Cognitive Aspects of Mathematics-aided Computer Science Teaching

Katalin Bubnó[1], Viktor László Takács[2]

[1]Doctoral School of Mathematical and Computational Sciences, University of Debrecen, Egyetem tér 1, 4032 Debrecen, Hungary, kbubno@lib.unideb.hu

[2]Department of Business Informatics, University of Debrecen, Böszörményi u. 138, 4032 Debrecen, Hungary, takacs.viktor@econ.unideb.hu

Abstract: Some years ago we started an experiment with teaching an algorithmic thinking method, in an old-new approach and with a newly developed ecosystem. We planned our method for the future, when students will use their own devices in schools, and computer programming should usually be integrated into various teaching environments, including teaching mathematical problem solving once again. Our method fits into analogy-based pedagogical research, which focuses on problem solving in computer science. We presented and examined our method at the 2017 CoginfoCom Conference, in the sense of Mathability. There, we also asked further questions concerning the efficiency of analogy-based computer programming teaching methods [2]. In this paper, we would like to answer these questions.

Keywords: problem solving; mathability; mathematical psychology; novice computer programming

1 Introduction: Analogy-based Algorithmization

At the beginning of Hungarian computer science teaching in public education (1980s) it was natural that teaching computer science meant teaching computer programming. It was also natural that mathematical problem solving and computational problem solving have common origins and tools. That was the reason why the first teachers, who started teaching computer programming in schools, were Mathematics teachers. The strong relation between mathematical and computational thinking enabled them to easily master and teach the basics of programming. As times changed, computer science and computer science teaching changed as well. Nowadays, the common origin, tools and thinking methods are not so obvious. Infocommunication tools became more and more complex and teaching the use of ICT tools is a priority, as opposed to computer programming.

Some years ago, we started an experiment with teaching an algorithmic thinking method, in an old-new approach and with a newly developed ecosystem. We planned our method for a future, when students will use their own devices in schools and computer programming would usually be integrated into various teaching environments, including teaching mathematical problem solving.

We introduced our method in 2013 ProMath Conference [1], and four years later at the 2017 CoginfoCom Conference, we summarized and demonstrated the mature method, we examined the method and the programming environment we used, in the sense of Mathability, and we presented some results from related teaching experiments [2].

The conception of mathability was first introduced in the 2013 CoginfoCom Conference [3]. Mathability research is intended to bridge the gap between cognitive infocommunication and information technology education by modeling mathematical thinking in computer problem solving. Former mathability research enumerates various cognitive aspects to take attention [6, 7, 8, 11, 20], or they try to match the mathability aspects of existing taxonomies [6, 7, 8, 9, 10, 11, 20]. Furthermore, we can find concrete classroom experiments and questionnaire research in different ages and education levels, to evaluate IT tools in the sense of mathability [6, 7, 9, 10, 20]. In [28] there is a review of Mathability and wider scale of education subjects of the former CoginfoCom conferences.

In this paper we want to expand the list of aspects of cognitive theories with those that have already been applied successfully in teaching mathematics.

At the same time, we want to introduce Blockly Code [12] programming editor, as a kind of new learning environment. Virtual reality systems, as new learning environments and especially 3D VR systems are relevant topics in cognitive infocommunications, from the beginning of Coginfocom conferences. Several studies prove the topicality and operability of these environments [29, 30, 31, 32, 38] and their benefits in cooperative learning [29], even in enterprise environment [37].

Furthermore, cognitive infocommunications subjects were completed with memory performance examinations [33, 34, 35, 36], that is in our inquiry as well.

Our method is based on the well-known mathematical problem solving model by George Pólya [4]. In Hungarian schools it is the most common and well-known mathematical problem solving model. Children from the first classes of elementary school learn this method, for solving simple word problems. We showed the analogies between this model and computational problem solving, and demonstrated some examples as well [1].

A great part of mathematical didactical literature studied the difficulty of problem solving from several aspects, for example, from the psychological aspect [5]. We think computer science education has to investigate the difficulty of problem solving also from the psychological point of view, because nowadays, in an

Information Society, it is a critical skill that a young people can learn computational thinking or not. Furthermore, if a youngster is not able to acquire this knowledge, what is the reason for their failure.

We think that children cannot easily recognize the relationship between Mathematics and computer programming, and they believe, computational thinking is a very difficult, complicated, new knowledge for them and that is the reason why they experience fear. We think, we can help them with our analogy-based approach, to help them learn that computer programming is the natural continuation of mathematical problem solving and it requires the same skills and thinking, as Math and nothing new.

Our method fits into analogy-based pedagogical research, which focuses on problem solving in computer science. In 2014 Coginfocom conference, Szi and Csapo [6] described various factors which influence human mathematical abilities. They mentioned analogy searching as one of the most important generic concepts which refers to the existence of mathematical intelligence in a human. They said: "Searching analogy (SA), which reflects the ability to recognize structural associations, is also an important foundation for the development of mathematical thinking. It has been argued that it is the structure of concepts and problem solving that distinguishes mathematics from other natural sciences. In this sense, the ability to find useful analogies is a crucial component of good mathematical abilities, as reflected in the various mathematical intelligence tests."

We believe, our method offers 'useful analogy' for children to solve problems with computers, and in this paper we should prove it with some of our measuring results.

In the CoginfoCom conference we propose a final test in our experiment and seek to answer the following questions:

1) Are the analogies in teaching methodology helpful or not, when they have been achieved as an established method from elementary school?
2) Can children recognize mathematical analogies or not in certain programming environments?
3) Can this approach help overcome fear and aversion of computer programming?

2 About Our Conscious Problem Solving Method with Computer Programming

In Hungarian schools teaching mathematical problem solving started in early the school years. Children repeat as a 'mantra' the engraved method:

1) Gather data from text
2) Create a plan
3) Count
4) Check and Answer

But from Pólya's, How to solve it [4], we know, there is also a step: 'Looking back'. Looking back means discussing the problem, examining the solutions, asking new questions related to the problem, forming the problem into another problem (posing new problems), formalizing and generalizing the problem.

