Received October 14, 2021, accepted November 3, 2021, date of publication November 8, 2021,
date of current version November 23, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3126757

Developing Computational Thinking Skills With Algorithm-Driven Spreadsheeting

MÁRIA CSERNOCH[1], PIROSKA BIRÓ[1,2], AND JÁNOS MÁTH[3]
[1]Faculty of Informatics, University of Debrecen, 4032 Debrecen, Hungary
[2]Socio-Human Sciences and Engineering, Faculty of Economics, Sapientia Hungarian University of Transylvania, 530104 Miercurea Ciuc, Romania
[3]Faculty of Humanities, University of Debrecen, 4032 Debrecen, Hungary

Corresponding author: Mária Csernoch (csernoch.maria@inf.unideb.hu)

This work was supported in part by the construction under Grant EFOP-3.6.3-VEKOP-16-2017-00002, and in part by the European Union through the European Social Fund.

This work involved human subjects or animals in its research. The authors confirm that all human/animal subject research procedures and protocols are exempt from review board approval.

ABSTRACT The paper presents the details of a four-year project to test the effectiveness of teaching spreadsheeting with spreadsheet programming, instead of the traditional, widely accepted surface approach methods. The novel method applied in the project, entitled Sprego (Spreadsheet Lego), is a concept-based problem-solving approach adapted from the didactics of other sciences and computer programming. In the experimental group contextualized, real-world programming problems are presented in a spreadsheet environment. A semi-unplugged data-driven analysis is carried out based on each problem, which is followed by the building of a feasible algorithm, expressed by natural language expressions. The coding is completed in the following step by applying a limited number of spreadsheet (Sprego) functions, multilevel, and array formulas. The final steps of the process are discussion and debugging. On the other hand, classical, tool-centered approaches are applied in the control groups. Our research reveals that the traditional surface approach methods for teaching spreadsheeting do not provide long lasting, reliable knowledge which would provide students and end-users with effective problem-solving strategies, while Sprego does. Beyond this finding, the project proves that Sprego supports schema construction and extended abstraction, which is one of the major hiatus points of traditional surface navigation methods. The project also reveals that developing computational thinking skills should not be downgraded, and the misconceptions of self-taught end-users and user-friendly applications should be reconsidered, especially their application in educational environments. Gaining effective computer problem-solving skills and knowledge-transfer abilities is not magic, but a time-consuming process which requires consciously developed and effective methods, and teachers who accept the incremental nature of the sciences.

INDEX TERMS Algorithm-driven spreadsheeting, long lasting knowledge, schema construction, cognitive load, end-user computing, computational thinking.

I. INTRODUCTION

"Each problem that I solved became a rule which served afterwards to solve other problems." [1]

The major concern of the present study is whether a thoroughly developed algorithm-driven spreadsheet programming method could serve the development of students' computational thinking skills in tertiary education. The focus is on (1) the improvement of software engineering students' algorithmic skills, (2) their computer problem-solving abilities through knowledge-transfer activation, and (3) how Sprego programming (Spreadsheet Lego) [2]–[4] could prepare them for further studies in informatics/computer sciences, especially in data-management and imperative and/or object-oriented programming languages. To provide the answer, we offer the results of a four-year study of testing and analyzing post-secondary and first year university students of informatics on their spreadsheet programming abilities. In the project, the effectiveness of the widely accepted, tested, and commercialized surface-approach methods were compared to the algorithm- and schema-driven Sprego programming. The question was

The associate editor coordinating the review of this manuscript and approving it for publication was Muhammad Imran Tariq.

whether students in tertiary computer science education needed such simplified programming instructions or knowledge built up in ''serious'' programming experiences would serve them in other environments, such as spreadsheets.

It is not surprising that the two most frequently asked questions considering computer science/informatics education are ≪ Should we teach students to program? ≫ and ≪ Should all students learn to program? ≫. Our local surveys – repeated in classes and in recently organized conferences – revealed that most students, teachers, and scholars are convinced that these questions were presented around the time that Wing [5] proposed her computational thinking approach in 2006. However, they have been lying around unsolved for about thirty years [6]. Soloway and his colleagues clearly declared that programming is ubiquitous, and it should be

– expanded to end-user computing,
– socially sanctioned intellectual advances for everyone,
– embedded in a rich cognitive context.

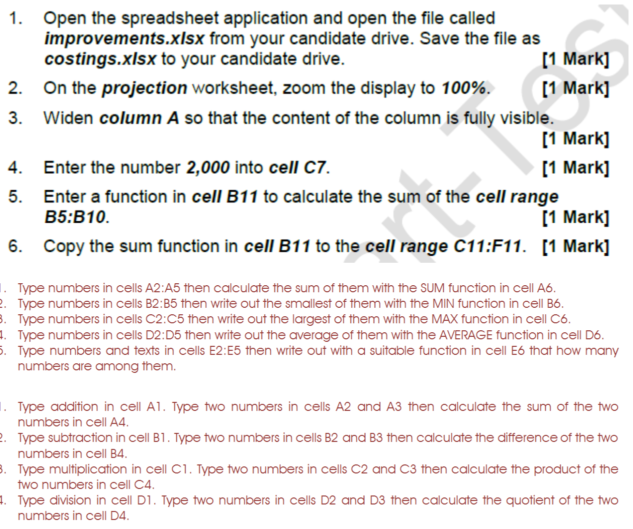
They also expressed the ideas that creating a computational medium requires making programming easier to learn and do, and requires expressiveness and usefulness. Considering the same problems, the misconceptions circulating around programming were brought up twenty years later, clearly indicating that not much had changed [7]. This happened in spite of the fact that the power of spreadsheeting was recognized: "There is clear potential to generate significant benefits by developing improved methodologies for many of the very important activities performed by millions of people who interact with spreadsheet information systems." [8]. However, the ''improved methodologies'' primarily focus either on the placement of spreadsheeting in a novel programming interface [9]–[12] or on reducing the complexity of spreadsheet interfaces [13]–[15]. The choice of methodology development might be influenced by the results of [15]. The authors of the paper found that out of four abilities and skills – logical reasoning, spatial visualization, mnemonic, and sequencing – only logical reasoning may be developed by spreadsheeting. Nevertheless, we have found that it is the tool-centered, low-mathability approaches [16] that narrow the list not spreadsheeting, since Sprego programming has positive effect on two more abilities, namely spatial visualization and sequencing [17], [18].

Beyond this potentials, we also have to deal with the risks of spreadsheet programming, thoroughly detailed, analyzed, [13], [20]–[29], and solved to some extent [30]–[34]. In general, the wider research community tends to focus on the financial losses caused by poorly designed spreadsheets rather than on the development of ''real'' spreadsheet methodologies. In this context ''real'' stands for methodologies where the programming (coding) happens on the spreadsheet interface by taking advantage of the built-in functions and the well-developed graphical interface. This shortcoming is clearly indicated by the fact that the thirty-year-old methodological problems are still with us. At this time, we must formulate a novel question ≪ Why have we not been able to carry out what was so clear thirty years ago, and ten years ago? ≫. Nothing has happened, in spite of the awareness that "For many people, the programming language of choice is a spreadsheet. In fact, spreadsheets are probably the most widely used end-user programming systems. The people who use spreadsheets to program are often end-user programmers. End-user programmers are people who often have little or no training in programming but still do some amount of programming. In the U.S. alone, the number of end-user programmers is conservatively estimated at 11 million, compared to only 2.75 million other, professional programmers.", and these numbers have increased since this estimate was made [14], [21], [26], [31], [34], [35], [36].

Finding solutions to these problems is far beyond the scope of the present study; yet some of the questions closely related to its focus can be answered. The following is a list of well-documented problems to which Sprego and its teaching-learning methodology could provide effective solution:

– over-mystification of the imperative and the object-oriented languages and misinterpretation of programming and coding [6], [35], [37]–[39];
– downgrading of end-user programming by IT professionals, corporate managers, and information systems researchers. Spreadsheets are almost invisible in the information systems and computer science research communities 8], [11], [40]–[43], are considered boring [44], [45], and involve mindless, routine tasks [46], and are reduced to computer driving licenses and product knowledge [47];
– decontextualized teaching materials [48]–[51], and interface-centered approaches to end-user activities 21], [49], [50], [28], [52]–[55] (Fig. 1);
– misleading user-friendly slogans from the leading software companies whose effectiveness have never been proved 56], [22], in accordance with software-centered exams [28], [50] (with 40 hours of study time [57]), a lack of schemata for utilizing fast thinking [1], [40], [58]–[65];
– computer illiterate teachers at all levels of education, which has sadly been proved during the pandemic, missing aspects of TPCK (Technological Pedagogical Content Knowledge) [66], [67], self-nominated educators of informatics [68], and experienced but not expert teachers in informatics [69], [50]. Pólya [58] clearly stated that "Thus, a teacher of mathematics has a great opportunity. If he fills his allotted time with drilling his students in routine operations he kills their interest, hampers their intellectual development, and misuses his opportunity. But if he challenges the curiosity of his students by setting them problems proportionate to their knowledge, and helps them to solve their problems with stimulating questions, he may give them a taste for, and some means of, independent thinking." The same is true, but left unattended in teaching informatics in general, and in end-user computing in particular;

– illiterate end-users [19], [28], [40], [52], [70]–[72] and unreliable self-evaluations – considering the level of digital competence [22], [73]. Wolfram [51] went even further by claiming that "The finery of writing odd maths symbols doesn't seem to me to be the essence of maths. ... With no effective, general education in computational thinking, most people can easily be misled, and they are." The same is true in end-user computing. Following the instructions of computer-cooking course-books and tests (Fig. 1), navigating on graphic interfaces at lightning speed does not develop computer problem-solving abilities;
– accepting the myth of the digital native and the multi-tasker generations proposed by Prensky [74] has been clearly rejected recently, and been proved to be only a myth [75], [76].



1. Open the spreadsheet application and open the file called ***improvements.xlsx*** from your candidate drive. Save the file as ***costings.xlsx*** to your candidate drive. **[1 Mark]**
2. On the ***projection*** worksheet, zoom the display to ***100%***. **[1 Mark]**
3. Widen ***column A*** so that the content of the column is fully visible. **[1 Mark]**
4. Enter the number ***2,000*** into ***cell C7***. **[1 Mark]**
5. Enter a function in ***cell B11*** to calculate the sum of the ***cell range B5:B10***. **[1 Mark]**
6. Copy the sum function in ***cell B11*** to the ***cell range C11:F11***. **[1 Mark]**

1. Type numbers in cells A2:A5 then calculate the sum of them with the SUM function in cell A6.
2. Type numbers in cells B2:B5 then write out the smallest of them with the MIN function in cell B6.
3. Type numbers in cells C2:C5 then write out the largest of them with the MAX function in cell C6.
4. Type numbers in cells D2:D5 then write out the average of them with the AVERAGE function in cell D6.
5. Type numbers and texts in cells E2:E5 then write out with a suitable function in cell E6 that how many numbers are among them.

1. Type addition in cell A1. Type two numbers in cells A2 and A3 then calculate the sum of the two numbers in cell A4.
2. Type subtraction in cell B1. Type two numbers in cells B2 and B3 then calculate the difference of the two numbers in cell B4.
3. Type multiplication in cell C1. Type two numbers in cells C2 and C3 then calculate the product of the two numbers in cell C4.
4. Type division in cell D1. Type two numbers in cells D2 and D3 then calculate the quotient of the two numbers in cell D4.

FIGURE 1. Examples of decontextualized tasks from testing and teaching materials.

At present, informatics as a school subject is reclaiming the infamous title of being the most hated topic from maths: "...mathematics has the dubious honor of being the least popular subject in the curriculum...Future teachers pass through the elementary schools learning to detest mathematics...They return to the elementary school to teach a new generation to detest it." [58]. Beyond being hated, informatics seems as secluded as mathematics [51].

It is clear that we are in need of fundamental changes, as stated by so many educators. The present paper clearly demonstrates that neither primary nor secondary school students can handle end-user problems from a programming point of view. Furthermore, software engineering students of informatics in tertiary education face the same problem. The research detailed in the paper reveals that the high-mathability [16], [77], [78] Sprego programming [2]–[4] would serve both end-users and professional software engineers in the development of computational thinking skills and knowledge-transfer activation.

