong strategy of solving a problem. How to change it? So far we have held the results of the product $1 \times 2 \times ... \times n$ as one value, now we will hold them as a sequence of values (the length of the sequence will be identified only by the length of the field type which we will use for storage). The program below (Drlík and Hvorecký, 1992) can be an inspiration for other students’ programs or projects (the result is in the field Figure):

```
program Faktorial1;

const Max = 5000;

type TRange = 0..9;

var I, J, Carry, Product, Most, N: Integer;

Figure: array [0..Max] of TRange;

begin

  Write('set N = ');

  ReadLn(N);

  for I := 0 to Max do Figure[I] := 0;

  Figure[0] := 1;

  Most := 0;
  for I := 1 to N do
  begin
    Carry := 0;
    for J := 0 to Most do
    begin
      Product := Figure[J] * I + Carry;
      Carry := Product div 10;
      Figure[J] := Product mod 10
    end;
    while Carry <> 0 do
    begin
      Most := Most + 1;
      Figure[Most] := Carry mod 10;
      Carry := Carry div 10
    end
  end;
  Write(N, ’! =  ’);
  J := Max;
  while Figure[J] = 0 do
    J := J – 1;
  for I := J downto 0 do
    Write(Figure[I]);
  WriteLn;
  WriteLn(’number of figures ’, J + 1)
end.
```

The following result of the work of this program proves that the problem is solved:

50! = 30414093201713378043612608166064768844377641568960512000000000000

There are 65 figures.

3.4 Didactic games, drama techniques

There is an extensive group of methods that can transfer pieces of knowledge in enjoyable forms and deeply encourage interest of students in learning (Milkova, 2003). They are exploited especially for unblocking, and at the same time for a good teaching atmosphere at the beginning, or during a lesson, for introducing new topics, vivid demonstration, recapitulation of the subject matter and its practice. They develop communication skills, cooperation and positive relations among students; they help to harden positive social structures in the class. These methods are used in combination with other methods (e.g., dialogic), they usually do not make up the content of the whole teaching unit, but they are a very important part of the communicative approach.

Examples of the application of the didactic game 'Assembling of the source text'

Students are divided into work groups; each group receives an envelope with a cut up source text of the program. The division of the text depends on the level of students, object of the activity and complexity of the program (for this activity we do not usually use source texts transcending 40 lines). The task of students is to put the cut up parts together so that no part is left out, and at the same time, a syntactically and semantically correct program arises. If the cut up text leads to more correct solutions that correspond to both conditions, any of them can be accepted. Depending on the connected activities, all groups can obtain the same or different source texts. When cutting the text, it is necessary to keep straight margins so that students have no possibility of assembling the text according to the shape of the margins. It is possible to modify this activity as a contest.

In the following example of a possible division of the program, the margins of single parts of the text are marked by changes of the tone.

```
program Bindec;

var Fig2, Fig10, Numfig, I: Integer;

begin

  Write('Number of figures of dyadic number notation: ');

  ReadLn(Numfig);

  WriteLn('Dyadic number notation, divide figuresby a gap: ');

  Fig10 := 0;

  for I := 1 to Numfig do

  begin

    Read(Cislo2);

    Fig10 := 2 * Fig10 + Fig2

  end;

  WriteLn('Target value: ', Fig10);

end.
```

4 Conclusion

The above-mentioned survey focuses only on the fundamental characteristics of commonly used methods of the communicative approach to teaching, and their possible application in communicative teaching of a programming language. The application of the methods of the communicative approach to teaching, in teaching programming, depends on the enthusiasm of teachers to accept this relatively extraordinary concept of teaching a technical subject, their will to devote a lot of time to the preparation of materials and activities, and last but not least, on the will to re-value the traditional concept of the role of pedagogues in the pedagogical and educational process.

References

Brumfit, C.J. and Johnson, K. (1979) *The Communicative Approach to Language Teaching*, Oxford University Press, Oxford.

Drlík, P. and Hvorecký, J. (1992) *Informatika – náčrt didaktiky*, *Pedagogická fakulta v Nitre*, Nitra.

Milkova, E. (2003) 'Teaching and learning using multimedia', *CBLIS: New Technologies and Their Applications in Education*, University of Cyprus, Nicosia, pp.30–36.

Neuner, G., Krüger, M. and Grewer, U. (1981) *Übungstypologie zum kommunikativen Deutschunterricht*, *Langenscheidt*, Berlin und München.

Nunan, D. (1991) *Designing Tasks for Communicative Classroom*, Cambridge University Press, Cambridge.

Vrbík, V. (2000) *Programování 1. ZČU v Plzni*, Plzeň.