In [2] we demonstrated via an example how we can recall Pólya's model and use it for teaching the base elements of computer programming via classic word problems. We also presented how to use 'Looking back', for teaching how to formalize and generalize a certain problem and evaluate a students' work. We presented that 'Looking back' precedes generalizing, debugging and optimizing the algorithm. Furthermore, we mentioned that during evaluation, we specified 5 critical factors of assessment: initial data extraction from word problem (Data), Problem solving correctness mathematically (Math), Problem solving correctness algorithmically (Alg), Algorithm checking (Check), Answering the problem (Answ).

In [2] we also presented a system of criteria to measure the level of discussion of the problem. We can see this discussion pyramid in Figure 1.

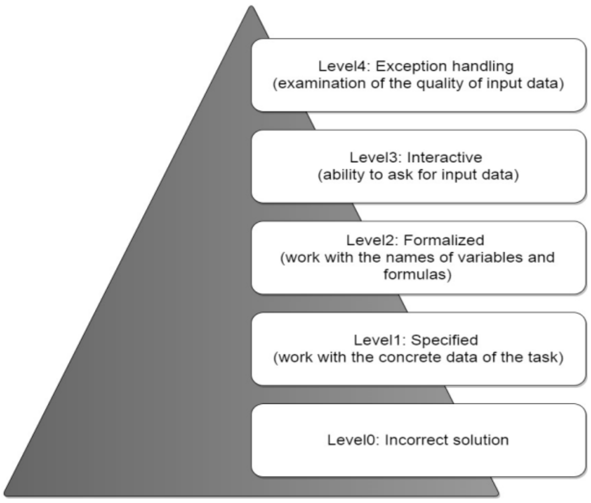

Level4: Exception handling
(examination of the quality of input data)
Level3: Interactive
(ability to ask for input data)
Level2: Formalized
(work with the names of variables and formulas)
Level1: Specified
(work with the concrete data of the task)
Level0: Incorrect solution

Figure 1
Discussion pyramid

3 Blockly Code

In [2] we presented, that we chose Blockly Code [12] programming editor, the base of many novices block based programming languages to work with. Furthermore, we showed, that Blockly is a high level mathability tool. We could say, Blockly is an end-user-friendly pseudocode. It is user friendly and comfortable for novice programmers as opposed to text-based languages. When we code with Blockly, we write algorithms in a formal language, but we do not have to memorize the elements of the language, just search for them in the structured toolbar, and drag-and-drop them into the code. The most well-known block based programming language, Scratch's official portal [13] we find some advantages and disadvantages, usually mentioned in this topic. David Weintrop examined this problem in details in his dissertation and articles. [14, 15].

He examined a very serious problem that "block-based" programming environments, while successful in changing attitudes and engaging learners, do not adequately prepare them to transition to more conventional programming languages, thus imposing an artificial ceiling on how far learners can progress with these tools [15].

This question is relevant for us, in the sense that we think it is a similar problem that mathematics didactic literature discusses, as the transit from a lower representation stage to a higher.

3.1 Blockly Code as Tool from Mathematical Didactical and Mathematical Psychological Point of View

Because we chose a mathematical teaching content, as an aid for teaching computer science content, we have to investigate what the mathematical didactical and mathematical psychological background teaching problems are, so that we can use and recall them, to support our teaching goals.

3.1.1 Bruner's Representation Stages

In mathematics teaching, we used three stages of representation and we called them, Bruner's representation stages, after Jerome Bruner [5].

In [2] we examined a Blockly Code programming editor, in the context of mathability, now we would like to investigate it in the context of mathematical representation stages.

Enactive or action-based stage means concrete tangible tools, hands-on manipulative methods to understand a certain problem and model (or solve) it. Nowadays, it is a very exciting question, for example, whether a computer software (for example a game) can be evaluated as an action-based tool or not.

Because there is no real physical interaction, just imaginary. Volk and colleagues examined this question [16] in the sense of tablet using in Math lessons. Based on their results, the authors argue for the introduction of tablets in schools, because 'their multi-sensory human-computer touch interaction provides interactive manipulatives supporting transition between representations on the concrete, visual and abstract level' [16].

As we mentioned before, we plan this method for the near future, and we believe that tablets or other computer hardware tools will have very important roles in schools, and not just as a tool for displaying, demonstrating and illustrating problems, but as a tool for solving the problem. Blockly, like other drag-and-drop programming environments, gives the feeling for children, that they are working with hands-on tools. So we can say, Blockly is able to make connection between representation stages. According to [16] we think, Blockly's comfortable handling, the unnecessary memorization of components of language and the illusion of physical manipulation, reassures children and reduces their fear of a difficult task.

Iconic representation is the next level of representation stages. It means, we use structured 'illustrations', i.e. diagrams, tables, etc. for modeling the problem. When we do computer programming, we have to plan for the type of data structures to order our data. It is the part of the implementation. So, what we use at the phase of mathematical solution for illustration, later, at the algorithmization phase, we have to implement it. Good examples are early heuristic problems, for example, in divisibility domain. Usually we order the possible solutions into a table and eliminate those cases that do not lead to a solution. At implementation of such, we use 'loop plus list' or 'double list' methods to run and find every case, and enumerate the good cases into a list. The problem below is from a 4th grade Hungarian Mathematical textbook [17].

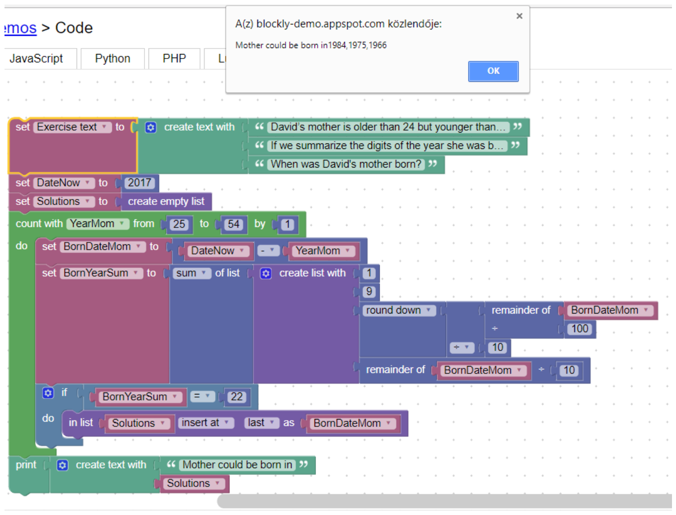
Example: David's mother is older than 24 but younger than 55. If we sum the digits of the year she was born, we get 22. When was David's mother born?

