The mathability of word problems as initial computer programming exercises

Katalin Bubnó
Doctoral School of Mathematical and Computational Sciences,
University and National Library,
University of Debrecen,
Debrecen, Hungary
kbubno@lib.unideb.hu

Viktor László Takács
Faculty of Business and Economics,
Lajos Kossuth Teacher Training School,
University of Debrecen,
Debrecen, Hungary
takacs.viktor@econ.unideb.hu

***Abstract*—In this paper we present our research and teaching methods related to solving word problems by computer programming. We have experience in secondary school projects in computer science lessons. In this paper we present the basic idea of the method, as well as some of the results of our experience. Furthermore we examine our method from the point of view of mathability.**

Keywords—teaching novice computer programming; Blockly; word problems; mathability

I. INTRODUCTION

At every level of mathematical studies we encounter word problems, and children have many problems with these types of exercises at every level. Hungarian research teams in the 2000s examined children of different ages many times using international tests in computer environments to show Hungarian children's mathematical difficulties and the similar difficulties faced by children of other nations. Summarizing these results Csapó et al. [1] laid the foundations of online mathematical measurements, and we found that this is a good starting point for our information technology methodology also. So, most of the problems do not depend on language, but on the way of thinking used. Csapó et al. [1], like many other researchers [2,3,10,11,18] from Pólya [2] to Skemp [3], declares the sad fact that children cannot use their mathematical knowledge well in solving real-life problems.

The very first problem is that children find it difficult to recognize the applicability of their studies in real life. In Hungarian textbooks [4] we can see great attempts to change this with exercises more closely based on reality, and to improve the second problem, which is the difficulty of modelling and translating daily life problems into the language of mathematics.

Considering the problem further, we have found that if children cannot apply maths in everyday life, then they probably also do not appreciate information technology as a tool for solving everyday problems. Unfortunately, Hungarian students' performances on the PISA2012 and 2015 surveys in comprehension, mathematical and digital competences all proved this. [5]

We present a method which can develop all the above competencies at the same time. We started an experiment to teach computer programming by first using word problems. In [6] we reported our first pilot studies on this topic and published our first impressions.

In [6] we show that there are certain analogies between classical mathematical problem solving methods and the steps of thinking when implementing a computer program for a particular problem. In Hungary, we use George Pólya's problem solving model [2] at every level of mathematical teaching, so it was natural for us to implement it for computer programming.

II. PÓLYA'S STEPS

A. Comprehending and understanding the problem

Comprehending and understanding the problem are common first requirements to solve problems both in mathematics and computer science.

B. Devising a plan

Devising a plan means firstly determining known and unknown variables in mathematics - its equivalent in programming is declaring variables - and secondly creating a mathematical model to solve the problem, which is equivalent to finding the right algorithm and implementing it in the chosen programming language.

C. Carrying out the plan

Carrying out the plan means mainly solving equations mathematically, answering the question in the exercise, and compiling and running the program in computer science, even if this is done by the machine and not by humans (we make no distinction between human and machine work, just as we make no distinction between using paper and pencil or only the mind when doing a counting task). Here we must develop the ability to answer the problem with our program, as well.

D. Discussion

Discussion (Looking back) is the most exciting part of problem solving and can lead to some kind of generalization of the problem; with the help of this process we can create formulas appropriate for a group of similar problems. When we change initial conditions, we can say we test our program, and debug it, when we draw up the problem in general terms, and perhaps, by looking for other solution methods (a better algorithm) we optimize our code.

III. PÓLYA'S STEPS IN COMPUTING EDUCATION LITERATURE

In the educational and psychological literature we can, of course, find similar aspects to those in our research. Our method can fit into analogy-based pedagogical research which focuses on problem solving in computer science. In [7,9] we can read an excellent summary of deep and surface metacognitive approaches to problem solving methods in computer science, mostly based on Pólya's model [2], and the similarity between the steps of Pólya's model and the master levels of the ACM&IEEE Computing Curricula [8, 12].

