IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 17 November 2022, accepted 16 December 2022, date of publication 20 December 2022,
date of current version 27 December 2022.

*Digital Object Identifier 10.1109/ACCESS.2022.3230981*

RESEARCH ARTICLE

# Do Structured Flowcharts Outperform Pseudocode? Evidence From Eye Movements

**MAGDALENA ANDRZEJEWSKA**[1] **AND ANNA STOLIŃSKA**[2]
[1]Institute of Security and Computer Science, Pedagogical University of Krakow, 30-084 Krakow, Poland
[2]College of Economics and Computer Science of Krakow, 31-150 Krakow, Poland

Corresponding author: Magdalena Andrzejewska (magdalena.andrzejewska@up.krakow.pl)

This work was supported by the Scientific Research Fund of the Pedagogical University of Krakow.

**ABSTRACT** Computational thinking is a key universal competence, often taught using methods specific to computer science. One step towards achieving it is learning to analyse and create algorithms. Researchers have long been trying to establish how the form of representation of algorithms (pseudocode versus flowchart) affects its understanding and have reached varying, sometimes conflicting results. This article presents findings that provide objective new data on this topic. In our experiment, we used two different types of algorithmic tasks with three levels of complexity and a group of 114 research participants with varying programming skills. In addition, we used an eye tracking technique that allowed us to collect detailed information about the subjects' attention distribution during analysis of algorithms. Our results show that subjects took significantly less time to analyse flowcharts (than they did with pseudocode), made much fewer errors, and had higher confidence in the correctness of their solution. Based on eye tracking data, a reduced number of both re-analyses of the algorithm and input data re-referencing was observed for graphically presented tasks. The difference in favour of flowcharts was revealed (with few exceptions) for all levels of algorithm complexity (simple, medium, complex), while regarding the duration of analysis the advantage of flowcharts increased with the growing complexity of algorithms. For complex algorithms, a significant relationship was observed between algorithm presentation and level of programming skills versus the duration of task solving and confidence level. Our study strongly supports the idea of using graphic representation of algorithms both when learning to code and in acquiring computational thinking skills.

**INDEX TERMS** Coding, computational thinking, eye tracking, flowchart, pseudocode, solving algorithmic problems.

## I. INTRODUCTION

Due to technological and civilisational progress and the resulting emergence of new professions, an ever-greater attention is being paid to developing universal competencies that can improve creativity in finding solutions to problems from different area s of life. Computational thinking (CT) is considered one such competence [1], [2]. A growing interest in CT education in schools has made it an integral part of curricula in many countries around the world [3], [4]. Attempts have been made to incorporate CT education in the curricula of various courses but, since computer thinking requires adherence to certain rules inherent in algorithmics [5], it is becoming very obvious that one should use coding lessons to teach it.

Programming skills are widely considered to be among the most desirable yet difficult to master [6]. Cabo [7] believes that first-year programming courses at universities are still characterised by high drop-out and failure rates. It is therefore recommended that teachers choose easy problems in the first phase of the learning process to provide appropriate guidance for their students so that they gain experience through

The associate editor coordinating the review of this manuscript and approving it for publication was Rebecca Strachan.

learning correct behaviour patterns [8]. Many studies have shown that it is not only creating new algorithms, but also analysing readymade solutions and implementing them in a programming language, that are difficult tasks for students [9]. It has also been observed that students encounter problems related to the correct interpretation of the syntax of the programming language; this is often a practical obstacle to solving even simple tasks [10].

One important question is which teaching strategies related to algorithms, code or flowchart, are more effective? Does the integration of these two forms of algorithm affect learning to program, and if so, how? The research in [10] found that the vast majority of students write algorithms successfully using flowcharts, and that using a programming language distracts them from the concept of the algorithm because of the difficulty involved in using the syntax of the programming language. When considering the problem of students' preferences regarding the manner of presenting algorithms, the aforementioned researchers showed that, when given a choice, 95% of students chose to express algorithms using flowcharts rather than a traditional programming language. This was a different conclusion from previous research, including studies by [11] and [12], according to which flowcharts were only an alternative form of representing the syntax of a language and did not help in understanding algorithms, especially in the case of experienced programmers.