We order the solutions into a table (see Table 1):

Table 1
Cases of David's mother task

YearMom	BornDateMom = = DateNow - YearMom	The sum of the digits of 'BornDateMom'	Solution (Y/N)
25	1992 = 2017 - 25	1+9+9+2 = 21	N
26	1991 = 2017 - 26	1+9+9+1 = 20	N
…	…	…	…
54	1963 = 2017 - 54	1+9+6+3 = 19	N

The loop we have to run for 'YearMom', from 25 to 54, we count the 'BornDateMom', and check with the sum of the digits if it satisfies the initial condition or not. If it does, we put it into the solution list. In Figure 2, we can see the implementation of this task in Blockly Code.



A(z) blockly-demo.appspot.com közlendője:

Mother could be born in1984,1975,1966

```
set Exercise text to create text with “David’s mother is older than 24 but younger than...”
                                      “If we summarize the digits of the year she was b...”
                                      “When was David's mother born?”
set DateNow to 2017
set Solutions to create empty list
count with YearMom from 25 to 54 by 1
do  set BornDateMom to DateNow - YearMom
    set BornYearSum to sum of list create list with 1
                                                    9
                                                    round down remainder of BornDateMom
                                                               ÷ 100
                                                               ÷ 10
                                                    remainder of BornDateMom ÷ 10
    if BornYearSum = 22
    do  in list Solutions insert at last as BornDateMom
print create text with “Mother could be born in”
                       Solutions
```

Figure 2
Blockly Code implementation of ‘David’s mother’ task[1]

If we only examine Blockly Code as a tool, we should mention the block-like arrangement, the coloration of the block groups what reports about the function of the certain blocks. We also have to emphasize the elements that support the possible attachment or mutation of the blocks. All of these are great help to model the problem. But, from the results of the final test, we can see how many children it helps to recognize the certain steps of problem solving.

The highest level of abstraction is the ***symbolic stage*** when we use a formal language. As we mentioned, Blockly Code is like a pseudocode. Its other advantage is that its mathematical toolbar and formalism are similar to the well-known mathematical symbols and concepts we use in Mathematics lessons. We have to mention, that Blockly Code is available in Hungarian. In our method Blockly Code plays the role of the language, and it means that writing confidently a right pseudocode for a problem with Blockly Code means that children can use the language of algorithmization consciously, so they can solve the problem at a symbolic stage, but of course, not immediately, not for the first look. We have to teach it step by step and this is what our method is all about.

1 https://blockly-demo.appspot.com/static/demos/code/index.html?lang=en#izn4tc

3.1.2 Teaching Concepts and Making Them Conscious

We called again for the methodology of Mathematics teaching, when we decided how to introduce and teach computer programming content. In computer science, like in other sciences there are usually descriptive names of concepts. For example, list, array, loop, etc. It gives an image in our mind at once, when we hear the word. But it is not obvious for children. Children's attention should be drawn to the relationship between common meaning of the word and the role of the tool in computer programming. But it is not enough. When we teach, based on mathematical analogies, we have to teach and make these analogies conscious as well, and – as we have mentioned before – the iconic stage can help this process. The example we presented anticipates that it depends on the mathematical problem type. So we have to teach the connection between types of word problems and basic programming elements.

3.1.3 Cognitive Load Theory and Mathematics Teaching

The theory of X, Y, and Z generations has become fashionable psychological theory. It investigates the psychological impact of technical tools and it tries to identify the characteristics of the given generations [18]. With this theory, today's high school children's natural need and important feature, is multitasking, that is, the ability to share attention.

Having this need does not mean that it is good for them. Indeed, if there is too much information that we have to process, our working memory is overloaded. The Working Memory model was first presented by Baddeley and Hitch (1974) and has been refined many times since. Working memory is the territory of our mind where conscious knowledge processing takes place – understanding, realizing, compiling knowledge, comparing, critical thinking, problem solving, planning strategies, using transformation strategies, making analogies, making connections between things, making mental representations and abstractions. This is the place where analogy and metaphorical based thinking takes place [19].

In the model of working memory it has four main components:

- Phonological loop (for storing verbal, and sound information and maintaining them by repeating)
- Visio-spatial sketchpad (for storing and maintaining visual information)
- Episodic buffer (making connection between the verbal and visual information by the supervising of 'central executive' and with the help of the information from the long-term memory)
- Central Executive: it is a supervisory system that controls the flow of information from and to the other subsystems [19]

In Hungarian mathematical didactics Ambrus, investigated and summarized the role of working memory in Mathematics teaching and learning processes. In [19] he makes some didactical suggestions to avoid cognitive overload during teaching mathematical problem solving. Cognitive load means the load of working memory during information processes.

Sweller and colleagues worked out their theory of cognitive overload and mathematical problem solving. They stated that problem solvers have to have many problem situations, problem positions in their mind (similar to professional chess players' mind about chess positions) and the schemas of steps by step solutions (as some kind of strategy). After problem solvers successfully recognize the problem they can recall and activate these schemas [19]. If a student does not possess suitable schemas, he/she has to activate trial and error, or other attempting methods and it depends on luck if these methods can help or not. Biró and Csernoch presented in [20] that although these methods require metacognitive processes they do not develop the algorithmic skills. Furthermore, the other problem is that these methods overload the capacity of working memory very much. Working memory has very small capacity: Miller's law states that the number of objects an average human can hold in working memory is 7 ± 2. [21]

Some new research states that because Miller made no distinction between the type or length of the information his law must be reconsidered, and nowadays research shows that 4 ± 1 units of information are closer to reality. Furthermore, if there is not only storing information but also processing it, this capacity is not more than 2 or 3 units. [19]

We believe that analogy-seeking thinking can be developed for everyone if we consciously pay attention. Furthermore, if our analogy based strategy could be automated, it does not occupy working memory, because automated methods are stored in long-term memory and recalling them uses only one information unit from working memory capacity [19].