We want to build these steps into the teaching-learning process recognizably, how we teach novice computer programming with the help of classic mathematical problem solving teaching methodology, as we teach it in elementary mathematics lessons in Hungary (Data, Plan (mathematical model), Implementation (counting), and Checking).

Many researchers, like [10] and [11], are faced with the same problem as university lecturers looking at or surveying [12] the abilities of incoming students at the age of 18-20. The methods they offer for skills development are mostly focused on this age group, too, a group already mature enough to recognize their own shortcomings and motivated enough to correct them.

However, it is obvious, that the problem started much earlier, and must be treated much earlier, somewhere at the point at which children first meet with word problems in primary schools.

In Hungary, before the new National Curricula, we started to teach computer programming at the age of 10 with LOGO, mostly using graphical or geometrical problems. LOGO was a great tool at the time Seymour Papert invented it [13], but today children have great expectations of modern computational tools and LOGO is not attractive enough for them. We think controlling skills can be developed more effectively with games based on block language projects, like those we can find on blocklygames.com and code.org, or our favourite starting controlling game, called Lightbot.

Geometric, fractal and other mathematical or computer graphical exercises that can be solved with LOGO are more suitable for students on university mathematics courses. LOGO would be a great tool, as we can read in [14]. Students at universities already study and understand the mathematics behind LOGO exercises, and it can be a recreational activity and, at the same time, useful practical knowledge for students.

IV. MATHABILITY

In 2013 CoginfoCom a new topic was born for making contact between cognitive infocommunication science and computer science education, when we 'model and understand mathematical capabilities of co-evolving human-ICT systems'. This new area is called mathability [20]. In former publications in mathability subject from CoginfoCom conferences we can see three approaches to use mathability for assessment [12, 21-22, 28-30]. First, when IT tools were evaluated from the point of view of mathability (we distinguish high and low-level mathability tools) [21, 22, 30]. Second, when evaluation focused also on the human mathematical abilities and mathematical intelligence [22, 29], and the third approach, when problem solving methods were evaluated from the point of view of mathability [12]. In [21] authors described the results of their survey in high schools in Poland, and discussed using mathability tools in ICT aided mathematics (science) teaching.

All the publications mentioned above all should be a valuable base for math aided algorithmization (computer programming) teaching, which is our main research. With our paper we attempt to interpret the concept of mathability behind our algorithmization teaching experiment in public education.

Further valuable applications in mathability domain we find mainly from higher education environments in mathematics seminar courses [31-33], and we also find introductory computer science course experiments [12, 28].

We feel our method successful, because the chosen computer programming tool and the problem solving method are supporting each other effectively, so we also exam the tool and the method in the sense of mathability.

V. BLOCKLY

In the summer of 2011 we encountered a new, graphical programming library called Blockly Code, created by Google Inc. This is a programming library for building visual programming editors [15], which means we can create our own programming languages with it. We have achieved some success with it, and we also use it in educational programs. [16][17][24]

We used Blockly Code as a web-based programming environment. This means that users do not have to install any kind of program on their computers, just a browser supporting Google services, and we can use it easily. The design and the simplicity of this programming environment was so arresting that we felt it could be perfect as a first language for teaching computer science to those pupils who are not specially gifted in computer science, or do not consider themselves to be talented in computer programming.

The language was also the problem. Can we expect children to learn to code in a foreign language? We decided we cannot, so we joined the call of Google Inc. to help translate Blockly into as many languages as possible, and helped the Google group with the Hungarian translation.

Obviously, Blockly, as a programming library (or the visual editor which allows creating our own visual programming languages) is a high-level mathability tool.

Several block languages were developed in the past 5 years, based on Blockly, and they could not have been created if Blockly had not been a high-level mathability tool. Several education program in STEM education are based on these block languages and run successfully.

If we examine what kind of basic math blocks we can find in Blockly, first it seems that its math toolbar is similar to a knowledge of a student in public education. This is why Blockly is obviously good for children. Mathematical signs are evident and well-known. However, the building of formulas already requires "two-variable approach", we think this is not useless, indeed! Each mathematical basic operation block can host at most two variables. So children have to use these blocks as 'paranthesises', and they have to use them with caution. In [6] we mentioned it was bothering for primary school children. But in secondary school we found that children can soon acclimate.