These discrepancies in the results obtained by researchers and, at the same time, the growing efforts made by many governments to adopt computational thinking and coding as a new student skill that is equal to literacy in terms of importance as well as to support students in creative problem solving have led us to attempt a new and objectified study of how the way an algorithm is presented (pseudocode versus flowchart) affects its understanding.

An in-depth analysis of the body of research concerning this issue allowed us to determine what tasks students should solve in an experiment and under what conditions. We expanded the methodology by an objective eye tracking research technique which permitted us to arrive at a triangulation of results: task solutions, survey data, and measurements of eye coordinates. We made a hypothesis that structured flowcharts would prove more efficient than pseudocode under all experimental conditions adopted. We also assumed that flowchart analysis takes less time compared to pseudocode, does not require as much re-analysis and referencing of input data, and generates fewer incorrect responses. We planned an experiment in which 114 students solved two types of tasks (presented in the form of pseudocode and flowcharts).

Our results enrich previous findings regarding the usefulness of teaching algorithmic elements using these forms of algorithm presentation. They will support teachers as well as other creators of algorithmic and programming instructional content to optimise methods of educating and working with students.

## II. BACKGROUND

The increasing risk of over-consumption and lack of reflection on the use of new technologies combined with the growing opportunities for using them for private and professional purposes have made it necessary to return to the sources of understanding problems and to provide training on the ability to approach them in such a way that information technology helps to solve these problems creatively and rationally [13]. As a result, a great deal of interest in the development of students' computational thinking skills has been observed in recent years on the part of educational researchers. The term computational thinking is understood broadly, some researchers assume that it is not a field of computer science but rather the application of rules that computers understand [14], [15]. It is also defined as ''an approach to solving problems, designing systems, and understanding human behaviour that draws on concepts fundamental to computing'' [16]. The need to use computational thinking arises when the solving of a problem requires the determining of logical relationships between data, drawing conclusions based on the analysis of their interrelationships, and the ability to create algorithms. As a result, it becomes natural to focus on teaching coding in schools, understood broadly as an approach to solving problems ranging from problem determination through finding and developing a solution to writing it in a programming language, testing its correctness and introducing possible corrections. Programming requires logical thinking, abstracting, and critical analysis skills that underlie computer thinking [17]. Programming has become the primary way (means) of teaching computational thinking [18], [19] as it has been introduced to primary and secondary school curricula [4], [20], [21].

Analysing, reading, and writing algorithms is one of the first skills acquired by novice programmers. Expanding knowledge in this area seems crucial since failure to master the competences related to the presentation and construction of algorithms is widely recognised as one of the primary causes of difficulties accompanying the learning of programming [8]. Whether an algorithm is easier to comprehend if presented as a flow chart or as pseudocode was the subject of research by [22] who showed that algorithms presented in the form of flowcharts had an advantage in the process of understanding them. In their studies, this advantage was revealed in the case of complex algorithms - test subjects made more errors when faced with a text representation of the algorithm, while no such relation was observed in the case of simple algorithms. Kammann [23] also studied the understanding of ''procedures'' written in the form of verbal instructions and diagrams. He assumed that both speed and correctness of response constituted the measure of comprehension; in his study, diagrams were found to work better. In contrast, [12] postulated that flowcharts were not helpful for writing, understanding, or modifying computer programs and that no statistically significant differences were found between the flowchart and non-flowchart groups. Arguing against the findings of [12] and [24] criticised the research methodology

they used. According to [24], its drawbacks included: using unstructured and only simple algorithms, a small number of subjects studied (N=20), and an inappropriate criterion for comparison. In the view of [24], that criterion should not be the score obtained but the time the respondents took to answer the questions, given that time needed to understand an algorithm is the key and most sensitive measure of its difficulty. In his studies, [24] applied algorithm structures that were practically identical to the ones used by [25] who studied the effectiveness of the graphic representation of algorithms versus other forms of algorithm presentation and found that decision trees were superior to both structured English and decision tables for representing conditional logic. The results obtained by [24] indicated that presenting algorithms in the form of flowcharts was superior to pseudocode.