3.1.4 Cognitive Load Theory and Computer Science Teaching

According to our research, we have to mention some results in this topic [22, 23, 24]. All of the three publications investigate the problem of cognitive overload when tutorials, exercises, and other materials are planned and created for users. However, tutorials are mainly developed for independent studying, and it is obvious that the question was raised because of the multimedia technology research, we appreciate the objective that the issue of working memory has been raised both in user training and in computer programming teaching and the methodology literature of computer science and information technology has begun to deal with it. But we also have to examine this question in the teaching-learning process, in computer science lessons/classroom work.

3.1.5 Examining Blockly Code in the Sense of Cognitive Overload

There are some ergonomic aspects of cognitive load theory which we can examine. Some block-based programming languages that were made for novice (mainly children) computer programmers are too colorful or contain too many unnecessary graphical elements. This can cause cognitive overload [19]. Further ergonomic inconvenience, can also cause cognitive overload, for example, when we have to scroll down a lot on a web-page, etc. Blockly Code was planned for being the programming library of making further block based languages, so its user interface is simple and free from unnecessary elements.

Drag and drop technique is an ordinary motion in our touch screen based world. Some of the blocks are able to mutate. It means that a block can be expanded if it is necessary. For example, in Blockly Code there are not 3 types of conditional statements (IF...THEN, IF...THEN...ELSE, ELSE IF…). There is one type of IF...DO block and it can mutate, if the problem solving process requires it.

4 Experiment

In [2] we presented our teaching experiment with analogy-based computer programming method and showed results from the student groups. The first groups who studied Blockly were in their graduation year, the time when we decided to measure the pupils who took part in our teaching experiment. So we do not measure the graduating classes, only the younger students. We wanted to know whether they recall some knowledge from our analogy-based method or not. We had 63 pupils in 4 groups with different attitudes to computer programming and at different ages.

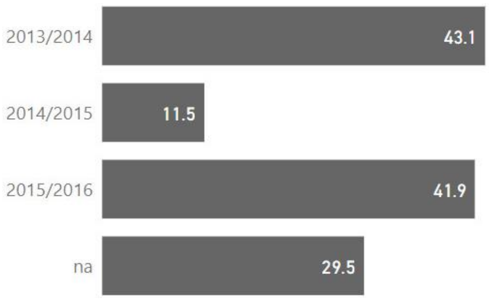


Figure 3

The performance of student groups by school year when they learned algorithmization with Blockly Code

In Figure 3 we can see that after one year (2015/2016), from teaching, the knowledge of algorithmization can easily be recalled. In the 2015/2016 school year, there were two study groups who studied algorithmization with Blockly Code. One was a 6th grade class with scientific orientation, the other was a classic 4th grade class with applied scientific orientation. After two years (2014/2015) the result is much worse, however this class was a six-graded scientific orientation class too. The group studied Blockly Code in 2013/2014 school year and the group who never studied Blockly (signed with 'na') was mainly from a special group in Informatics. These children chose Informatics to take the graduation exam in the 12th grade, so they had two more Informatics lessons in 11th and 12th classes to prepare them. So, despite the time, their motivation was much better than the others.

4.1 The Structure of the Worksheet and the Relationships between the Tasks

In [2] we presented some results from students' work during the teaching experiment. Now, we would like to show the results of a final test we wrote with four groups, who were taught using this method. They were different orientation classes and the time elapsed, since the teaching experiment, is different for each group.

Every class completed the final test in May 2017. The final test was in a printed format, not on computers. So we create tasks from solutions of word problems of different types, or part of algorithms. In different tasks children had to recognize the mathematical concepts involved in the algorithm, they had to recognize analogies of mathematical problem solving, they had to decide whether a solution was right or not and sometimes they had to troubleshoot.

In the test we had 9 tasks that measure how children can recognize the analogy between mathematical tasks, and the mathematical problem solving method in computer algorithms. Task 1-5 are from the questionnaire of TAaAS project (Testing Algorithmic and Application Skills), that measured Hungarian university students' algorithmic skills by Csernoch and colleagues [25]. Their test had two parts. There was a questionnaire, from which the researcher could get information concerning the students' former results (graduation exam, etc.) and the circumstances of their former education in Math and Computer Science. The other part was the test that measured students' computational thinking and algorithmization skills in different (traditional and nontraditional) programming environments and examined whether students think consciously, when they are doing or evaluation an algorithm. They clustered it based on SOLO taxonomy.

We should mention another research from Spain, where Roman-Gonzales and colleagues created and validated their own computational thinking test based on Dr. Scratch [25] and the international Bebras [26] tests. They also extended the nomological network of "Computational Thinking" with some non-cognitive factors in [27], so they moved forward investigating and mapping the psychological aspects of this domain with their research.

Task 1-3 originally in TAaAS project, used simple pseudocodes to present algorithms and students had to tell what the codes did. We implement some of these tasks in Blockly Code environment and ask further questions. Task4 and Task5 were left original, we did not implement them in Blockly. Task4 was illustrated with a diagram; Task5 was implemented in block diagram.

Task1 is a simple algorithm for changing the values of two variables.

Task2 is a simple algorithm to decide about 3 required numbers if they are Pythagorean triples or not.

Task3 is a simple algorithm to decide about 4 required numbers (in a certain order) if they could satisfy the general formula of the linear function (when the first number is the slope of the function, the second is the y-intercept, and the third and fourth are the abscissa and the ordinate of a point).

Task4 was a short pseudocode and a set of 50 numbers. Furthermore, these numbers were described in a diagram. The pseudocode is about counting how many numbers are more than 800.

Task5 was an algorithm in block diagram. There was a list with smiley figures with different size. The block diagram was about a loop that counts the number of the smilies smaller than a defined size.

We created another 4 tasks. We chose 4 word problems solved with Blockly Code by pupils. There were good and also bad solutions. The children had to answer some questions related to the solutions.

Task6 was a good solution of a classic mathematical word problem about the connection of distance, speed and time. We asked pupils to sign the place where they find the steps of classic mathematical solving (Data gathering, Plan, Count, Check, Answer), decide, whether the solution is suitable for generalizing the problem, and what should be changed for that in the code. Furthermore, if they would have to implement, how did they use programming tools for the solution of the problem and for the printing of the result.