Furthermore, in Blockly, there are some mathematical algorithms for examining attributions of numbers (ie. whether a number is prime or not), but they have to code (or write their own procedures that can be reused in codes), if they want to examine perfect squares.

VI. Conscious problem solving with computer programming

A pleasing computer programming environment could be full of motivation in teaching, but it is not enough as a pedagogical goal. We think teaching computer science should start at the point where mathematics finishes: conscious problem solving.

The critical point in creating qualified computer programs is the mathematical discussion of the problem. Certain mathematical problems we learn to solve in math lessons. But there are only a few pupils who can discuss a problem in detail. It is not an expectation in mathematics that all pupils should be able to generalize the problem, formulate it and solve it in general terms. However, if they do not do this, they will not be able to create computer programs for a certain type of problem in general. We think this is the main reason behind the failure to teach computer programming: the limited ability to generalize in mathematics. Computer science teachers expect too much from pupils when they wish them to solve the general problem first and work with formulas and build them into the algorithms at skill level.

According to Piagét's theory of cognitive development and the adaptation theories in mathematical education [18], children typically pass through the concrete operational stage at the age of 7 to 11. They can solve problems in a logical way, but they are not able to think abstractly or hypothetically. So we suggest solving word problems, but only certain problems, at these ages.

Children from 11 years can already do more, but most of them cannot do it without help. Working with and, above all, creating formulas is not easy [19]. We have to teach them the technique involved in this process first.

So after solving problems mathematically, we solve word problems with computer programming (specifically, we use Blockly Code, as the language to achieve this), and continue the detailed discussion of the exercise toward the fully generalized problem in a conscious way. We think this type of teaching of computer programming gives more positive results than previous methods. In the 1990s, there was a similar experiment conducted in Hungarian computer science education [27], but unfortunately the method was not used widely and there was no continuation.

There is one important thing we have to make children conscious of: the reason why it is useful. We have to explain to children that using computers is not effective if we can solve just one particular problem. We have to endeavour to use them more efficiently, and that is why we want our software to be able to solve as many similar problems as possible.

VII. How to solve it — step by step discussion

Step by step discussion is about how we teach how to design, formalize and generalize in algorithms. We present the analogy-based solution method and discussion levels with an example from students' work.

Task1: On a camping site there are only 4-bed wooden houses. 32 students and 2 teachers want accommodation in this camping site. What is the minimum number of houses we must reserve for them, if we adhere to two rules: the first is that students and teachers must not stay together, and the second is that persons of a different sex are not allowed to stay together?

A. Comprehending and understanding the problem

What we are looking for (the unknown) is the number of wooden houses. The number of teachers is 2, and the number of students is 32, but we do not have exact information about their sex (so it is no help that 32 can be divided by 4). This means that for us, there could be more than one case we have to examine.

B. Devising a plan

We must declare the minimal number of wooden houses as a variable, and we should create a formula to count it. The numbers of teachers and students are constants (we declare them the same way in Blockly as variables). The numbers of girls and boys are unknown, but we have to declare only one of them as an unknown variable - the other depends on the former. The situation is similar in the case of male and female teachers. (NBoy; NMteacher; NGirl=32 – NBoy; NFteacher=2 – NMteacher). NMinHouses = ROUNDUP(NBoy/4) + ROUNDUP(NGirl/4) + ROUNDUP(NMteacher/4) + ROUNDUP(NFteacher/4)

More than one solution means that we have to order them into a list data structure. Furthermore, considering that we work within an interval (with natural numbers) this means that we will have to use loops. To see exactly how many lists we have to apply and why, it is helpful to sort the cases in a table for children.