A. SURFACE-CENTERED VS. ALGORITHM-DRIVEN END-USER APPROACHES

For the placement of the different teaching-learning approaches in informatics, the typology of computer problem-solving approaches would serve us. The typology defines two hypernym categories – deep and surface approaches –, and within them two and three hyponyms, respectively – concept-based and computer algorithmic and debugging-based; algorithm-based, information-based and trial-and-error wizard-based [79]–[81], [50]. The hypernym categories are in complete accordance with the problem-solving systems of various sciences (Table 1):
– mathability – high vs. low [16]
– evidence utilization – innovation-led-evidence vs. evidence-led-innovation [51], [82]
– higher order thinking skill (HOTS) – [concept-based] problem-solving vs. scientific approach [65]
– minimalist learning theory (MLT) – active-user vs. end-user [83], [84].

TABLE 1. Systems of problem-solving approaches in various sciences.

	Deep	Surface
mathability	high-mathability: based on existing means of the system, develop new programs and functions for solving new problems	low-mathability: use of existing functions and methods provided by a system
innovation	innovation-led evidence	evidence-led innovation
HOTS	[concept-based] problem-solving	scientific approach
MLT	active-user	end-user

In general, deep approach methods are problem-driven, while surface approaches focus on tools.

Considering the three hyponyms of the surface approach methods, we must make a distinction between the algorithm-based and the other two methods – information-based and trial-and-error wizard-based – due to the thinking mode [62] which is activated in the process of problem-solving. In the information- and the trial-and-error wizard-based methods slow thinking is activated, which has been proved error-prone [40], [62]. On the other hand, the algorithm-based methods activate reliable and effective fast thinking [62] by using and calling schemata built up with computer algorithmic- and debugging-based methods [58]–[61], [63], [64].

B. SURFACE-CENTERED SPREADSHEETING

The essence of the surface approach methods is that they tend to present details on the interfaces, wizards, or helps; they place emphasis on non-spreadsheeting (e.g., typing, text formatting) or minor issues, mismatch knowledge-transfer and program-specific items, but leave real problem-solving unattended [13], [14], [15], [51], [85]. The problems of surface approach methods in relation to computers was expressed as early as 1981 [86], but only a few listened: "It is sometimes thought that, with the availability of the hand-calculator and its big brother, the computer, the use of mathematics is reduced to pushing buttons and feeding in canned programs. This view would be quite false." Wolfram [51] goes one step further and claims that "When a major new machinery comes along – as computers have – it's rather disorientating." With the recently published and readily available digital tools, this fear has become reality. End-users, including teachers, cannot realize that "It is the drudgery that has been eliminated from mathematics by these modern devices and not the need for thought. We assuredly still need to recognize when a problem is suitable – and ripe – for mathematical treatment, and we need to plan a strategy for tackling the problem." [86] and "Machinery has mostly replaced human brawn not human brain." [51].

However, warnings are overridden by the marketing voices of software companies, by teachers' lack of belief in the incremental nature of science, expressed in the Meaning System Model [87], and by the sunk cost fallacy [62].

Traditional spreadsheet-educational materials, on the one hand, do not present real-world problems (Fig. 1). Furthermore, these surface approach methods do not consider any proven problem-solving methods which work effectively either in STEM or computer programming [56], [88]–[90]. One further undesired effect of surface approach methods is that they lead to false conclusions by claiming that end-user activities are boring [44], [45] routine [46], and are only suitable for low-level secretaries, and as such, should be banished from computer education.

C. SPREADSHEET PROGRAMMING

Spreadsheet programs are officially meant to serve end-users, but it is thoroughly documented that they are Integrated Development Environments (IDE) [8], and as such spreadsheet programming is an example of exaptation [91]. (Exaptation is typically used to describe a change in the function of a feature during the evolution process, and is not only applicable in biological evolution but also in technological innovation.) It is also claimed by Hatamleh & Tilesch [91] that "Currently there is an abundance of intellectual property that can be repurposed or used in areas and functions outside of their original intended application.", and it has been found that spreadsheet programming is one of them [2]–[4], [8]–[12], [17], [18], [21], [30], [33]–[35], [41]–[43], [53], [55], [96], [97]–[101], [109], [110].

One of the greatest advantages of spreadsheet programming is the simplicity of syntax. Several of the major concerns of teaching programming with imperative and object-oriented languages are not present in spreadsheet environments [10], [80], [92], [93]. On the other hand, a well-established concept of function and practice arrives from mathematics [51], [82], [94], [95], which is enough for spreadsheet programming [92]. By making the syntax simple, there is more room for solving real-world problems [51], [82], [95], handling real data, building algorithms, and discussing and debugging outputs. In general, making students interested in both handling data and programming [10], [51], [53], [55], [80], [82], [95]–[101].

II. SPREGO

A. THE ESSENCE OF SPREGO

In a nutshell, Sprego (Spreadsheet Lego) [2]–[4] primarily follows the concept-based problem-solving method of Pólya [58] supported by further studies and theories, such as the cognitive load theory [63], [64], the psychology of teaching mathematics [102], the theory of thinking fast and slow [62], and the meaning system model [87].

In the following, the exaptation from end-user spreadsheeting to Sprego programming is summarized (without repeating the theoretical background detailed above).

- the adaptation and combination of the above mentioned theories to teach spreadsheeting and spreadsheet programming aiming at active-users;
- developing programming and knowledge-transfer-centered tasks based on real-world data sources. In this way the widely accepted decontextualized classwork (Fig. 1) can be replaced with interesting and motivating contents;
- introducing concept-based methods focusing on problems and their algorithms, instead of tools. This is carried out by (1) avoiding data-typing and meaningless data sources, (2) reducing the number of functions to a dozen (with possible extensions) (Table 2) and avoiding function wizards, (3) whenever it is possible omitting formula-copying. By leaving behind these surface-handling methods, there is the opportunity for building up firm schemata for activating fast thinking in solving future tasks;
- inventing and introducing unplugged and semi-unplugged teaching materials and tools for better understanding and making classes more enjoyable [103]–[108].
- using array formulas [109], [110]. One advantage of array formulas is that one of the most frequently occurring errors originating in formula-copying can be reduced [26], [40], [111]. Array formulas also serve the introduction of the concept of n-dimensional vectors and operations on these vectors, knowledge which can be transferred to programming and database management.

TABLE 2. The basic set of sprego functions [2].

Sprego Text	Sprego Number	Sprego Pro
LEFT()	SUM()	IF()
RIGHT()	MIN()	ISERROR()
LEN()	MAX()	MATCH()
SEARCH()	AVERAGE()	INDEX()

The aims of this method, beyond teaching spreadsheeting effectively, are:

– preparing students for higher and ''serious'' informatics – programming, database management,
– developing their computational thinking skills and abilities,
– strengthening mathematical notions through spreadsheet problem-solving and practice,
– solving real-world problems,
– giving students the chance to see beyond interfaces and pure surface-navigation,
– revealing the incremental nature of the sciences.

Our previous teaching and research experiences and tests have already proved the effectiveness of Sprego [17], [18], [73], [112]–[116] compared to surface approach methods. These studies have been carried out primarily in primary and secondary education and with relatively small samples in tertiary education. In the present paper, we would like to emphasize that Sprego may be introduced at all levels where teachers are open to algorithm-driven spreadsheet teaching, and are able to completely leave behind the surface approach methods. The latter condition is extremely important, because we also found proof that mixing the surface and the deep approach methods causes more harm than good [116]. Our previous papers and books [2]–[4], [77], [78] provide the details of the methods, and can be adapted to any spreadsheeting environment.

B. DESCRIPTION OF METHODOLOGY

According to the principles of Sprego, strictly real-world data sources are presented in the form of 1NF (first normal form) tables during both lectures and seminars. The contents of tables are in accordance with age, background knowledge, and students' interests (e.g., local food places, game boards, geographical contents, movies, youtubers, sports).

Due to the limited time available for the subject, pre-prepared tables have been uploaded to a website accessible for the class.

The first step of the teaching-learning process is the thorough analysis of the actual table. In this phase, the number of data fields, records, and data types are discussed and decided.

Being aware of the nature of the data table, (1) problems are presented by the teacher – later on by the students. (2) The problem is discussed and analyzed, all the characteristics and connections describing the correspondent data are collected, written on the blackboard (teacher) and copied in exercise books, notepads, etc. (students). (3) Next, the algorithm of the problem is discussed and built along with the input and out values and their data types. Using natural language expressions, the algorithm is written on the blackboard and various unplugged tools by both the teacher and the students. The most frequently used unplugged tools to take notes and demonstrate multilevel functions are matryoshka dolls (original or 3D-printed), origami boat sets and toy barrel sets. (4) The next stage is coding. In the coding process, as detailed above, a limited number of spreadsheet functions – Sprego functions, Table 2 – are applied to reduce the cognitive load of the ≈500 built-in spreadsheet functions [117]. Furthermore, whenever it is possible, array formulas are created [2], [109], [110] to reduce copying errors [20], [26] and making formulas as secure as possible. (5) The final step of the problem-solving process is discussion and debugging. Here it must be emphasize here that this step is as important in spreadsheet programming as it is in other programming languages. Furthermore, being aware of the relatively high number of errors in spreadsheet documents, utmost care must be taken to discuss and debug spreadsheeting issues [13], [20]–[29].

Using unplugged tools along with role-play in tertiary education might seem childish. However, it has been found that first year students are as keen on play-along as K-12 students. In addition, the interviews following these experiences proved that these occasions helped students understand algorithms, including those whose codes they have already known by heart.

III. STUDY

A. SCHEDULING

The research took place over a four-year period, collecting data in three pre-tests (PRET), and in three delayed post-tests (DPOSTT). In both cases, approximately twenty minutes were given to complete the paper-based tests.

Both the experimental (G1) and the control groups (G2) were made up of students starting their studies in tertiary computer science education. According to the national curriculum, all the participating students studied spreadsheeting in elementary and high school. Most of them took the maturation exams in informatics, which include spreadsheeting [118], [119]. On average, our first year university students completed these exams with excellent results, and no significant differences were found between the experimental and the control groups. Officially, students arrived at the university well prepared in spreadsheeting.

The pre-tests were carried out in the very first week of the first semester of the students in their tertiary education. At this time the students' brought in knowledge was tested. The delayed post-tests were administered at least one year after the treatment, depending on the students' availability. The students who were tested in the delayed post-tests were still studying at the faculty.

The intervention took place in a two-week period, as part of the subject entitled Introduction to Informatics, which involved two-hour lectures and two-hour computer lab sessions every week, adding up to eight hours. This is an extremely short time, but both students and faculty members considered teaching spreadsheeting a waste of time.

The number of students participating in the test is presented in Table 3. Between the pre-tests and the delayed post-tests there was no further spreadsheet teaching and documented spreadsheeting in either the experimental or the

TABLE 3. The number of students participating in the pre-test (PRET) and the delayed post-test (DPOSTT).

	N	PRET	DPOSTT	PRET&DPOSTT
G1&G2	1680	1500	1023	853
G1	625	557	364	302
G2	1055	943	659	551

control groups. The reason for this is that spreadsheeting and spreadsheet programming is still not considered "serious" informatics, consequently it is absent from all the activities of the students. However, it is documented that both groups studied at least two semesters of high-level programming languages, covering at least three imperative and object-oriented languages.

Furthermore, students were not informed in advance of the tests, consequently they did not have the opportunity to prepare for testing. We can assume that they solved the tasks based on their knowledge derived from their last official encounter with spreadsheeting.

B. METHOD OF TESTING

Students were required to complete the tasks on the test papers, without any help, neither from their fellow students nor computer programs (helps, wizards, miscellaneous online sources). The reason for this "old-fashioned" testing method was to track how students create spreadsheet formulas, and we also wanted to identify the algorithms behind their solutions. These aims of the study cannot be carried out in spreadsheet environments due to the autocorrecting of syntactical errors, the loss of uncompleted formulas, and the loss of unsaved files.

C. SAMPLE

The number of students participating in the pre- and the delayed post-tests were determined by their availability. The pre-tests were administered in the first week of the students' tertiary studies, when there were quite a number of students attending the classes. One or more years later, however, the students' willingness to attend classes declined, and the high rate of dropout made them unavailable. In both G1 and G2 groups about half of the students filled in both tests (Table 3).

D. TREATMENT IN THE EXPERIMENTAL GROUP

In the experimental groups, both in the lectures and the seminars, the 'jug and mug' education – where students come to class and listen to the teacher without interruption, without any dialogue or interaction even when prompted to do so –, were replaced by inquiry-based learning [58]. The effectiveness of this method has been proved in various sciences, primarily in mathematics [51] and programming [121], [122]. Consequently, considering it in the context of teaching spreadsheet programming seemed reasonable.