Task7 was a troubleshooting exercise. We created a very chaotic algorithm based on a bad solution for a word problem. The students had to recognize the types of 8 mistakes.

Task8 and Task9 were similar to Task6, but we also asked pupils whether the solution was correct or not, because Task8 was a mathematically incorrect solution. It would lead to a quadratic equation, but because of a wrong initial value of the loop in the Blockly implementation (we missed one root). Task9 was a correct solution of a heuristic problem we presented in the details in [2].

4.2 The Aspects of Assessment of Students' Solutions

First we categorized the tasks from four aspects.

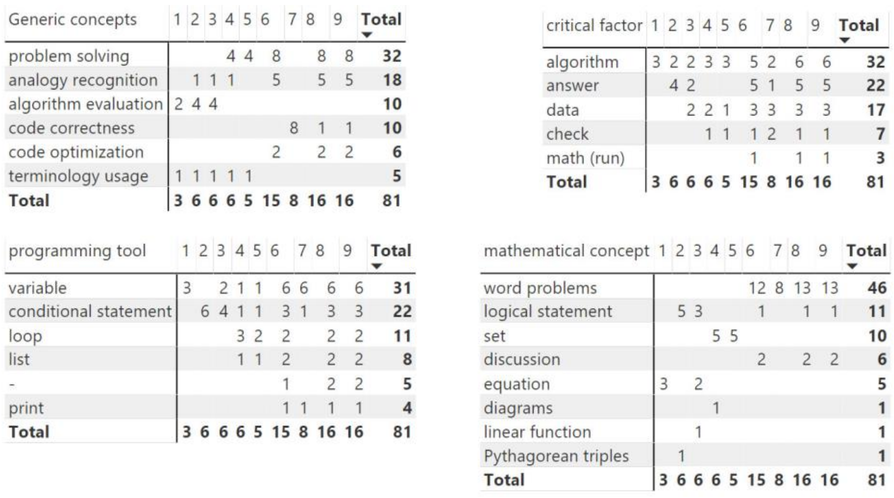
1) The identified generic concepts we wanted to measure with the certain task could be: algorithm evaluation, analogy recognition, code correctness, code optimization, problem solving and terminology usage.

2) The identified critical factors are the elements of problem solving in algorithmization process: data, algorithm, math (run), check and answer.

3) The teaching content in algorithmization: list, loop, variable, conditional statement, printing.

4) Related Mathematics teaching concept: diagrams, discussion, equation, linear function, logical statement, Pythagorean triple, sets, solving word problems.

We have 9 Task, with 81 subtasks. In Figure 4 we can see how the certain tasks were built by the aspects above:



Generic concepts	1	2	3	4	5	6	7	8	9	Total
problem solving				4	4	8		8	8	**32**
analogy recognition		1	1	1		5		5	5	**18**
algorithm evaluation	2	4	4							**10**
code correctness							8	1	1	**10**
code optimization						2		2	2	**6**
terminology usage	1	1	1	1	1					**5**
Total	**3**	**6**	**6**	**6**	**5**	**15**	**8**	**16**	**16**	**81**

critical factor	1	2	3	4	5	6	7	8	9	Total
algorithm	3	2	2	3	3	5	2	6	6	**32**
answer		4	2			5	1	5	5	**22**
data			2	2	1	3	3	3	3	**17**
check				1	1	1	2	1	1	**7**
math (run)						1		1	1	**3**
Total	**3**	**6**	**6**	**6**	**5**	**15**	**8**	**16**	**16**	**81**

programming tool	1	2	3	4	5	6	7	8	9	Total
variable	3		2	1	1	6	6	6	6	**31**
conditional statement		6	4	1	1	3	1	3	3	**22**
loop				3	2	2		2	2	**11**
list				1	1	2		2	2	**8**
-						1		2	2	**5**
print						1	1	1	1	**4**
Total	**3**	**6**	**6**	**6**	**5**	**15**	**8**	**16**	**16**	**81**

mathematical concept	1	2	3	4	5	6	7	8	9	Total
word problems						12	8	13	13	**46**
logical statement		5	3			1		1	1	**11**
set				5	5					**10**
discussion						2		2	2	**6**
equation	3		2							**5**
diagrams				1						**1**
linear function			1							**1**
Pythagorean triples		1								**1**
Total	**3**	**6**	**6**	**6**	**5**	**15**	**8**	**16**	**16**	**81**

Figure 4
Distribution of the final results by the aspects of assessment

During the evaluation of solutions we categorized the mistakes depending on whether a certain mistake was algorithmically, mathematical or terminological (and of course we also made a difference, when there was no solution or no mistake).

We marked the solutions from 0 to 1, depending on the aspects above.

From our pupils, we know their grade, orientation, gender, whether they learned algorithmization with our method and Blockly environment or not, and the elapsed time from learning algorithmization. Furthermore, students can express their opinion about the tasks by a 5 graded Likert scale where: 1 - task was unknown problem for the student and does not know what to do with it; 2 - task causes a lot of difficulties, but student tried to solve although they were not sure about whether their solution was correct or not; 3 - task was difficult, but student solved it successfully; 4 - student had to think, but soon realized the solution; 5 - task was very easy. Really they assess their knowledge, and furthermore, this self-assessment we can use as the subjective measuring of cognitive load [19].

We organized our data into a dataset. We process our research data with OLAP technology for better visualization and further extensibility. Our dataset is a multidimensional data cube[2]. The indicators (facts) of the cube are the point of the solution and the self-assessment point. Task, Solution and Pupils are formed in Tables (dimensions) of the data cube with the attributes we can see in Figure 5.