TABLE I. CASES IN A TABLE

NBoy	NGirl	NMteacher	NFteacher	NMinHouses
0	32	0	2	0+8+0+1=9
0	32	1	1	0+8+1+1=10
0	32	2	0	0+8+1+0=9
1	31	0	2	1+8+0+1=10
1	31	1	1	1+8+1+1=11
1	31	2	0	1+8+1+0=9
…				
32	0	0	2	8+0+0+1=9
32	0	1	1	8+0+1+1=10
32	0	2	0	8+0+1+0=9

For the right solution we just choose the maximum value of NMinHouses; we can certainly accommodate them all, in any kind of combination in which they arrive. We have to use two nested loops; the outer loop is determined by the number of boys and by the number of male teachers inside.

C. *Carrying out the plan*

Here, this means compiling and running the program, but first, we have to prove the program can provide an answer. By compiling and running we can detect the mathematical mistakes or misconceptions regarding the solution. When pupils solve mathematical word problems in the traditional way (on paper), they often see what they want to see as a result, even if the solution does not contain it. The computer will confront children the results of their implementation (with all of the mistakes or deficiencies of the solution), and not childrens' expectations.

D. *Discussion*

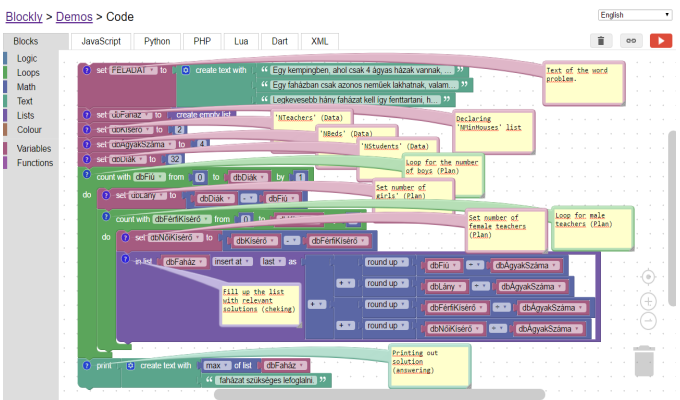
Csíkos in [25] says 'In order to evaluate the proof of other subject areas on the basis of mathematics, a testing context and a testing method are needed to reveal the highest level proof on mathematics based categorization between the many possible proofs that can be given by the learner.'

We adapt this thought in the teaching and also the evaluation of students' works. We teach based on the level of discussion of the certain problem and the evaluation of the computer implementation of discussion.

We have established four different categories in valuable solutions. (Not acceptable solutions we mentioned as level 0.)

The first level means acceptable solution, but there is no discussion. The program is not suitable for solving the problem generated, only the particular task. Variables are declared with their concrete values and the algorithm works with concrete data in the body of the program. The program also provides an answer only to the particular task.

Fig. 1. Solution and explanation of 'wooden house problem'



The second level is the formalization of the problem. Every piece of data is gathered and declared at the head of the program; for example, NBoy; NMteacher; NGirl=AllStudent – NBoy; NFteacher=AllTeacher – NMteacher; NHouses = ROUNDUP(NBoy/4) + ROUNDUP(NGirl/4) + ROUNDUP(NMteacher/4) + ROUNDUP(NFteacher/4). Variables are declared with their concrete values for the particular problem, but we can change it manually in the program head if we want to. The algorithm works with the name of the variables (formalized data). If we discuss the problem by changing the initial data the program can answer the modified problem. In Figure1, we can see a student's solution of this task. The solution is at the 2nd level. (It is in Hungarian, so we explain it graphically in English. The solution link can be seen [23].)

The third level of discussion is a program's ability to communicate interactively. This is the case when our program is able to ask for input data. There are no variables declared with concrete constants; the program will ask for the input data. In the body of the program the algorithm works with the names of the variables. Discussing the problem means that the program works with actual incoming data and answers the particular problem which is generated with actual data.