The keywords associated with this intervention are presenting real-word contents, analyzing data, building algorithm(s), the simplicity of coding – calling up simple, general-purpose functions (Table 2) and building composite functions –, discussing and debugging step-by-step [75], [123].

During the experimental period, using various data sources, we start with problems handling texts, forming yes/no questions for both text/number output and conditional formatting, and handling error-values. Based on this knowledge, the next algorithm is solving conditional counting, summing, averaging, minimum and maximum. The essence of these problems is a three-step algorithm (1) a yes/no question for formulating the condition, (2) marking the TRUE answers with a selected value – 1 for counting, in other cases, values from the table –, and (3) carrying out the indicated operation. Advancing from the simplest conditional counting to the more and more demanding conditional problems, the same algorithm is applied. The next algorithm handled is the linear search, where a two-step algorithm is applied: (1) finding the record-index of the searched value, and (2) calculating and writing out the required value from the identified record.

Regardless of the algorithm in question, the same method is applied. In the introductory period, the focus is on the building of the schema, which is a deep approach method requiring slow thinking [60]–[64]. In the advanced period, the calling up of the built-up schema/schemata is carried out, which is a surface approach method requiring fast thinking [60]–[64].

The problems are solved with array formulas for (1) avoiding errors originated in the copying of the formulas, (2) delaying the introduction of the extremely demanding concepts of relative and absolute reference for as long as possible, and (3) preparing students for further studies in computer sciences (e.g., database management, programming).

The entire problem-solving procedure is carried out with the coaching method, where the teacher's role is best described as that of a moderator, who intervenes only to improve functioning (not infrequently with unplugged tools) and to avoid the development of misconceptions [13]. The problems are thoroughly analyzed, the algorithms built, the coding carried out, and outputs discussed [76]. One further characteristics of the method is the special coding process, where composite functions are built, in accordance with the limited number of functions (Table 2). In the coding process, the innermost step is carried out first, the output discussed, and then this output is presented as an argument or operand for the following step. This process is repeated until the final output is written out. Solutions are built up step-by-step in complete accordance with Pólya [58]: "A great discovery solves a great problem but there is a grain of discovery in the solution of any problem. Your problem may be modest; but if it challenges your curiosity and brings into play your inventive faculties, and if you solve it by your own means, you may experience the tension and enjoy the triumph of discovery. Such experiences at a susceptible age may create a taste for mental work and leave their imprint on mind and character for a lifetime.".

In addition to classwork, students were encouraged to complete homework tasks involving similar problems, and their effort was rewarded with extra points. They were also motivated to present real-world problems of their own in classes.

E. TREATMENT IN THE CONTROL GROUP

In the control groups the well-established surface approach method was applied, with which the students were familiar from their previous studies in elementary and high school.

IV. RESEARCH QUESTION

Our previous studies proved that teaching spreadsheeting with traditional surface approach methods does not develop long lasting knowledge [17], [18], [73], [112]–[116]. We found proof that students, as end-users, can pass the interface-centered exams with good or excellent results, but forget almost everything in a couple of weeks, between the maturation exams and the first week of their tertiary studies.

It was also found in our previously conducted studies in primary and secondary education that teaching spreadsheet with Sprego programming:

- is significantly more effective compared to the traditional surface-approach methods,
- long-lasting knowledge in the form of schemata can be built up and called on to activate fast thinking in problem-solving,
- surface approach methods applied in advance of Sprego programming are serious distractors.

The primary question of the present research is whether or not studying spreadsheeting with a deep approach method would result in long lasting knowledge similar to formal achievements. In a scenario in which students are overconfident because of the results of their maturation exams [118]–[120] and burnt-out due to mindless and meaningless surface approach methods in primary and secondary school, teachers encounter students who are completely uninterested in spreadsheeting. Furthermore, the question of time was our concern, i.e. whether the two-week period available in the curriculum would be enough to cover all the algorithms planned, or students would need more time even in a minimal-syntax IDE.

V. HYPOTHESES

(1) Sprego programming is more effective than traditional surface approach methods in teaching spreadsheeting in tertiary education.

(2) Sprego programming supports schema construction, and reliable knowledge is stored in long-term memory.

(3) Sprego programming attracts students and has a motivating effect even on those who are uninterested in the tool-centered approaches.

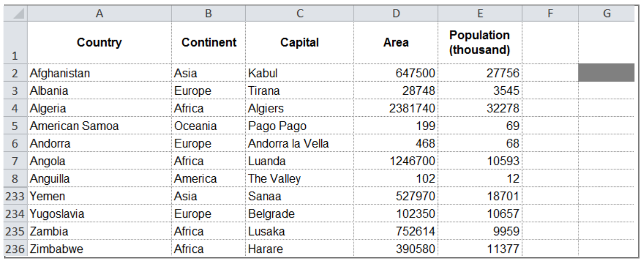
(4) Teaching spreadsheet programming, similarly to other programming languages, requires direct, strong instructional guidance.

VI. PROBLEMS OF THE TEST

A. THE TASKS

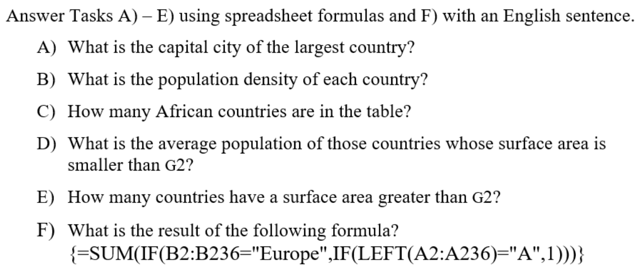
The primary aim of the tasks was to measure how students would be able recognize the algorithms and the schemata of the problems presented. To achieve this the following tools and tasks were provided.

- A data table of 5 fields and 235 records. Due to the length of the table, only a section of the table was presented, with the field names (row 1) and eleven records (rows 2–8 and rows 233–236) (Fig. 2).
- A variable in cell G2, whose value is unknown. (Fig. 2).
- Six tasks. Five tasks were to be answered with spreadsheet formulas and one with a native language sentence (a classical decoding task) (Fig. 3).



	A	B	C	D	E	F	G
1	Country	Continent	Capital	Area	Population (thousand)		
2	Afghanistan	Asia	Kabul	647500	27756		■
3	Albania	Europe	Tirana	28748	3545		
4	Algeria	Africa	Algiers	2381740	32278		
5	American Samoa	Oceania	Pago Pago	199	69		
6	Andorra	Europe	Andorra la Vella	468	68		
7	Angola	Africa	Luanda	1246700	10593		
8	Anguilla	America	The Valley	102	12		
233	Yemen	Asia	Sanaa	527970	18701		
234	Yugoslavia	Europe	Belgrade	102350	10657		
235	Zambia	Africa	Lusaka	752614	9959		
236	Zimbabwe	Africa	Harare	390580	11377		

FIGURE 2. **"The countries of the World" table, provided as the data source to solve the tasks presented in the tests.**



Answer Tasks A) – E) using spreadsheet formulas and F) with an English sentence.

A) What is the capital city of the largest country?

B) What is the population density of each country?

C) How many African countries are in the table?

D) What is the average population of those countries whose surface area is smaller than G2?

E) How many countries have a surface area greater than G2?

F) What is the result of the following formula?
{=SUM(IF(B2:B236="Europe",IF(LEFT(A2:A236)="A",1)))}

FIGURE 3. **The instruction and the tasks of the tests.**

B. SOLUTIONS: TASK B

Task B has one solution, leaving out of consideration how the output vector is created: array formula or copying. Both solutions were accepted if the presence of the vector was clearly indicated.

Task B contains three knowledge-transfer items:

- division (1),
- population divided by area (knowledge transferred from another subject) (1), and
- population presented in thousands (information retrieved from the sample table) (1).

There is only one spreadsheeting item (1) – how to create a vector output. The array formula (AF) solution is the {=1000*E2:E236/D2:D236} formula.

Four items were assigned to the task (the numbers of items are indicated in parentheses).

C. SOLUTIONS: TASK A

Task A is a linear search problem which officially has two solutions.

- In spreadsheet programming the shorter solution is when one output is assumed. In this case, the = INDEX(C2:C236,MATCH(MAX(D2:D236),D2:D236, 0)) composite function can be used.
- The longer solution is when there is no presumption regarding the number of possible outputs. Here the {=IF(MAX(D2:D236) = D2:D236,C2:C236)} array formula would provide the output vector. When the answer to the yes/no question is TRUE, the IF() function returns the name of the capital city (the marker is the capital city in this case). When the answer is FALSE, the IF function returns the default FALSE value. This solution shares the algorithm with Tasks C, D, and E).

However, considering these two solutions, there is no difference between the algorithms and the construction of codes. Only the applied functions are different.

Fifteen items were assigned to the task:

- the names of the functions (3)
- the positions of the functions (3)
- which one is encapsulated in which one
- the values and their argument positions for
- finding the largest area (1)
- finding the record-index of the largest area (6)
- writing out the capital city/cities (2)

Considering the capacity of long-term memory, in Task A, we must call attention to the falsely assumed use of HLOOKUP() and VLOOKUP() BIF functions, which cannot be applied in this instance, given the structure of the table. HLOOKUP() is ruled out because the data fields are arranged in columns, VLOOKUP() because the output field – Capital – is on the left side of the search field – Area. These restrictions – and many more [2] – make the BIF functions mentioned extremely difficult to use, because they require access to unnecessary and hard-to-recall information from long-term memory.

D. SOLUTIONS: TASK C, D, AND E

Tasks C, D, and E are different from A and B in the sense that they can be solved both with built-in-functions (BIF) (Table 5 and Table 6) and algorithm-driven array formulas (AF) (Table 4). The BIF solution is much preferable according to spreadsheet providers, course-books, and instructors blindly following the new trends of the software companies. On the other hand, the AF solutions require only one schema stored in long-term memory and two general purpose functions. In the case of the BIF functions, there are problem-specific functions with different syntactical rules and with confusing orders of arguments which can be used to solve the three similar problems.

The number of items in the array formulas were assigned to the solution by considering

TABLE 4. The AF solutions of Tasks C, D, and E. With the AF formulas the same algorithm can be used to solve all three problems.

Solutions with array formulas		Items
C	{=SUM(IF(B2:B236="Africa",1))}	10
D	{=AVERAGE(IF(D2:D236<G2,E2:E236))}	9
E	{=SUM(IF(D2:D236>G2,1))}	9

TABLE 5. One-conditional BIF functions to solve Tasks C, D, and E.

Solutions with one-conditional built-in functions		Items
C	=COUNTIF(B2:B236,"Africa")	6
D	=AVERAGEIF(D2:D236,"<"&G2,E2:E236)	10
E	=COUNTIF(D2:D236,">"&G2)	8

TABLE 6. Multi-conditional BIF functions to solve Tasks C, D, and E.

Solutions with multi-conditional built-in functions		Items
C	=COUNTIFS(B2:B236,"Africa")	6
D	=AVERAGEIFS(E2:E236,D2:D236,"<"&G2)	10
E	=COUNTIFS(D2:D236,">"&G2)	8

- the name of the functions (2)
- the positions of the functions (2)
- which one is encapsulated in which one
- values – the constant and the vectors involved (3)
- their argument positions (2)
- syntax (1)

The three tasks are solved with the same algorithm, so the number of items is the same. In Task C there is one extra point for handling the string constant.

The comparison of the three tasks reveals that considering the level of generalization Task C is the lowest, followed by E, and the most general is D.

Even though the three tasks are solved with the same algorithm, the number of items with the BIF functions varies depending on the followings:

- the operation: counting vs. summing or averaging,
- the container of the values: constant or variable,
- the logical operator: equality or inequality,
- concatenation: in the case of inequality with variable.

The items assigned to Tasks C, D, and E are 6, 10, and 8, respectively.

We must call attention to the different order of the arguments in one- and multi-conditional functions, which further deepens the confusion (for comparison see Table 5 and 6). However, the difference in the order of the arguments does not change the number of items assigned to the solutions.

Additional solutions can also be considered (e.g., Database functions). These solutions were accepted and evaluated according to the items detailed above.

E. SOLUTIONS: TASK F

The answer to Task F is: The number of European countries whose name starts with A. In accordance with the answer 3 items were assigned to the task:

- number of something (1)
- European (1)
- starts with A (1)

We must call attention to the structure of the test, which was intended to help students. If students were able to recognize the same algorithm of Tasks C–F, they would realize that Task F provides the syntax to the solution of the conditional calculations in Tasks C–E.