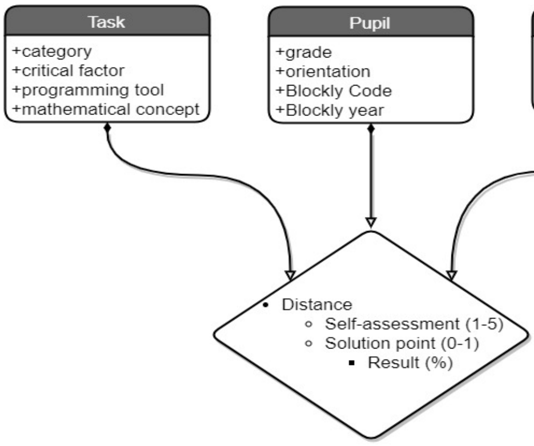
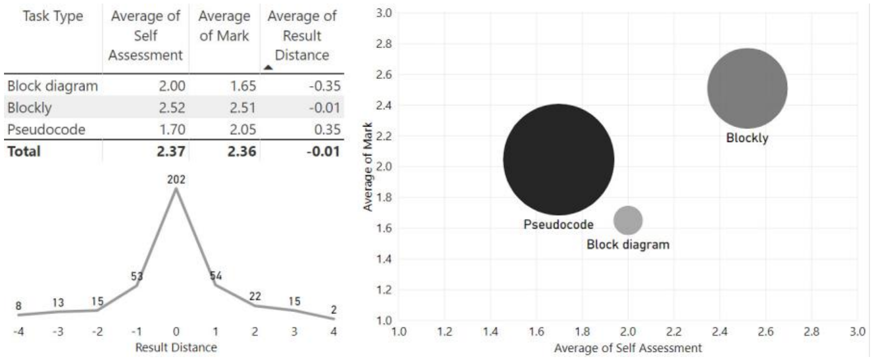


Figure 5
Structure of the OLAP-cube

4.3 Results

First we present the average results related to the Blockly Code, as programming environment.

[2] The whole dataset and visualization can be seen at https://goo.gl/tHKpkS



Task Type	Average of Self Assessment	Average of Mark	Average of Result Distance
Block diagram	2.00	1.65	-0.35
Blockly	2.52	2.51	-0.01
Pseudocode	1.70	2.05	0.35
Total	**2.37**	**2.36**	**-0.01**

Figure 6
Blockly vs. pseudocode or block diagram

In Figure 6 we compared the achieved average points and the self-assessment points related to the tasks that were implemented in Blockly Code or in another algorithm modeling tool (pseudocode or block diagram).

First, we must point out that the children's self-assessment shows a normal distribution, undervaluation or overvaluation of their own performance is not typical in the examined group.

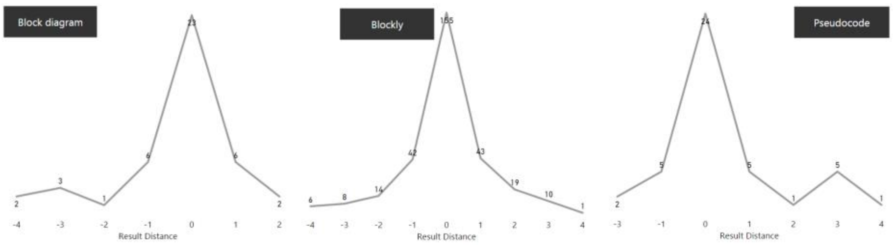
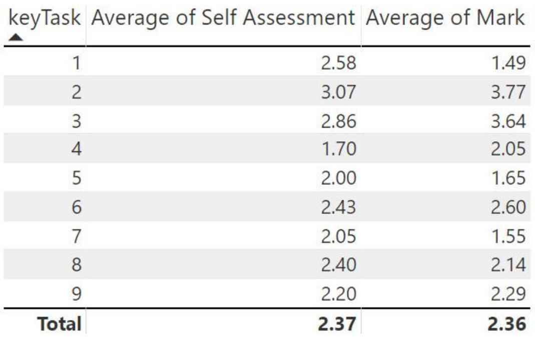


Figure 7
Students' self-assessment in different environments

From results in Figure 7, we can see that the best result was in the tasks where the implementing tool was Blockly Code, furthermore pupils could rate their own performance, most real, on these tasks. Children could experience block diagraming, in their former (primary school) studies and we think this was the reason they overvalued their own performance in this tasks. However, they had not seen pseudocode before, as we have already told Blockly Code actually is a user-friendly pseudocode, so we think it is the reason why they rate themselves under their real performance.

The next result we present are the average results by tasks and in total in Figure 8.

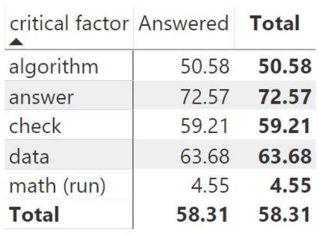


keyTask	Average of Self Assessment	Average of Mark
1	2.58	1.49
2	3.07	3.77
3	2.86	3.64
4	1.70	2.05
5	2.00	1.65
6	2.43	2.60
7	2.05	1.55
8	2.40	2.14
9	2.20	2.29
Total	**2.37**	**2.36**

Figure 8
Average results by tasks and total

Students could tell their opinion about the tasks by the 5 graded Likert scale. A self-assessment can be used as the subjective measuring of cognitive load. In Figure 8 we can see that students thought these tasks sometimes were significantly more difficult than their performance showed. The most difficult problem was Task4 for the students. It was the pseudocode illustrated with a diagram. As we mentioned previously – and also can be seen in Figure5 – they were more successful than they thought. The less cognitive load they had at Task2 and Task3. These tasks were algorithm evaluation based on counting with concrete numbers, furthermore related to mathematical concepts and after pupils have noticed these mathematical concepts they can easily make success with the tasks.