The highest, fourth level of discussion, we refer to as the 'examination of the quality of input data'. From the mathematical point of view, we get children to build into their program not only the initial data but data arising during the mathematical problem solving as conditions, and data which come from the reality of the particular text problem. Furthermore, pupils have to ensure that the program is able to examine the input data from the perspective of this system of conditions. From a software engineering point of view this leads to software quality questions. In our example above, we work with natural numbers. Pupils have to build the examination of these conditions which raises the quality of the software. In previous so-called typed computer programming languages (for example in Pascal) which we taught, this kind of examination occurred during the declaration of variables. Blockly is not a typed (variable) language, the 'Mathematical Blocks' group contains simple numbers. Students must not declare the type of input data and the expected result. So, while with typed languages we have to pay attention to type of the number at the beginning of the problem solving process, in Blockly we should apply this in the final checking and discussion of the problem solving, which brings it nearer to Pólya's classical mathematical problem solving model [2].

During evaluation we specified 5 factors of assessment: initial data extraction from world problem (Data), problem solving correctness mathematically (Math), problem solving correctness algorithmically (Algorithm), algorithm checking (Check), answering the problem (Answer). We refined the evaluation of the algorithm with two major structures, e.g. list and loop.

We can say that with block languages computer programming is much more similar to mathematical problem solving than in other traditional programming languages. Furthermore, we can say that discussion of word problems as novice computer science exercises are suitable tasks in the

sense of mathability. With this method we successfully create an algorithm for problem generalization, and we think it could be used to model human thinking.

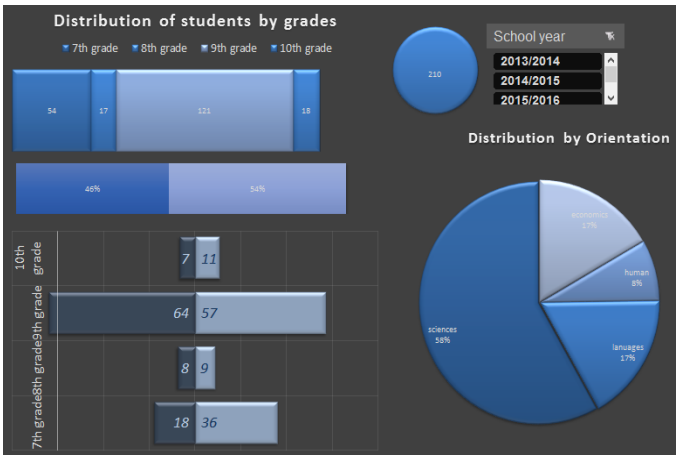
VIII. Experiences

We have been working with the method for four school years in the Secondary School Unit of the Lajos Kossuth Teachers' Training School of the University of Debrecen.

In this school every year there are two six-grade classes, one is with arts and humanities and the other is with science orientation, depending on children's former performance at written entrance exam from mathematics and Hungarian language. Both classes start their studies in secondary school in the 7th class and leave the school in the 12th class. Every year there is a language orientation class from 9th to 13th grades. In the first semester of these classes, children have lessons in two different foreign languages with high numbers of lessons. In mathematics and Hungarian language they have only a few lessons for maintenance of their knowledge, and they have 3.5 lessons per week in Informatics. Beside 5 sport lessons (mandatory for each grades in Hungarian schools) they have no other subjects in the first year. In the next two years, they have 1 lesson per week in Informatics. Because of the 'lost' first year, they leave the school one year later (at the end of 13th class). These classes are the one and only training type classes, who study Informatics in block lessons in their secondary school years. There are two more classes in every grade, and these classes are 'normal' four-grade training classes from 9th-12th, but the one is scientific (engineering or medicine) and the other is social scientific (economic or law) speciality. Beside the mandatory writing entrance exam in Hungarian language and Mathematics, the entrance oral exams are in Biology or Physics for scientific orientation, and in History for social scientific orientation. All of the children from any kind of orientation classes have the opportunity to choose Informatics to learn in 11th-12th grades in two lessons per weeks. This is usually chosen by those pupils who want to take final graduate exam in Informatics.

We had study groups from 7th to 10th grades, from the ages of 13 to 16 from different orientation classes. In Figure 2 we can see the descriptive statistical features of our study groups: number of children and the distribution by grade, year, sex and orientation of classes in the semesters of 2013/14 to 2015/16.