VII. METHODS

The data were processed with the help of Microsoft Excel and SPSS (Statistical Package for the Social Sciences). The students' results were recorded in Excel tables, year-by-year, according to the items detailed above. Most of the analyses of the descriptive statistics were carried out and the primary diagrams were also mapped with the help of the spreadsheet program.

After the pre-processing of the data, they were transferred to SPSS to conduct inferential statistics.

Beside descriptive statistics (average, deviation, variance, minimum, maximum), other statistical methods, probes were used to test the data in consideration of our hypotheses.

The following tests were used:

- Pearson Chi-squared test,
- Whitney U test,
- one-sample t-test for two independent samples,
- LEM – loglinear model,
- one-way ANOVA,
- two-way ANOVA.

VIII. RESULTS

A. RESULTS: TASK B

Considering Task B, a significant difference was found in the PRET test between the two groups, in which the results for group G1 are higher than those of G2 (Fig. 4). As mentioned above, the groups were assigned to the teachers randomly and there was no significant difference between their maturation exam results [73], [118], [119].

Since our concern is how the difference between the students in the experimental and the control groups developed from the pre-test to the delayed post-test – i.e. understanding what is stored in long-term memory –, we were testing the significance of the interaction. In this respect, a significant difference was found between the two groups (p = 0.012, Partial Eta Squared = 0.007). Fig. 4 clearly shows the changes in knowledge in the two groups. While group G1 achieved better results, the results of G2 students declined. As mentioned above, among the tasks, Task B involves the least 'spreadsheeting' in the sense that it requires knowledge-transfer and spreadsheet items in a ratio of 3:1, which means that teaching spreadsheeting deals with only one quarter of the required knowledge items.

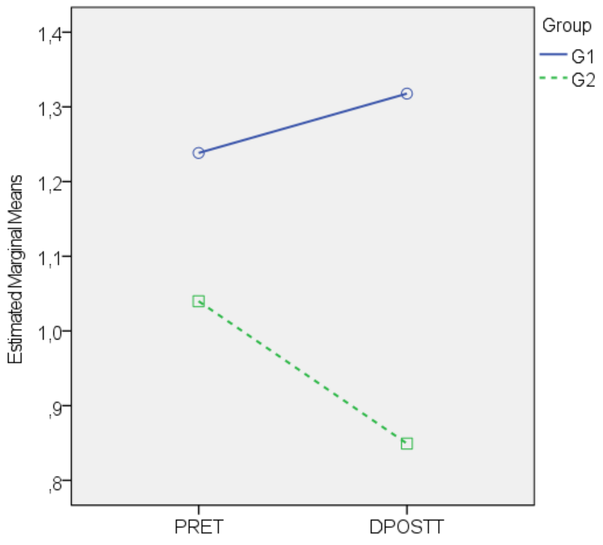


FIGURE 4. The results of Task B in tests PRET and DPOSTT.

B. RESULTS: TASK A

Task A is the least successful task of the test. Its major characteristic is that it requires the application of the algorithm of the linear search and the spreadsheet coding tools to create the formula. As mentioned above, the table and the order of the data fields rule out calling up the BIF functions; consequently, the students were required to build a multilevel solution, either with one formula or with substitute cells referring to previous outputs (Table 7, Fig. 5).

TABLE 7. Task A: the number of students who did not do anything (Ignored), who applied one of the incorrect BIF functions – VLOOKUP() or HLOOKUP() –, and who tried the correct IF() function.

	PRET		DPOSTT	
	G1	G2	G1	G2
Total	557	943	364	659
Ignored	232 (42%)	391 (42%)	153 (42%)	303 (46%)
VLOOKUP()	32 (5.75%)	89 (9.44%)	12 (3.3%)	33 (5.01%)
HLOOKUP()	7 (1.26%)	10 (1.06%)	1 (0.27%)	2 (0.3%)
IF()	3 (0.54%)	0 (0%)	15 (4.12%)	6 (0.91%)

Similar to Task B, the results of group G1 were higher than those of G2 in the pre-test. The significance of the interaction was checked, and we found a significant difference (p = 0.001, Partial Eta Squared = 0.013) between the two groups in the delayed post-test.

The results of group G1 show a moderate increase, while those of group G2 show a strong decline (Fig. 5). In both groups, there was only one item where a significant increase was detectable – providing the argument of the MAX() function (p = 0.016). The other items were the following: position of INDEX(): p = 0.056; output vector: p = 0.02; position of MATCH(): $p < 0.001$; search value: p = 0.009; position of search value: p = 0.002; MAX(): $p < 0.001$; search vector: p = 0.025; position of search vector: p = 0.008; column number: p = 0.007.

The difference between the two groups was that while in group G2 this was the only item which was higher in

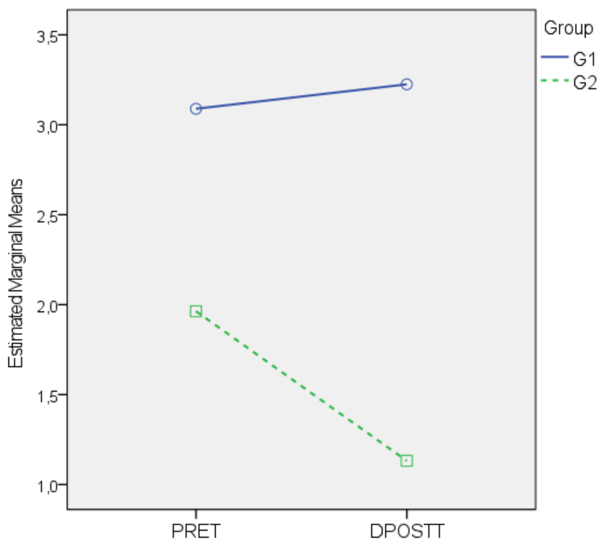


FIGURE 5. The results of Task A in tests PRET and DPOSTT.

DPOSTT than in PRET, in group G1 more than half of the items were higher in DPOSTT. We can conclude that the eight-hour-long block of the interaction was enough in group G1 to maintain fundamental knowledge, while in G2 students forgot almost everything. This result shows, on the one hand, that eight classes are not enough to teach programming, while on the other hand, our high-mathability spreadsheet programming approach, even in this short period of time, is more effective than low-mathability surface approach methods.

C. RESULTS: TASK C, D, AND E

As mentioned above, when solving Tasks C, D, and E there are two fundamentally different approaches which can be taken into consideration. One method is to solve these tasks with problem specific BIF functions, accompanied by several difficulties and incoherencies [2]. The other option is to build up the algorithm of the problems – a conditional calculation – and then carry out the coding with simple, general purpose functions.

The comparison of the interaction in Tasks C, D, and E considering groups G1 and G2 showed a significance difference between the two groups ($p = 0.054$, Partial Eta Squared = 0,004; $p < 0.001$, Partial Eta Squared = 0.038; $p < 0.001$, Partial Eta Squared = 0.032 respectively) (Figure 10). However, the results of the conditional calculating tasks revealed a different pattern compared to Tasks A and B.

Since Task C is the easiest among the three tasks sharing the same algorithm, it is no surprise that its result is the highest (Fig. 6, 7, and 8). Even in PRET, group G1 achieved a relatively high result. As the tasks become more and more general the results are lower and lower. This means that even if they understand the algorithm, students need more time to practice in order to increase the level of abstraction, to be able to build up a schema which can be recalled in similar situations.

The results further demonstrate that students studying with the spreadsheet programming method have a much higher

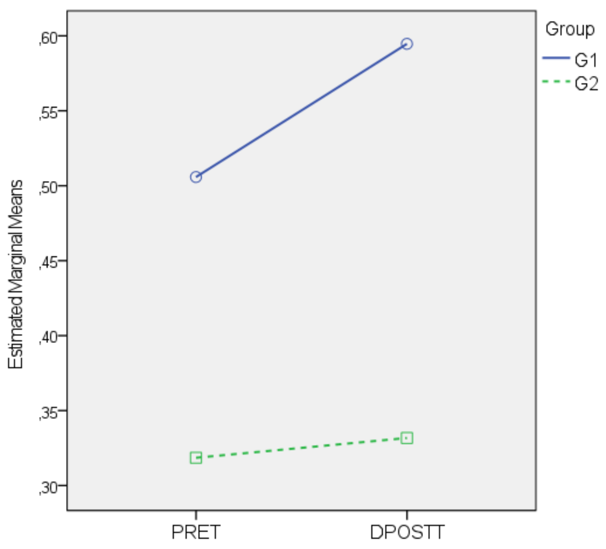


FIGURE 6. The results of Task C in tests PRET and DPOSTT.

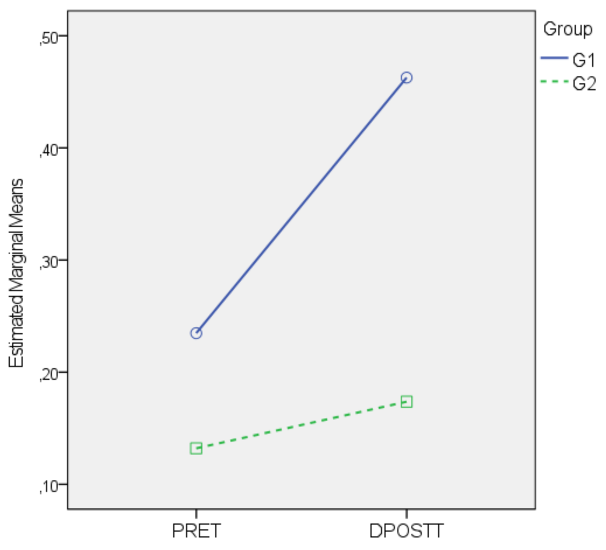


FIGURE 7. The results of Task E in tests PRET and DPOSTT.

abstraction level and, furthermore, a much more reliable fast thinking ability to call up schemata from long-term memory.

D. RESULTS: TASKS F

Task F is a classical decoding task, where a code is presented, and the students must answer it with a natural language sentence. This task does not require any special spreadsheet knowledge; it can be solved by recalling knowledge-transfer items from mathematics and programming, namely, how functions and composite functions work, how parentheses rule the order of execution, and how embedded conditions work (Fig. 9). The pattern is similar to that which is observed in Tasks C, D, and E: the results of group G1 significantly increased from PRET to DPOSTT (C: $p = 0.01$; D: $p < 0.001$; E: $p < 0.001$; F: $p < 0.001$), while in group G2 there was hardly any change. Between the groups the difference in

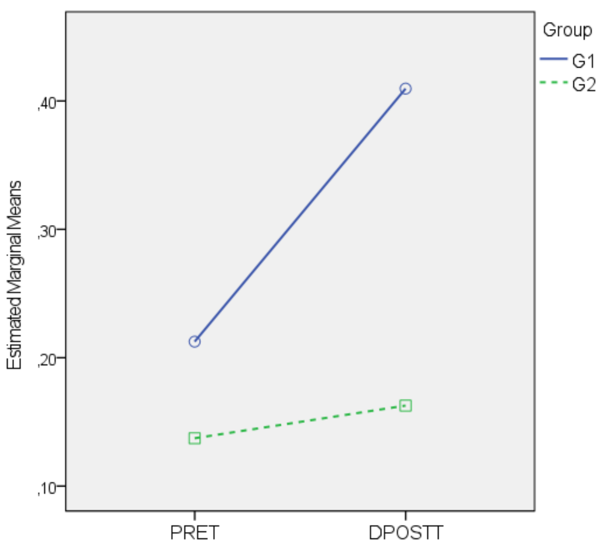


FIGURE 8. The results of Task D in tests PRET and DPOSTT.

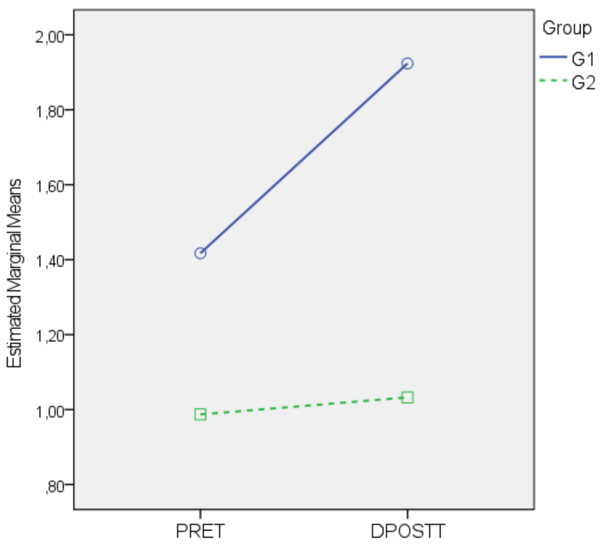


FIGURE 9. The results of Task F in tests PRET and DPOSTT.

the interaction is also significant (C: not significant; D: not significant; E: $p = 0.01$; F: $p < 0.001$).