The next diagram we present concerns the distribution of the identified critical factors, based on the 63 students' work. We can see that our analogy based method works well, except we have to pay more attention in the future, to eliminate/avoid some mistakes.



critical factor	Answered	Total
algorithm	50.58	**50.58**
answer	72.57	**72.57**
check	59.21	**59.21**
data	63.68	**63.68**
math (run)	4.55	**4.55**
Total	**58.31**	**58.31**

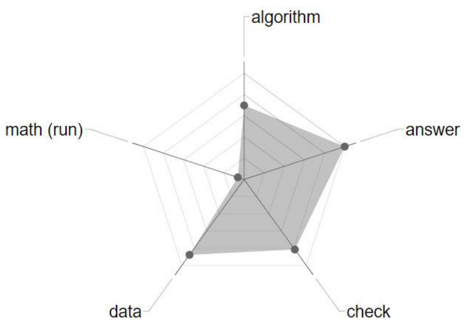
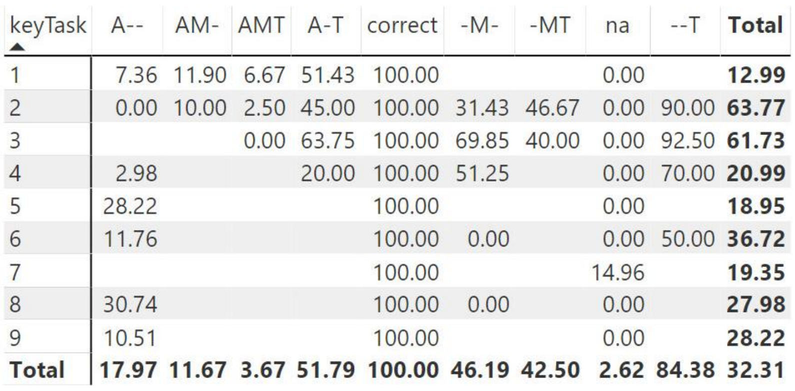


Figure 9
Average results of critical factors

Figure 9 shows how successful the analogous steps of the mathematical problem solving were discovered during computer problem solving. The most obvious and recognized analogy, by children, were data gathering and answering. We feel, we get an acceptable result at Checking as well, but the result of the algorithm needs to be improved. This result shows that the implementation of the mathematical plan did not cause any problems only in the quarter of the total solutions, however we have to note that in this result there are those who did not respond at all. To eliminate them, we created the column in the middle, to see only those results who started to solve the task. The other problem is Math (run). In the teaching process we emphasized many times – but not enough – that there is no analogous step we made with counting. Counting means that computer runs the program and we do not count. Nevertheless, it was a common mistake to consider the step of 'counting' in the code, mostly at the expense of the 'plan' or 'check' stages.

Finally, we present the results, categorizing the types of mistakes.



keyTask	A--	AM-	AMT	A-T	correct	-M-	-MT	na	--T	**Total**
1	7.36	11.90	6.67	51.43	100.00			0.00		**12.99**
2	0.00	10.00	2.50	45.00	100.00	31.43	46.67	0.00	90.00	**63.77**
3			0.00	63.75	100.00	69.85	40.00	0.00	92.50	**61.73**
4	2.98			20.00	100.00	51.25		0.00	70.00	**20.99**
5	28.22				100.00			0.00		**18.95**
6	11.76				100.00	0.00		0.00	50.00	**36.72**
7					100.00			14.96		**19.35**
8	30.74				100.00	0.00		0.00		**27.98**
9	10.51				100.00			0.00		**28.22**
Total	**17.97**	**11.67**	**3.67**	**51.79**	**100.00**	**46.19**	**42.50**	**2.62**	**84.38**	**32.31**

Figure 10
Results by mistakes

We categorized students' answer whether there was an algorithmically, mathematical or terminological mistake (nothing or none were answered). In Figure 10 we can see the type of mistakes they made, by tasks. Our experience is that although students were successful in solving Task2 and Task3, at the same time, they had many faults, inaccuracies in their terminology usage, both in mathematics and in informatics. So we should pay much more attention when teaching the correct use of terminology and we have to make sure that they use the language correctly.

Conclusions

Summarizing, when using Blockly Code and our analogy-based introductory computer programming teaching method, we successfully taught and automatized a problem-solving strategy, that pupils could store in their long-term memory and they did not overload their working memory during coding. They could focus on the problem they had to solve and not on the language or other difficulties of implementation. We feel, we have provided answers to our opening questions herein. Analogies are helpful in teaching problem solving methodologies.

Children can recognize mathematical analogies in the Blockly-based environment and this approach can help overcome the aversion to traditional computer programming and strengthens the relationship between computer science and mathematics.

Based on our experience we think this analogy-based method could be an effective way (both in content and time) of teaching mathematical problem solving – and during the generalization – the basic elements of future, more complex computer programming.

During the evaluation of the students' work we identified some typical mistakes and we think, this could be the next step of our research when we explore potential misconceptions, common bad terminology and what we then what we need to pay more attention to during the teaching process.

We believe, it should be introduced into Mathematics curriculums by our Mathematics teachers to lay the foundation for algorithmization.

References

[1] K. Takácsné Bubnó and V. Takács: Solving word problems by computer programming. Proceedings of Problem Solving in Mathematics Education 2013 Conference (ProMath), A. Ambrus and É. Vásárhelyi, (eds.), Budapest, 2014, pp. 193-208

[2] K. Bubnó and V. Takács: The mathability of word problems as initial computer programming exercises. Proceedings of the 8th IEEE International Conference on Cognitive Infocommunications (CoginfoCom) Debrecen, 2017, pp. 39-44

[3] P. Baranyi and A. Gilányi, Mathability: Emulating and enhancing human mathematical capabilities, Proceedings of the 2013 IEEE 4th International Conference on Cognitive Infocommunications (CogInfoCom) Budapest, 2013, pp. 555-558

[4] G. Pólya: How to solve it, 2nd ed., Doubleday, 1957

[5] A. Ambrus, Introduction to the didactics of mathematics, 2nd ed., ELTE, 2004 (In Hungarian)

[6] B. Szi and A. Csapo, An outline of some human factors contributing to mathability research, Proceedings of the 2014 5th IEEE Conference on Cognitive Infocommunications (CogInfoCom) Vietri sul Mare, 2014, pp. 583-586

[7] K. Chmielewska, A. Gilányi and A. Łukasiewicz, Mathability and mathematical cognition, Proceedings of the 2016 7th IEEE International Conference on Cognitive Infocommunications (CogInfoCom) Wroclaw, 2016, pp. 245-250

[8] K. Chmielewska and A. Gilányi, Mathability and computer aided mathematical education, 2015 6th IEEE International Conference on Cognitive Infocommunications (CogInfoCom) Győr: IEEE 2015, pp. 473-477

[9] P. Biró and M. Csernoch, The mathability of spreadsheet tools, 2015 Proceedings of the 6th IEEE International Conference on Cognitive Infocommunications (CogInfoCom) Győr, 2015, pp. 105-110