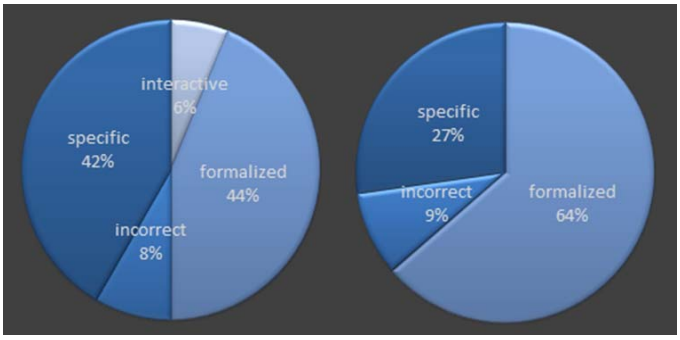
Fig. 2. Study groups



In Figure 3 we can see the results and the evolution of the step-by-step method at the same time. In this figure we can compare the 9th science (left) and 9th language (right) orientation study group.

Our experience was that a not science oriented class can reach level 3 of the discussion, and can start to think more mathematically with the help of studying computer programming. They need more time for practice in lessons (they have 3.5 lessons per week) and we let them work in pairs, but they started to think more consciously when solving problems, and they like these topics.

Fig. 3. Discussion: Science and language orientation groups



According to Piagét's theory of cognitive development, and their adaption in mathematical didactics [18] Level 1 is feasible already in the age of 10-12. In Hungarian mathematics curricula working with formulas start at the 7th grade (almost at 13 years old pupils), and we can expect the Level 2 and higher discussion from that point. Most of the word problem types we organize and teach in details in 7th-8th grades (in the age of 13-14). So this is the suitable age for starting to implement these tasks for computer programs.

For gifted children in mathematics or in computer programming (age 14+) the aim must be to reach level 4, the whole and specific mathematical discussion level, the others (not talented, or not scientific-engineering oriented) must reach level 3.

We agree with Igor Rivin [26] that a computer program could be a mathematically correct proof, and we can add that if a computer program satisfies the fourth level of criteria discussed above, and the program is able to solve not just a particular, but also its generated task (theorem), and we cannot spoil it with "stupid" input data, then we can accept this program as a proof of the particular mathematical theorem.

IX. Conclusions, further questions

The levels we described function as the bases for teaching instructions and the evaluation of pupils' work. From our experience, we can establish what expectations we have at different levels, and lay down a norm both for gifted children and for everyone else. Furthermore we have to make an impact assessment of our method in our study groups. If the impact assessment concludes with a satisfactory result we continue the experiments following a wider pattern, involving other schools and teachers in order to spread and extend the method.

For proving the impact of the method, we are created a final test for answering the following questions:

- Whether analogies in teaching methodology are helpful or not, when they have been achieved as already an engraved method from elementary school?
- Whether children recognize mathematical analogies or not in certain programming environments?
- Can this approach help to overcome fear and aversion from computer programming or not?

Finally, we have to say, that our first aim with teaching experiments was not to measure children's performance and compare them to each other. We hope we create a method which can help children to understand: computer programming is not a new (fearful) knowledge for them, but the direct continuation of mathematical problem solving with the help of a machine.

REFERENCES

[1] B. Csapó, E. Korom and G. Molnár, Eds., The content framework for online diagnostic assessment of mathematical knowledge. Budapest: OFI, 2015. (In Hungarian)

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

[3] R. Skemp, The psychology of learning mathematics. New York, NY: Routledge, 1987.

[4] G. Ambrus, Real-life mathematics. Budapest: Műszaki, 2007. (In Hungarian)

[5] OECD PISA results. http://www.oecd.org/pisa/

[6] K. Takácsné Bubnó and V. Takács, "Solving word problems by computer programming" in Problem Solving in Mathematics Education, Eds. A. Ambrus and É. Vásárhelyi, Budapest: Eötvös Loránd University, Faculty of Science, Institute of Mathematics, 2014., pp. 193-208.