Since Task F is an open question task there was space left, and the students used it, to make comments on the task. A notable number of students self-confidently declared that the formula was not correct. (When we tested teachers, the percentage of those who claimed that the formula was incorrect was even higher). Another phenomenon which must be mentioned in this context is the level of understanding according to SOLO categories [112], [124]–[126]. During the three semesters which passed between the pre- and the delayed post-test, the students studied programming for at least two semesters. In group G2 the number of students who were stuck at uni-structural level increased from PRET to DPOSTT, while in G1 this undesired phenomenon was not present.

The comparison of Tasks C, D, and E, on the one hand, clearly reveals the differences between the development of groups G1 and G2. On the other hand, it is also obvious that the level of abstraction which the three tasks require are different, and students cannot reach the highest SOLO category – extended abstract – in the eight classes of teaching.

IX. SELECTION OF PROBLEM-SOLVING APPROACHES

The next research question was how the traditional and the Sprego methods affect the selection of problem-solving approaches. At this stage of the analysis, only those students were taken into account who participated in both PRET and DPOST; 302 and 551 students, respectively. Three groups of the students were formed both in the experimental and the control group, based on the selected method (BIF or AF) and those who did not do the task (Ignored, I).

A. APPROACHES: TASK C

According to the algorithm, there is no difference between the three tasks; however, built-in spreadsheet functions make it complicated. Among them, Task C is the most common and simplest conditional calculation: counting with equality and a string constant. The number of students who selected BIF, AF, and I (Ignored) solutions for answering Task C (Table 8).

TABLE 8. Task C: selection of methods in PRET and DPOSTT.

	Test	BIF	AF	I	Total
G1	PRET	133	74	95	302
G2	PRET	178	114	259	551
G1	DPOSTT	23	182	97	302
G2	DPOSTT	73	199	279	551

The chi-square test proved ($p = 0.04$) that there is a connection between the PRET and DPOSTT solutions to Task C in group G1. The essence of the connection is that those who selected BIF in PRET did the same in DPOSTT with a greater probability than they selected AF or I. It is also proved that those who started with an array formula do not switch to built-in functions in DPOSTT. In group G2 a similar pattern can be revealed ($p < 0.001$); however, there is a greater probability that those who started with BIF stay with BIF.

We tested the difference between PRET and DPOSTT with a marginal homogeneity test. The number of students ignoring Task C does not change significantly from PRET to DPOSTT, neither in G1 (95 and 97 in PRET and DPOSTT) nor in G2 (259 and 279 in PRET and DPOSTT).

The selection of approach reveals a different pattern in the two tests. In group G1, the number of students solving the problems with BIF significantly decreased, while those who used AF increased ($p < 0.001$). In group G2 there is also a significant difference in the method selected; however, this is due to an increase in those ignoring the task ($p < 0.001$).

B. APPROACHES: TASK E

The second ranked task in terms of in complexity among the conditional calculations is counting with inequality and variable. This task is at a two-step higher level of abstraction than Task C: equality is changed to inequality in the condition, and the string constant of Task C is replaced by a variable.

The number of students who ignored the task increased compared to Task C, in both groups. The selection of BIF or AF shows a different pattern compared to Task C. The number of AF solutions increased in both groups, which indicates that even in PRET students intuitively select the algorithm-driven solution when the abstraction level is increased (Table 9).

TABLE 9. Task E: selection of methods in PRET and DPOSTT.

	Test	BIF	AF	I	Total
G1	PRET	55	63	184	302
G2	PRET	56	82	413	551
G1	DPOSTT	9	156	137	302
G2	DPOSTT	24	127	400	551

In DPOSTT, in terms of the number of tasks ignored there is huge difference between the two groups. In the experimental group the figure dropped from 184 to 137, while in the control group there was no change (413 vs. 400). The number of those selecting BIF decreased in both groups (G1: 55 vs 9, G2: 56 vs. 24). For AF, an increase was detected (G1: 63 vs. 156, G2: 82 vs. 127). However, the increase in the experimental group is much more noticeable than in the control group.

C. APPROACHES: TASK D

The most complex among the conditional calculations is calculating average with inequality and variable. In Task D a similar pattern was revealed as in Task E (Table 9 and 10).

TABLE 10. Task D: selection of methods in PRET and DPOSTT.

	Test	BIF	AF	I	Total
G1	PRET	21	103	178	302
G2	PRET	33	129	389	551
G1	DPOSTT	5	153	144	302
G2	DPOSTT	9	164	378	551

In PRET the number of tasks ignored does not change compared to Task E, in spite of the increase in the level of abstraction. However, considering the choice between BIF and AG an even greater preference for the AF solution over BIF can be detected.

In DPOSTT, the students from group G1 almost unanimously selected the algorithm-driven solution at the highest abstraction levels. In group G2 a similar pattern can be revealed but on a lower scale. The explanation for this change even in group G2 can be found in their background studies, which included at least two semesters of programming.

X. TYPES OF INTERACTIONS

Considering the type of interaction which distinguishes the two groups, it is obvious that the algorithm-driven spreadsheeting is significantly more effective than the traditional

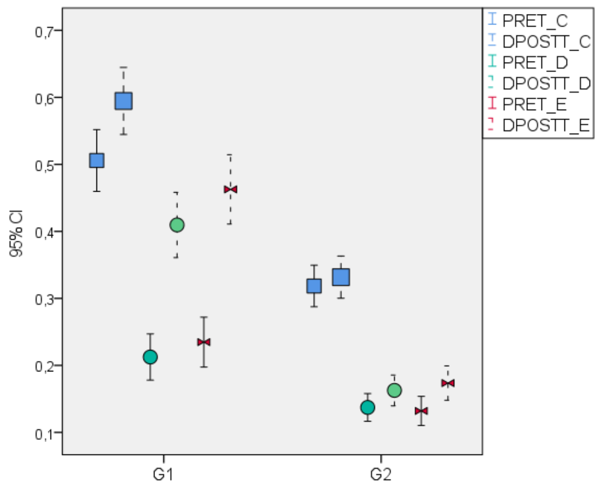


FIGURE 10. The results of Task C, D, and E in tests PRET and DPOSTT.

surface approach methods (Fig. 10). The algorithm-driven solution supports building up knowledge and schemata in long-term memory, which can be utilized in similar problems.

These results were further tested with Log-Linear Modelling (LEM), where three variables were considered: pre-test (PRET), delayed post-test (DPOSTT) with values BIF, AF, I, and the groups (G), experimental (G1) and control (G2). The question was what kind of relations can be revealed between the three variables. It was found that the model of PRET × G, DPOSTT × G, PRET × DPOSTT fits the data.

The connection parameters of PRET × DPOSTT reveal that in Task C those who used the BIF solutions in PRET tend to use the BIF solutions in DPOSTT. Those who used AF in PRET use AF or ignore the task in DPOSTT more frequently than would a random selection. A similar pattern was revealed when considering those students who ignored the task in PRET: selected AF or ignored the task, however this choice of selection is less attractive [15], [127], [128] (Table 11).

TABLE 11. The results of LEM in Task C when comparing PRET and DPOSTT, considering the selection of approaches.

		DPOSTT		
		BIF	AF	I
PRET	BIF	0.60	−0.24	−0.36
	AF	−0.39	0.15	0.24
	I	−0.21	0.09	0.12

The connection parameters of PRET × G reveal that at the beginning of the experiment more students in group G1 tend to solve the problems than in group G2 (Table 12).

After the APPROACH intervention, the parameters of the DPOSTT × G connection in the delayed post-test show that more students applied the AF solutions in the experiment group than in the control group (Table 13). This result shows the effectiveness of the Sprego interaction.

TABLE 12. The results of LEM in Task C in PRET, considering the selection of approaches.

	BIF	AF	I
G1	0.16	0.06	−0.22
G2	−0.16	−0.06	0.22

TABLE 13. The results of LEM in Task C in DPOSTT, considering the selection of approaches.

	BIF	AF	I
G1	−0.25	0.36	−0.11
G2	0.25	−0.36	0.11

TABLE 14. The results of LEM in Task E when comparing PRET and DPOSTT, considering the selection of approaches.

		DPOSTT		
		BIF	AF	I
PRET	BIF	0.56	−0.22	−0.34
	AF	−0.22	0.19	0.03
	I	−0.34	0.03	0.31

TABLE 15. The results of LEM in Task E in PRET, considering the selection of approaches.

	BIF	AF	I
G1	0.17	0.02	−0.19
G2	−0.17	−0.02	0.19

TABLE 16. The results of LEM in Task E in DPOSTT, considering the selection of approaches.

	BIF	AF	I
G1	−0.23	0.42	−0.19
G2	0.23	−0.42	0.19

The parameters show that the selection of the method is more possibly the same in the tests than a random choice would be (Table 14).

In PRET most of the students ignored the task in both the experiment and the control groups (the parameters of PRET are −0.37, −0.43, 0.8 for the solutions of BIF, AF, I, respectively). However, the students in group G1 tried to solve it more frequently, preferring the BIF solution to the AF (Table 15).

The pattern follows the pattern of task C: after the Sprego intervention, the parameters of the DPOSTT × G connection in the delayed post-test show that more students applied the AF solutions in the experimental group than in the control group. This result also shows the effectiveness of the Sprego interaction (Table 16).

In PRET, it is also shown that there is an extremely low number of students who tried the BIF solution in the case of task D, whose abstraction level is the highest among the three tasks with the same algorithm (the parameters of PRET are −1.26, 0.22, 1.04 for the solutions of BIF, AF, I, respectively). Furthermore, it was found that more students ignored the task in both the experiment and the control groups. The students in group G1 preferred the AF solutions more frequently than did those in group G2, while in group G2 the most preferred choice was to ignore the task.

TABLE 17. The results of LEM in Tasks D in PRET, considering the selection of approaches.

	BIF	AF	I
G1	0.02	0.13	−0.15
G2	−0.02	−0.13	0.15

Test DPOSTT reveals that selection of BIF is reduced compared to AF and I taken together (the parameters of DPOSTT are −2.24, 0.92, 1.32 for the solutions of BIF, AF, I, respectively) (Table 17).

Considering the method selected to solve Task D in DPOSTT, it was found that in group G1 the relative frequency of AF, while in G2 the relative frequency of Ignored, increased (Table 18).

TABLE 18. The results of LEM in Task D in DPOSTT, considering the selection of approaches.

	BIF	AF	I
G1	−0.02	0.24	−0.22
G2	0.02	−0.24	0.22

XI. DISCUSSION

A comparison of the conditional calculation tests with the different levels of abstraction allows us to draw the following conclusions.

In the experimental group, the number of students ignoring Task C (lowest level of abstraction) does not change significantly from PRET (95) to DPOSTT (97). Furthermore, in Task C the number of tasks ignored is the lowest (Task E: PRET, DPOSTT: 184, 137, Task D: PRET, DPOSTT: 178, 144), and there is no significant between PRET and DPOSTT (C: not significant; D: $p = 0.019$; E: $p = 0.001$). These findings prove that the interaction in the experimental group resulted in significant changes in cases in which schemata were not formed in previous studies. More importantly, Sprego programming attracts students (Hypothesis 3). On the other hand, the chi-square test indicates the tendency to the sunk cost fallacy, meaning that students cannot let go of the ''good old'' method, even though it has been proved less effective. This finding implies that students recognized the algorithm which connects the three tasks, but showed some reluctance in applying it in the simplest task. The LEN parameters show that the method selected is more possibly the same in the tests than a random choice would be.

In group G2, the number of students who ignored the tasks reveals a somewhat different pattern. No significant difference was found between the number of tasks ignored in the three tasks (PRET: 259, 413, 389, DPOSTT: 279, 400, 378 in Tasks C, E, D, respectively). They remained as resistant to spreadsheeting as before.

We can also conclude that the differences between the results of the two tests are not influenced by those students who ignored the tasks, but by the performance of those who worked on the problems. In general, it was found that Sprego is also more effective than the traditional approaches in tertiary education, just as it is in primary and secondary schools (Hypothesis 1).