[10] B. Szi and A. Csapo, An outline of some human factors contributing to mathability research, 2014 5th IEEE Conference on Cognitive Infocommunications (CogInfoCom) Vietri sul Mare, 2014, pp. 583-586

[11] M. Török, M. J. Tóth and A. Szöllősi, Foundations and perspectives of mathability in relation to the CogInfoCom domain, 2013 IEEE 4th International Conference on Cognitive Infocommunications (CogInfoCom) Budapest, 2013, pp. 869-872

[12] Google: Blockly Code, https://developers.google.com/blockly/

[13] MIT: Scratch, https://scratch.mit.edu/

[14] D. Weintrop and U. Wilensky: Comparing block-based and text-based programming in high school Computer Science classrooms. ACM Trans. Comput. Educ. 18, Article 3

[15] D. Weintrop: Modality matters: understanding the effects of programming language representation in high school Computer Science classrooms (PhD dissertation) Evanston, 2016

[16] M. Volk, M. Cotič, M. Zajc, A. Istenic Starcic: Tablet-based cross-curricular maths vs. traditional maths classroom practice for higher-order learning outcomes, Comput. Educ. 114, 2017, pp. 1-23

[17] K. Némethné Rakos: My Mathematics 4, Celldömölk, 2012 (In Hungarian)

[18] A. Tari: Z generation, Budapest, Tericum, 2011 (In Hungarian)

[19] A. Ambrus: Some cognitive psychological issues in mathematics education, Gradus 2, 2015, pp. 63-73 (In Hungarian)

[20] P. Biró and M. Csernoch, Deep and surface structural metacognitive abilities of the first year students of Informatics, 2013 IEEE 4th International Conference on Cognitive Infocommunications (CogInfoCom) Budapest, 2013, pp. 521-526

[21] G. A. Miller: The magical number seven plus or minus two: some limits on our capacity for processing information. Psychological Review 63, 1956, pp. 81-97

[22] J. Á. Velázquez-Iturbide, Exploring the joint use of educational theories and information technology to improve CS courses, Proceedings of the

2017 IEEE Global Engineering Education Conference (EDUCON) Athens, 2017, pp. 1561-1570

[23] S. Feinberg and M. Murphy, Applying cognitive load theory to the design of Web-based instruction, IPCC/SIGDOC '00 Proceedings of IEEE professional communication society international professional communication conference and Proceedings of the 18th annual ACM international conference on Computer documentation: technology & teamwork, Cambridge, MA, 2000, pp. 353-360

[24] K. J. Harms, Applying cognitive load theory to generate effective programming tutorials, 2013 IEEE Symposium on Visual Languages and Human Centric Computing, San Jose, CA, 2013, pp. 179-180

[25] M. Csernoch, P. Biró, J. Máth and K. Abari, Testing algorithmic skills in traditional and non-traditional programming environments, Inf. Educ. 14, 2015, pp. 175-197

[25] Dr. Scratch, http://www.drscratch.org/

[26] Bebras, http://www.bebras.org/

[27] M. Román-González, J.-C. Pérez-González, J. Moreno-León and G. Robles, Extending the nomological network of computational thinking with non-cognitive factors, Comput. Human. Behav. 80, 2018, pp. 441-489

[28] A. Kővári, CogInfoCom Supported Education: A review of CogInfoCom based conference papers, 2018 IEEE 9th International Conference on Cognitive Infocommunications (CogInfoCom) Budapest, 2018, pp. 233-236

[29] V. Kövecses-Gősi, Cooperative learning in VR environment, Acta Polytech. Hung. 15, 2018, pp. 205-224

[30] B. Lampert, A. Pongracz, J. Sipos, A. Vehrer and I. Horvath, MaxWhere VR-learning improves effectiveness over clasiccal tools of e-learning. Acta Polytech. Hung. 15, 2018, pp. 125-147

[31] I. Horvath and A. Sudar, Factors Contributing to the Enhanced Performance of the MaxWhere 3D VR Platform in the Distribution of Digital Information. Acta Polytech. Hung. 15, 2018, pp. 149-173

[32] G. Csapó, Sprego virtual collaboration space, 2017 IEEE 8th International Conference on Cognitive Infocommunications (CogInfoCom) Debrecen, 2017, pp. 137-142

[33] A. D. Di Sarno, S. Dell'Orco, R. Sperandeo, G. Iorio, N. M. Maldonato, M. L. Fusco, V. Cioffi, E. Moretto, T. Longobardi and G. Buonocore, Conscious experience using the Virtual Reality: A proposal of study about connection between memory and conscience, 2018 IEEE 9th International Conference on Cognitive Infocommunications (CogInfoCom) Budapest, 2018, pp. 289-292

[34] L. L Mosca, A. D. Di Sarno, R. Sperandeo, V. Cioffi, N. M. Maldonato, M. Duval, S. Dell'orco, G. di Ronza and E. Moretto, "I am a brain, Watson. The rest of me is a mere appendix" The memory, a characteristic of the human being, 2018 IEEE 9th International Conference on Cognitive Infocommunications (CogInfoCom) Budapest, 2018, pp. 254-298

[35] B. Berki, Better Memory Performance for Images in MaxWhere 3D VR Space than in Website, 2018 IEEE 9th International Conference on Cognitive Infocommunications (CogInfoCom) Budapest, 2018, pp. 281-283

[36] A. Csapo, I. Horváth, P. Galambos and P. Baranyi, VR as a Medium of Communication: from Memory Palaces to Comprehensive Memory Management, 2018 IEEE 9th International Conference on Cognitive Infocommunications (CogInfoCom) Budapest, 2018, pp. 389-394

[37] P. Bőczén-Rumbach, Industry-Oriented Enhancement of Information Management Systems at AUDI Hungaria using MaxWhere's 3D Digital Environments, 2018 IEEE 9th International Conference on Cognitive Infocommunications (CogInfoCom) Budapest, 2018, pp. 417-422

[38] Z. T Horváth, Another e-learning method in upper primary school: 3D spaces, 2018 IEEE 9th International Conference on Cognitive Infocommunications (CogInfoCom) Budapest, 2018, pp. 405-408