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

[8] IEEE&ACM Report 2013. Curriculum Guidelines for Undergraduate Degree Programs in Computer Science. December 20, 2013. The Joint Task Force on Computing Curricula Association for Computing Machinery (ACM) IEEE Computer Society. http://www.acm.org/education/CS2013-final-report.pdf

[9] M. Csernoch and P. Biró, "Problem solving with computers,". Tud. Műsz. Táj 62, pp. 86-94., 2015, (In Hungarian)

[10] J. F. P. Ferreira, "Principles and applications of algorithmic problem solving," Thesis submitted to The University of Nottingham for the degree of Doctor of Philosophy, 2010.

[11] Z. Michalewicz and M. Michalewicz, "Puzzle-based learning," Melbourne: Hybrid Publ., 2008.

[12] P. Biró and M. Csernoch, "The mathability of computer problem solving approaches," 2015 6th IEEE International Conference on Cognitive Infocommunications (CogInfoCom), Győr: IEEE 2015., pp. 111-114.

[13] S. Papert, Mindstorms, 2nd ed., New York: Basic Books, 1993.

[14] C. Buteau, E. Muller, N. Marshall, "When a university mathematics department adopted core mathematics courses of an unintentionally constructionist nature: really?", Digital Experiences in Math. Educ. vol. 1, pp. 133–155., 2015.

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

[16] V. L. Takács, "BlockImpress," IEEE 18th International Conference on Intelligent Engineering Systems INES 2014, Tihany, 2014, pp. 221-226.

[17] P. Domokos and M. Széll, Blocklino. http://blocklino.org/apps/blocklino/index.html

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

[19] B. Kallós, "Teaching experiment with algebraic proofs," in Handbook of mathematics teaching improvement, ed. S. Turnau, Rzeszów: University of Rzeszów, pp. 237-250., 2008.

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

[21] K. Chmielewska, A. Gilányi and A. Łukasiewicz, "Mathability and mathematical cognition," 2016 7th IEEE International Conference on Cognitive Infocommunications (CogInfoCom), Wroclaw: IEEE, 2016, pp. 000245-000250

[22] 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.

[23] Pupil's solution of 'wooden house problem', https://blockly-demo.appspot.com/static/demos/code/index.html?lang=hun#f4abe2 (In Hungarian)

[24] Z. Bécsi, B. Bojda, K. Takácsné Bubnó, P. Domokos, A. Kelemenné Nagy, V. L. Takács, "Mobile programming in block languages" in Proceedings MAFIOK XXXIX Conference, Kaposvár, 2015. p. 83-88. (In Hungarian)

[25] C. Csíkos, "Mathematical proofs in school, and prooving capability". Magy. Pedag. vol. 99. pp. 3-21., 1999., (In Hungarian)

[26] I. Rivin, "Some thoughts on the teaching of mathematics: ten years later," Notices of the AMS, vol. 61, pp. 597-602., Jun/July 2014.

[27] F. Schneider, First steps of computer programming. Gyula: APC-Stúdió, 1998.. (In Hungarian)

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

[29] 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: IEEE, 2014, pp. 583-586.

[30] 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: IEEE, 2013, pp. 869-872.

[31] A. Gilányi, N. Merentes and R. Quintero, "Mathability and an animation related to a convex-like property," 2016 7th IEEE International Conference on Cognitive Infocommunications (CogInfoCom), Wroclaw: IEEE, 2016, pp. 000227-000232.

[32] A. Gilányi, N. Merentes and R. Quintero, "Presentation of an animation of the m-convex hull of sets," 2016 7th IEEE International Conference on Cognitive Infocommunications (CogInfoCom), Wroclaw: IEEE, 2016, pp. 000307-000308

[33] S. Czirbusz, "Regularity of functional equations and computer algebra systems," 2016 7th IEEE International Conference on Cognitive Infocommunications (CogInfoCom), Wroclaw: IEEE, 2016, pp. 497-502.