The comparison of the method selected in both groups revealed that as the tasks become more abstract, students tend to use the algorithm-driven approach, or simply ignore the task; problem specific built-in functions do not play a role in their answers.

It is also clear that the algorithm-driven Sprego programming with its methodology provided the students with such firm background knowledge, in spite of the extremely short period of time available for teaching, that they switched from the surface approach methods (which they had studied for years) to the new one. This means that students studying in group G1 were able to build up schemata which were called upon in the delayed post-test (more than one year after the intervention), which proves Hypothesis 2. The results also indicate that the switch from the old methodology to the new has merit, since their results significantly increased despite the long gap between the pre- and the delayed post-tests.

XII. CONCLUSION

In the present paper we analyzed the effectiveness of teaching spreadsheeting with Sprego programming (Spreadsheet Lego) compared to traditional, widely accepted user-friendly, but low-mathability methods.

The results clearly demonstrate that the traditional tool-centered methods do not build up long-lasting knowledge and do not help students develop their problem-solving abilities. On the other hand, Sprego proved more effective both in problem-solving and extending abstraction. Beyond this, SPREGO methodology provides students with long lasting knowledge, which is a great advantage of the method compared to traditional surface-navigation approaches.

This result shows that the directed inquiry-based methodology applied in the experimental group, in spite of the low number of contact lessons, was enough to build up the algorithm of the conditional calculation. However, using the same methodology, the time was not enough to build up the algorithm of the linear search. In the experimental group, a slight increase is detectable, while in the control group there is a steep decrease. We can conclude that the method was also more effective in the case of this algorithm, but building up numerous algorithm(s) in this short period is impossible. Algorithms require more time (Hypothesis 4) and direct, strong instructional guidance.

In the population density task, the missing knowledge-transfer items deriving from background studies caused a problem, and there was no time to fill in this gap. This task clearly exemplifies that real-world problem-solving should be transferred to other disciplines, where computers are used as tools, and are not the aim of the teaching-learning process.

However, we must keep in mind that developing fundamental skills, including computational thinking skills and abilities, is a time-consuming process. Based on the results of the project, it was also found that the eight-hour period assigned to the subject by the curriculum is not sufficient to cover the subject of spreadsheeting and functional programming. This finding leads to further consequences in terms of creating curricula on the subject of data management and programming, as well as integrating them.

Based on the results of previous research and the results of our project described in the present paper, it is clear that some of the misconceptions circulating in computer science education have held back students' chances of developing their computational thinking skills and their effectiveness in real-world computer problem-solving. Our results prove that there are methods and approaches which can supersede the less effective methods and aid our students, and Sprego is among them.

REFERENCES

[1] R. Descartes and J. Lessing Rosenwald Collection. *Discours de la Methode Pour Bien Conduire sa Raison, & Chercher la Verité Dans Les Sciences. Plus La Dioptriqve. Les Meteores. Et La Geometrie. Qui Sont des Essais de Cete Methode*. A Leyde, De l'imprimerie de I. Maire. Accessed: Jan. 30, 2021. [Online]. Available: https://www.loc.gov/item/32034972/

[2] M. Csernoch, "Programming with spreadsheet functions: Sprego," in Hungarian, Programozás Táblázatkezelő Függvényekkel—Sprego, Műszaki Könyvkiadó, Budapest, Hungary, 2014.

[3] M. Csernoch and P. Biró. (2015). *Sprego Programming*. Spreadsheets in Education (eJSiE). Accessed: Jan. 30, 2021. [Online]. Available: https://sie.scholasticahq.com/article/4638-sprego-programming

[4] M. Csernoch and P. Biró, "Sprego programming," LAP Lambert Academic Publishing, OmniScriptum, Gmbh & Co. KG, Saarbrücken, Germany, Tech. Rep., 2015.

[5] J. M. Wing, "Computational thinking," *Commun. ACM*, vol. 49, no. 3, pp. 33–35, 2006, doi: 10.1145/1118178.1118215.

[6] E. Soloway, "Should we teach students to program?" *Commun. ACM*, vol. 36, no. 10, pp. 21–24, Oct. 1993, doi: 10.1145/163430.164061.

[7] M. Ben-Ari, "Non-myths about programming," in *Proc. 6th Int. Workshop Comput. Educ. Res. (ICER)*, 2010, pp. 1–2, doi: 10.1145/1839594.1839595.

[8] T. A. Grossman, V. Mehrotra, and Ö. Özlük, "Lessons from mission-critical spreadsheets," *Commun. Assoc. Inf. Syst.*, vol. 20, no. 1, pp. 1009–1042, 2007.

[9] J. K. Ousterhout, "Scripting: Higher level programming for the 21st century," *Computer*, vol. 31, no. 3, pp. 23–30, Mar. 1998, doi: 10.1109/2.660187.

[10] M. Burnett, J. Atwood, R. Walpole Djang, J. Reichwein, H. Gottfried, and S. Yang, "Forms/3: A first-order visual language to explore the boundaries of the spreadsheet paradigm," *J. Funct. Program.*, vol. 11, no. 2, pp. 155–206, Mar. 2001, doi: 10.1017/S0956796800003828.

[11] A. G. Yoder and D. L. Cohn, "Real spreadsheets for real programmers," in *Proc. IEEE Int. Conf. Comput. Lang. (ICCL)*, May 1994, pp. 20–30, doi: 10.1109/ICCL.1994.288396.

[12] D. Wakeling, "Spreadsheet functional programming," *J. Funct. Program.*, vol. 17, no. 1, pp. 131–143, Jan. 2007, doi: 10.1017/S0956796806006186.

[13] ICAEW THOUGHT LEADERSHIP 2016. Spreadsheet Competency Framework. *A Structure for Classifying Spreadsheet Ability in Finance Professionals*. Accessed: Jan. 28, 2021. [Online]. Available: https://www.icaew.com/-/media/corporate/files/technical/information-technology/it-faculty/spreadsheet-competency-framework.ashx

[14] C. Chambers and C. Scaffidi, "Struggling to excel: A field study of challenges faced by spreadsheet users," in *Proc. IEEE Symp. Vis. Lang. Hum.-Centric Comput.*, Sep. 2010, pp. 187–194, doi: 10.1109/VLHCC.2010.33.

[15] S. E. Kruck, J. J. Maher, and R. Barkhi, "Framework for cognitive skill acquisition and spreadsheet training," *J. Organizational End User Comput.*, vol. 15, no. 1, pp. 20–37, Jan. 2003.

[16] P. Baranyi and A. Gilanyi, "Mathability: Emulating and enhancing human mathematical capabilities," in *Proc. IEEE 4th Int. Conf. Cognit. Infocommunications (CogInfoCom)*, Dec. 2013, pp. 555–558, doi: 10.1109/CogInfoCom.2013.6719309.

[17] G. Csapó, M. Csernoch, and K. Abari, "Sprego: Case study on the effectiveness of teaching spreadsheet management with schema construction," *Educ. Inf. Technol.*, vol. 25, no. 3, pp. 1585–1605, May 2020.

[18] G. Csapó, K. Sebestyén, M. Csernoch, and K. Abari, "Case study: Developing long-term knowledge with sprego," *Educ. Inf. Technol.*, vol. 26, no. 1, pp. 965–982, Jan. 2021, doi: 10.1007/s10639-020-10295-0.

[19] Eusprig 2021. *Horror Stories. European Spreadsheet Risks Interest Group*. Accessed: Jan. 30, 2021. [Online]. Available: https://www.eusprig.org/horror-stories.htm

[20] R. R. Panko, "What we know about spreadsheet errors," *J. Organizational End User Comput.*, vol. 10, no. 2, pp. 15–21, Apr. 1998, doi: 10.4018/joeuc.1998040102.

[21] R. Abraham, M. Burnett, and M. Erwig, "Spreadsheet programming," in *Wiley Encyclopedia of Computer Science and Engineering*. Hoboken, NJ, USA: Wiley, 2009, doi: 10.1002/9780470050118.ecse415.

[22] ICAEW THOUGHT LEADERSHIP. (2018). *Twenty Principles for Good Spreadsheet Practice*. Third Edition. Accessed: Jan. 28, 2021. [Online]. Available: https://www.icaew.com/-/media/corporate/files/technical/technology/excel-community/20-principles-of-good-spreadsheet-practice-2018.ashx

[23] T. Antoniu, P. A. Steckler, S. Krishnamurthi, E. Neuwirth, and M. Felleisen, "Validating the unit correctness of spreadsheet programs," in *Proc. 26th Int. Conf. Softw. Eng.*, May 2004, pp. 439–448, doi: 10.1109/ICSE.2004.1317466.

[24] R. R. Panko and R. P. Halverson, "Spreadsheets on trial: A survey of research on spreadsheet risks," in *Proc. 29th Hawaii Int. Conf. Syst. Sci. (HICSS)*, 1996, pp. 326–335, doi: 10.1109/HICSS.1996.495416.

[25] F. Hermans, M. Pinzger, and A. van Deursen, "Detecting code smells in spreadsheet formulas," in *Proc. 28th IEEE Int. Conf. Softw. Maintenance (ICSM)*, Sep. 2012, pp. 409–418, doi: 10.1109/ICSM.2012.6405300.

[26] F. Hermans, B. Sedee, M. Pinzger, and A. van Deursen, "Data clone detection and visualization in spreadsheets," in *Proc. 35th Int. Conf. Softw. Eng. (ICSE)*, May 2013, pp. 292–301, doi: 10.1109/ICSE.2013.6606575.

[27] A. Azam, K. A. Alam, and A. Umair, "Spreadsheet smells: A systematic mapping study," in *Proc. Int. Conf. Frontiers Inf. Technol. (FIT)*, Dec. 2019, pp. 345–3455, doi: 10.1109/FIT47737.2019.00071.

[28] N. Garrett, "Textbooks for responsible data analysis in excel," *J. Educ. Bus.*, vol. 90, no. 4, pp. 169–174, May 2015, doi: 10.1080/08832323.2015.1007908.

[29] F. Hermans and E. Murphy-Hill, "Enron's spreadsheets and related emails: A dataset and analysis," in *Proc. IEEE/ACM 37th IEEE Int. Conf. Softw. Eng.*, May 2015, pp. 7–16, doi: 10.1109/ICSE.2015.129.

[30] M. Burnett, C. Cook, O. Pendse, G. Rothermel, J. Summet, and C. Wallace, "End-user software engineering with assertions in the spreadsheet paradigm," in *Proc. 25th Int. Conf. Softw. Eng.*, 2003, pp. 93–103, doi: 10.1109/ICSE.2003.1201191.

[31] S. Badame and D. Dig, "Refactoring meets spreadsheet formulas," in *Proc. 28th IEEE Int. Conf. Softw. Maintenance (ICSM)*, Sep. 2012, pp. 399–409, doi: 10.1109/ICSM.2012.6405299.

[32] J. Cunha, J. Saraiva, and J. Visser, "Discovery-based edit assistance for spreadsheets," in *Proc. IEEE Symp. Vis. Lang. Human-Centric Comput. (VL/HCC)*, Sep. 2009, pp. 233–237, doi: 10.1109/VLHCC.2009.5295255.

[33] M. Burnett, A. Agrawal, and P. van Zee, "Exception handling in the spreadsheet paradigm," *IEEE Trans. Softw. Eng.*, vol. 26, no. 10, pp. 923–942, Oct. 2000, doi: 10.1109/32.879817.

[34] G. Miller and F. Hermans, "Gradual structuring in the spreadsheet paradigm," in *Proc. IEEE Symp. Vis. Lang. Hum.-Centric Comput. (VL/HCC)*, Sep. 2016, pp. 240–241, doi: 10.1109/VLHCC.2016.7739696.

[35] F. Hermans, B. Jansen, S. Roy, E. Aivaloglou, A. Swidan, and D. Hoepelman, "Spreadsheets are code: An overview of software engineering approaches applied to spreadsheets," in *Proc. IEEE 23rd Int. Conf. Softw. Anal., Evol., Reengineering (SANER)*, Mar. 2016, pp. 56–65, doi: 10.1109/SANER.2016.86.

[36] C. Scaffidi, M. Shaw, and B. Myers, "Estimating the numbers of end users and end user programmers," in *Proc. IEEE Symp. Vis. Lang. Hum.-Centric Comput. (VL/HCC)*, Sep. 2005, pp. 207–214, doi: 10.1109/VLHCC.2005.34.

[37] G. Futschek, "Algorithmic thinking: The key for understanding computer science," *Informatics Education—The Bridge between Using and Understanding Computers*. Berlin, Germany: Springer, 2006, pp. 159–168, doi: 10.1007/11915355_15.

[38] S. Y. Lye and J. H. L. Koh, "Review on teaching and learning of computational thinking through programming: What is next for K-12?" *Comput. Hum. Behav.*, vol. 41, pp. 51–61, Dec. 2014.

[39] V. Aleksić and M. Ivanović, "Introductory programming subject in European higher education," *Informat. Educ.*, vol. 15, no. 2, pp. 163–182, Oct. 2016.

[40] R. R. Panko, "The cognitive science of spreadsheet errors: Why thinking is bad," in *Proc. 46th Hawaii Int. Conf. Syst. Sci.*, Maui, HI, USA, Jan. 2013, pp. 4013–4022.

[41] D. Kadijevich, "Learning about spreadsheet," in *Improving Computer Science Education*, D. Kadijevich, C. Angeli, and C. Schulte, Eds. New York, NY, USA: : Routledge, 2013. pp. 19–33.

[42] M. Campbell-Kelly, "Number crunching without programming: The evolution of spreadsheet usability," *IEEE Ann. History Comput.*, vol. 29, no. 3, pp. 6–19, Jul. 2007, doi: 10.1109/MAHC.2007.4338438.

[43] K.-C. Yeh, Y. Xie, and F. Ke, "Teaching computational thinking to non-computing majors using spreadsheet functions," in *Proc. Frontiers Educ. Conf. (FIE)*, Oct. 2011, pp. 1–5, doi: 10.1109/FIE.2011.6142980.

[44] M. Gove. (Jan. 13, 2012). *Michael Gove speech at the BETT Show 2012*. Accessed: Jan. 30, 2021. [Online]. Available: https://www.gov.U.K./government/speeches/michael-govespeech-at-the-bett-show-2012

[45] M. Gove. (Jan. 22, 2014). *Michael Gove Speaks About Computing and Education Technology*. Accessed: Jan. 30, 2021. [Online]. Available: https://www.gov.U.K./government/speeches/michael-gove-speaks-aboutcomputing-and-education-technology

[46] T. Bell and H. Newton, "Unplugging computer science," in *Improving Computer Science Education*, D. M. Kadijevich, C. Angeli, and C. Schulte, Eds. Oxfordshire, U.K.: Routledge, 2013.

[47] K. Freiermuth, J. Hromkovič, and B. Steffen, "Creating and testing textbooks for secondary schools," in *Informatics Education—Supporting Computational Thinking* (Lecture Notes in Computer Science). Berlin, Germany: Springer, 2008, pp. 216–228, doi: 10.1007/978-3-540-69924-8_20.

[48] C. Angeli, "Teaching spreadsheets: A TPCK perspective," in *Improving Computer Science Education*, D. M. Kadijevich, C. Angeli, and C. Schulte, Eds. Oxfordshire, U.K.: Routledge, 2013.

[49] P. Papp and M. Csernoch. *Spreadsheeting is Problem Solving?*. Hungarian: A Táblázatkezelés is Problémamegoldás? Infó Éra 2018. Accessed: Jan. 30, 2021. [Online]. Available: https://people.inf.elte.hu/szlavi/InfoDidact18/Infodidact2018.pdf?fbclid=IwAR38Vk3h2w_81Iv61C76V6xkErpLdxX4Ubc96P4VuR4EXYmIFW0b5-Jj0Z4

[50] M. Csernoch, "Thinking fast and slow in computer problem solving," *J. Softw. Eng. Appl.*, vol. 10, no. 1, pp. 11–40, 2017.

[51] *The Math(s) Fix: An Education Blueprint for the AI Age*, C. Wolfram, Wolfram Media, Inc, MB, Canada, 2020.

[52] T. Rattenbury, J. M. Hellerstein, J. Heer, S. Kandel, and C. Carreras, *Principles of Data Wrangling: Practical Techniques for Data Preparation*. Newton, MA, USA: O'Reilly Media, 2017.

[53] F. Hermans. (Sep. 13, 2019). *Strange Loop 2019—How to Teach Programming (and Other Things)?*. Sara McCombs. Accessed: Jan. 30, 2021. [Online]. Available: https://about.sourcegraph.com/strange-loop/strange-loop-2019-how-to-teach-programming-and-other-things/

[54] A. Swidan and F. Hermans, "The effect of reading code aloud on comprehension," in *Proc. ACM Conf. Global Comput. Educ.*, May 2019, pp. 178–184, doi: 10.1145/3300115.3309504.

[55] A. Sarkar, J. W. Borghouts, A. Iyer, S. Khullar, C. Canton, F. Hermans, A. D. Gordon, and J. Williams, "Spreadsheet use and programming experience: An exploratory survey," in *Proc. Extended Abstr. CHI Conf. Hum. Factors Comput. Syst.*, Apr. 2020, pp. 1–9, doi: 10.1145/3334480.3382807.

[56] R. Lister, "After the gold rush: Toward sustainable scholarship in computing," in *Proc. 10th Conf. Australas. Comput. Educ. (ACE)*, vol. 78, 2008, pp. 3–17.

[57] ECDL 2020. *ECDL U.K. Course Syllabus. European Computer Driving License in EU and US*. Accessed: Jan. 30, 2021. [Online]. Available: https://www.ecdluk.co.U.K./documents/ECDL%20Syllabus.pdf

[58] G. Pólya, *How To Solve It: A New Aspect of Mathematical Method*, 2nd ed. Princeton, NJ, USA: Princeton Univ. Press, 1957.
[59] G. Pólya, *Mathematical Discovery. on Understanding, Learning, and Teaching Problem Solving*. New York, NY, USA: Wiley, 1981.
[60] O. Müller, ''Pattern oriented instruction and the enhancement of analogical reasoning,'' in *Proc. Int. workshop Comput. Educ. Res. (CER)*, 2005, pp. 57–67, doi: 10.1145/1089786.1089792.
[61] O. Müller, D. Ginat, and B. Haberman, ''Pattern-oriented instruction and its influence on problem decomposition and solution construction,'' *ACM SIGCSE Bull.*, vol. 39, no. 3, pp. 151–155, Jun. 2007, doi: 10.1145/1269900.1268830.
[62] D. Kahneman, *Thinking, Fast and Slow*. New York, NY, USA: Farrar, Straus; Giroux, 2011.
[63] J. Sweller, P. Ayres, and S. Kalyuga, *Cognitive Load Theory*. Berlin, Germany: Springer, 2011.
[64] M. Roter, ''Managing people for improvement,'' in *Adaptiveness and Superior Results*. New York, NY, USA: McGraw-Hill, 2009.
[65] H. Tambunan, ''The effectiveness of the problem solving strategy and the scientific approach to Students' mathematical capabilities in high order thinking skills,'' *Int. Electron. J. Math. Educ.*, vol. 14, no. 2, pp. 293–302, Feb. 2019, doi: 10.29333/iejme/5715.
[66] P. Mishra and M. J. Koehler, ''Technological pedagogical content knowledge: A framework for teacher knowledge,'' *Teachers College Rec.*, vol. 108, no. 6, pp. 1017–1054, 2006.
[67] C. Angeli and N. Valanides, *Technological Pedagogical Content Knowledge: Exploring, Developing, and Assessing TPCK*. New York, NY, USA: Springer, 2015.
[68] *Best Practice Awards 2015*, Train the Trainer, ECDL Foundation, Dublin, Ireland, 2015, pp. 36–38.
[69] J. Hattie, *Visible Learning for Teachers: Maximizing Impact on Learning*. Oxfordshire, U.K.:Routledge, 2012.
[70] M. Ben-Ari, ''Bricolage forever!'' in *Proc. 11st Annu. Workshop (PPIG)*, Leeds, U.K., Jan. 1999, pp. 1–5. Accessed: Jul. 21, 2015. [Online]. Available: http://www.ppig.org/papers/11st-benari.pdf
[71] M. Ben-Ari and T. Yeshno, ''Conceptual models of software artifacts,'' *Interacting Comput.*, vol. 18, no. 6, pp. 1336–1350, Dec. 2006, doi: 10.1016/j.intcom.2006.03.005.
[72] M. Csernoch, ''Do you speak and write in informatics?'' in *Proc. 10th Int. Multi-Conf. Complex., Inform. Cybern. (IMCIC)*, 2019, pp. 147–152. Accessed: Jan. 15, 2021. [Online]. Available: https://www.iiis.org/CDs2019/CD2019Spring/papers/ZA216TN.pdf
[73] M. Csernoch, P. Biró, J. Máth, and K. Abari, ''Testing algorithmic skills in traditional and non-traditional programming environments,'' *Informat. Educ.*, vol. 14, no. 2, pp. 175–197, Oct. 2015.
[74] M. Prensky. (Oct. 2001). *Digital Natives, Digital Immigrants*. From on the Horizon MCB University Press. Accessed: Mar. 21, 2018. [Online]. Available: https://www.marcprensky.com/writing/Prensky%20-%20Digital%20Natives,%20Digital%20Immigrants%20-%20ParPRET.pdf
[75] P. A. Kirschner, J. Sweller, and R. E. Clark, ''Why minimal guidance during instruction does not work: An analysis of the failure of constructivist, discovery, problem-based, experiential, and inquiry-based teaching,'' *Educ. Psychol.*, vol. 41, no. 2, pp. 75–86, Jun. 2006.
[76] P. A. Kirschner and P. De Bruyckere, ''The myths of the digital native and the multitasker,'' *Teaching Teacher Educ.*, vol. 67, pp. 135–142, Oct. 2017.
[77] P. Biro and M. Csernoch, ''The mathability of computer problem solving approaches,'' in *Proc. 6th IEEE Int. Conf. Cognit. Infocommunications (CogInfoCom)*, Oct. 2015, pp. 111–114, doi: 10.1109/CogInfoCom.2015.7390574.
[78] P. Biro and M. Csernoch, ''The mathability of spreadsheet tools,'' in *Proc. 6th IEEE Int. Conf. Cognit. Infocommunications (CogInfoCom)*, Oct. 2015, pp. 105–110, doi: 10.1109/CogInfoCom.2015.7390573.
[79] J. Case and R. Gunstone, ''Metacognitive development as a shift in approach to learning: An in-depth study,'' *Stud. Higher Educ.*, vol. 27, no. 4, pp. 459–470, Oct. 2002.
[80] S. Booth, *Learning to Program: A Phenomenographic Perspective*. Gothenburg, Sweden: Acta Universitatis Gothoburgensis, 1992.
[81] M. Csernoch and P. Biró, ''Computer Problem-solving,'' *Hungarian, Számítógépes Problémamegoldás*, vol. 62, no. 3, pp. 86–94, 2015.
[82] C. Wolfram. (2015). *Evidence: Let's Promote not Stifle Innovation in Education*. Accessed: Oct. 12, 2015. [Online]. Available: https://www.conradwolfram.com/home/2015/5/21/role-of-evidence-in-education-innovation
[83] J. Cao, S. D. Fleming, M. Burnett, and C. Scaffidi, ''Idea garden: Situated support for problem solving by end-user programmers,'' *Interacting Comput.*, vol. 27, no. 6, pp. 640–660, Nov. 2015, doi: 10.1093/iwc/iwu022.
[84] J. Carroll and C. Rosson, ''The paradox of the active user,'' in *Interfacing Thought: Cognitive Aspects of Human-Computer Interaction*, J. M. Carroll, Ed. Cambridge, MA, USA: MIT Press, 1998, pp. 80–111.
[85] M. Csernoch and P. Biró, ''Edu-edition spreadsheet competency framework,'' in *Proc. Conf. Spreadsheet Risk Manage. (EuSpRIG)*, London, U.K., 2018, pp. 121–136.
[86] P. Hilton and F. G. Pólya, *Mathematical Discovery. On understanding, learning, and teaching problem solving*. New York, NY, USA: Wiley, 1981.
[87] J. A. Chen, D. B. Morris, and N. Mansour, ''Science teachers' beliefs. Perceptions of efficacy and the nature of scientific knowledge and knowing,'' in *International Handbook of Research on Teachers' Beliefs*, H. Fives and M. G. Gill, Eds. Oxfordshire, UK.: Routledge, 2015, pp. 370–386.
[88] *PISA 2009 Results: Student on Line: Digital Technologies and Performance*, OECD, Paris, France, 2011.
[89] *Main Results From the PISA 2012 Computer-Based Assessments, in Students, Computers and Learning: Making the Connection*, OECD Publishing, Paris, France, 2015.
[90] OECD. *Education at a Glance 2016*. OECD Indicators (Summary Hungarian), Hungarian: OECD Mutatók. (Összefoglalás magyarul), Oktatási Körkép 2016. Accessed: May 17, 2018. [Online]. Available: https://www.keepeek.com/Digital-Asset-Management/oecd/education/education-at-a-glance-2016/summary/hungarian_24bbf13e-hu#page2
[91] O. Hatamleh and G. Tilesch, *Between Brains*. Milton Keynes, U.K.: Lightning Source UK Ltd., 2020.
[92] S. Krishnamurthi and K. Fisler, ''Programming paradigms and beyond,'' in *The Cambridge Handbook of Computing Education Research* (Cambridge Handbooks in Psychology), S. Fincher and A. Robins, Eds. Cambridge, U.K.: Cambridge Univ. Press, 2019, pp. 377–413, doi: 10.1017/9781108654555.014.
[93] K. D. Lee, *Foundations of Programming Languages*, 2nd ed. Cham, Switzerland: Springer, 2017.
[94] G. Szanyi, ''The investigation of students' skills in the process of function concept creation,'' *Teaching Math. Comput. Sci.*, vol. 13, no. 2, pp. 249–266, 2015.
[95] C. Wolfram. *Stop Teaching Calculating, Start Teaching Math-Fundamentally Reforming the Math Curriculum*. Transcript: Wolfram Technology Conference Talk. TED Global. Accessed: Oct. 12, 2018. [Online]. Available: https://www.computerbasedmath.org/resources/Education_talk_transcript.pdf
[96] G. Lovászová and J. Hvorecký, ''On programming and spreadsheet calculations,'' *Spreadsheets Educ.*, vol. 1, no. 1, pp. 44–51, 2003. Accessed: Jan. 25, 2016. [Online]. Available: https://epublications.bond.edu.au/ejsie/vol1/iss1/3
[97] M. Schneider, ''An empirical study of introductory lectures in informatics based on fundamental concepts,'' in *Informatics and Students Assessment* (Lecture Notes in Informatics), vol. 1. Germany: GI-Edition, Schloss Dagstuhl, 2004, pp. 123–133.
[98] M. Schneider, ''A strategy to introduce functional data modeling at school informatics,'' in *From Computer Literacy to Informatics Fundamentals*, R. T. Mittermeir, Ed. 2005, pp. 130–144. Berlin, Germany: Springer, doi: 10.1007/978-3-540-31958-0_16.
[99] P. Hubwieser, ''Functional modelling in secondary schools using spreadsheets,'' *Educ. Inf. Technol.*, vol. 9, no. 2, pp. 175–183, Jun. 2004, doi: 10.1023/B:EAIT.0000027929.91773.ab.
[100] P. Warren, ''Learning to program: Spreadsheets, scripting and HCI,'' in *Proc. 6th Australas. Conf. Comput. Educ.*, vol. 30, 2004, pp. 327–333. Darlinghurst, NSW, Australia: Australian Computer Society. Accessed on: Jan. 25, 2018. [Online]. Available: https://dl.acm.org/citation.cfm?id=979968.980012.
[101] P. Sestoft, ''Spreadsheet technology,'' IT Univ. Copenhagen, Copenhagen, Denmark, Tech. Rep. ITU-TR-2011-142, 2011.
[102] R. Skemp, *The Psychology of Learning Mathematics*. Mahwah, NJ, USA: Lawrence Erlbaum Associatives, 1971.
[103] K. Sebestyén, G. Csapó, and M. Csernoch. *Introduction to Algorithmic_Based Data Management in Spreadsheet Environment*. The Turkish Online Journal of Educational Technology. INTE 2019. Accessed: Jan. 25, 2020. [Online]. Available: https://www.int-e.net/publication_folder/inte/inte_iticam_2019.pdf

[104] K. Sebestyén and G. Csapó. *Visualising Sprego Inequality Problems With 2D Representations*. Turkish Online Journal of Educational Technology. INTE 2018. Accessed: Jan. 25, 2020. [Online]. Available: https://www.int-e.net/publication_folder/inte/inte_iticam_idec2018_v2.pdf

[105] Á. Gulácsi and N. Dienes. *3D Software Environment for Educational Sprego Programming*. The Turkish Online Journal of Educational Technology, INTE 2018. Accessed: Jan. 25, 2020. [Online]. Available: https://www.tojet.net/special/2018_12_3.pdf

[106] Á. Gulácsi, N. Dienes and M. Csernoch, "Sprego toolbox: A way to teach spreadsheeting meaningfully," in *Turkish Online J. Educ. Technol.*, vol. 2, pp. 296–302, Dec. 2019.

[107] G. Csapo, "Sprego virtual collaboration space," in *Proc. 8th IEEE Int. Conf. Cognit. Infocommunications (CogInfoCom)*, Sep. 2017, pp. 000137–000142, doi: 10.1109/CogInfoCom.2017.8268230.

[108] G. Csapo, "Sprego virtual collaboration space: Improvement guidelines for the MaxWhere seminar system," in *Proc. 8th IEEE Int. Conf. Cognit. Infocommunications (CogInfoCom)*, Sep. 2017, pp. 000143–000144, doi: 10.1109/CogInfoCom.2017.8268231.

[109] C. Wilcox and J. Walkenbach. (2003). *Guidelines and Examples of Array Formulas*. Accessed: Dec. 13, 2018. [Online]. Available: https://support.office.com/en-us/article/Guidelines-and-examples-of-array-formulas-3be0c791-3f89-4644-a062-8e6e9ecee523?CorrelationId=643cf62a-061f-461e-8008-d601efbad369&ui=en-U.S.&rs=en-U.S.&ad=U.S

[110] J. Walkenbach, "Excel 2010 Bible," *Accessed on: December*, vol. 13, 2018. [Online]. Available: http://www.seu.ac.lk/cedpl/student

[111] R. R. Panko, "What we don't know about spreadsheet errors today: The facts, why we don't believe them, and what we need to do," in *Proc. EuSpRIG*, London, U.K., Jul. 2015, pp. 1–15.

[112] P. Biro and M. Csernoch, "Deep and surface metacognitive processes in non-traditional programming tasks," in *Proc. 5th IEEE Conf. Cognit. Infocommunications (CogInfoCom)*, Nov. 2014, pp. 49–54, doi: 10.1109/CogInfoCom.2014.7020507.

[113] P. Biró, M. Csernoch, K. Abari, and J. Máth. (2015). *Testing Algorithmic and Application Skills*. Turkish Online Journal of Educational Technology. Accessed: May 15, 2020. [Online]. Available: https://www.tojet.net/special/2015_8_1.pdf

[114] P. Biró and M. Csernoch, "Maths problems in pseudo-codes compared to computer usage," in *Proc. Int. Conf. Educ. New Developments*, M. Carmo, Ed. Lisboa, Portugal: InScience Press, 2018, pp. 341–346.

[115] M. Csernoch and P. Biró, "Are digital natives spreadsheet natives?" 2019, *arXiv:1909.00865*.

[116] M. Csernoch, "From webtables to datatables," 2020, *arXiv:2006.14694*.

[117] Microsoft. *Excel Functions (Alphabetical)*. Excel for Microsoft 365 Excel for Microsoft 365 for Mac Excel for the web Excel 2021 Excel 2021 for Mac Excel 2019 Excel 2019 for Mac Excel 2016 Excel 2016 for Mac Excel 2013 Excel Web App Excel 2010 Excel 2007 Excel for Mac 2011 Excel Starter 2010. Accessed: Sep. 23, 2021. [Online]. Available: https://support.microsoft.com/en-us/office/excel-functions-alphabetical-b3944572-255d-4efb-bb96-c6d90033e188

[118] SLE 2020. *School Leaving Exams*. Hungarian: Érettségi. Accessed: May 17, 2020. [Online]. Available: https://www.oktatas.hu/kozneveles/erettsegi/jogszabalyok

[119] SLE in Informatics 2020. *School Leaving Exams in Informatics*. Hungarian: Informatika Érettségi. Accessed: May 17, 2020. [Online]. Available: https://www.oktatas.hu/pub_bin/dload/kozoktatas/erettsegi/vizsgakovetelmenyek2017/informatika_vk_2017.pdf

[120] M. Csernoch and P. Biro, "First year students' attitude to computer problem solving," in *Proc. 8th IEEE Int. Conf. Cognit. Infocommunications (CogInfoCom)*, Sep. 2017, pp. 225–230, doi: 10.1109/CogInfoCom.2017.8268247.

[121] K. Pintér. (2012). *On Teaching Mathematical Problem-Solving and Problem Posing. PhD thesis*. Doctoral School in Mathematics and Computer Science University of Szeged Faculty of Science and Informatics Bolyai Institute. Accessed: May 29, 2020. [Online]. Available: https://www.math.u-szeged.hu/phd/dreposit/phdtheses/pinter-klara-a.pdf

[122] K. Chmielewska and D. Matuszak, "Mathability and coaching," in *Proc. 8th IEEE Int. Conf. Cognit. Infocommunications (CogInfoCom)*, Sep. 2017, pp. 000427–000432, doi: 10.1109/CogInfoCom.2017.8268284.

[123] R. E. Mayer, "The psychology of how novices learn computer programming," *ACM Comput. Surveys*, vol. 13, no. 1, pp. 121–141, Mar. 1981, doi: 10.1145/356835.356841.

[124] J. B. Biggs and K. E. Collis, *Evaluating the Quality of Learning: The SOLO Taxonomy*. New York, NY, USA: Academic, 1982.

[125] R. Lister, B. Simon, E. Thompson, J. L. Whalley, and C. Prasad, "Not seeing the forest for the trees: Novice programmers and the SOLO taxonomy," in *Proc. 11st Annu. SIGCSE Conf. Innov. Technol. Comput. Sci. Educ.*, New York, NY, USA, 2006, pp. 118–122.

[126] J. Sheard, A. Carbone, R. Lister, B. Simon, E. Thompson, and J. L. Whalley, "Going SOLO to assess novice programmers," *ACM SIGCSE Bull.*, vol. 40, no. 3, pp. 209–213, Aug. 2008.

[127] L. E. Klopfer, A. B. Champagne, and R. F. Gunstone, "Naive knowledge and science learning," *Res. Sci. Technol. Educ.*, vol. 1, no. 2, pp. 173–183, Jan. 1983.

[128] M. Moyo, A. C. Tiba, and K. Madzima. (2016). *Impact of Pre-Service Teachers' Prior Knowledge of Information Technologies on Perceptions and Beliefs on Computers in Education Modules*. Accessed: Jun. 15, 2019. [Online]. Available: https://uir.unisa.ac.za/bitstream/handle/10500/22888/Moses%20Moyo%2C%20Anyen%20Chantylee%20Tiba%2C%20Kudakwashe%20Madzima.pdf?sequence=1

MÁRIA CSERNOCH was born in Szentes, Hungary, in 1963. She received the teacher degree in mathematics—descriptive geometry, the B.Sc. degree in software engineering, in 1986, the teacher degree in informatics, in 1997, the degree in English, in 2000, the Ph.D. degree in mathematics and computer sciences, in 2006, and the Dr. habil. degree in applied linguistics from the University of Debrecen, Hungary, in 2012.

She currently works as an Associate Professor at the Faculty of Informatics, University of Debrecen. Her research interests include didactics of Informatics—specialized in developing algorithmic skills, computational thinking, and teaching programming languages, computational linguistics, and computer aided language teaching and learning.

PIROSKA BIRÓ was born in Ditro, Romania, in 1983. She received the M.Sc. degree in computational mathematics from the Faculty of Mathematics-Informatics, University of Babes-Bolyai, and the Ph.D. degree in informatics from the University of Debrecen, Hungary, in 2015.

She currently works as a Senior Lecturer at the University of Debrecen and Sapientia Hungarian University of Transylvania. Her research interests include didactics of informatics, algorithmic thinking, high level programming languages, and computer aided education.

JÁNOS MÁTH was born in Cegléd, Hungary, in 1959. He received the degree in mathematics and the Ph.D. degree from the University of Debrecen, Hungary, in 1984 and 1997, respectively.

He currently works as an Associate Professor at the Faculty of Arts and Humanities, University of Debrecen. His research interests include statistics, knowledge space theory, and cognitive load theory.