A later study [26] also confirmed that a graphic representation of algorithms seemed to be more effective (in terms of correct reading and determining the result of their operation), if only because tracking the control flow (the sequence of instructions) was easier.

Meanwhile, [27] found that both visual programming and flowcharts were suitable for designing algorithms, with no statistical difference in terms of the number of errors or the time required to write the corresponding Java code.

The discrepancies in research results show how crucial it is to seek objective research methods and techniques, as well as a methodologically correct manner of executing the research process. All this drew out interest towards using eye tracking techniques in our research.

## III. EYE TRACKING RELATED WORK

Previous research using eye tracking, which provides a better understanding of the process of acquiring programming skills through analysing cognitive processes (related to visual attention), was mainly concerned with various aspects of program code analysis [28]. The study by [29], in which they compared oculographic data of novice and advanced programmers to see how experience affected code scanning patterns in the process of code interpretation (understanding), can be considered ground-breaking.

The research in [30] was also involved in searching for behavioural patterns during code analysis. The researchers noted that, when searching for bugs in programs, most subjects first performed a preliminary scan of the entire code, presumably to understand the structure of the program, and only later focused on its selected parts. Furthermore, the time spent on this initial code scan influenced the effectiveness of error detection.

Similar conclusions were reached by the study by [31], which further demonstrated that scanning time affected the visual effort (measured by the number and time of fixations) needed to identify errors. They also noted that experienced programmers devoted less time to the initial scanning of the program preceding the actual bug search compared to novices. While continuing the research on these issues, [32] showed that program content analysis was a non-linear procedure and that experts read code less linearly than novices. Meanwhile, [33] recorded the oculomotor activity of students analysing Java codes in a programming environment that provided code visualisation in addition to code editing tools. The purpose of their experiment was to investigate the dynamics of students' interactions with different forms of program representations in the course of program analysis (understanding program operation) and to compare whether these processes occurred similarly in individuals with different levels of experience. Oculographic results also show that the process of reading program source code is fundamentally different from reading natural language text but that these discrepancies disappear if the program code becomes similar to natural language text [34], [35].

However, there is little research that uses eye tracking while focusing on solving algorithmic problems presented in pseudocode and flowchart form. In their analysis of this topic, [36] divided study participants into two groups: correct answers (effective group) and incorrect answers (ineffective group). Algorithms presented in graphic form were easier to interpret for the ineffective group. The results of the study confirmed the hypothesis that the use of formal notation specific to a programming language for the purpose of presenting algorithms is often a practical difficulty in the process of solving even simple tasks. Algorithms presented in graphic form were easier to interpret for the ineffective group.

In the approach that we present which uses the eye tracking technique, we undertook to develop research concepts which compare the efficiency of solving algorithmic tasks presented in the form of pseudocode with those presented as flowcharts.

## IV. METHODS

### A. CURRENT STUDY

Our study takes into account the majority of the methodological elements identified by [24] and in addition, an eye tracking technique was applied to provide a qualitative expansion of the set of experimental data obtained. The said data provided detailed information not only regarding the temporal, but also the spatial distribution of the respondents' attention during the process of algorithm analysis. The eye tracking study made it possible, among other things, to separate the phases of reading the instructions, analysing the algorithm, and referencing the input data. Furthermore, we expanded the outlines of our experiment by a new element, i.e. we introduced an additional type of assignment that required a different solving procedure. Two types of assignments were used to ensure that the same trends related to the level of complexity and the form of representation of the assignment emerged in both cases. The design of those assignments was modelled on algorithms used by [37] in their study; it should be noted, however, that the structure of the algorithms in both types of assignments was exactly the same. In addition, our study sample was more diverse in terms of programming skills.

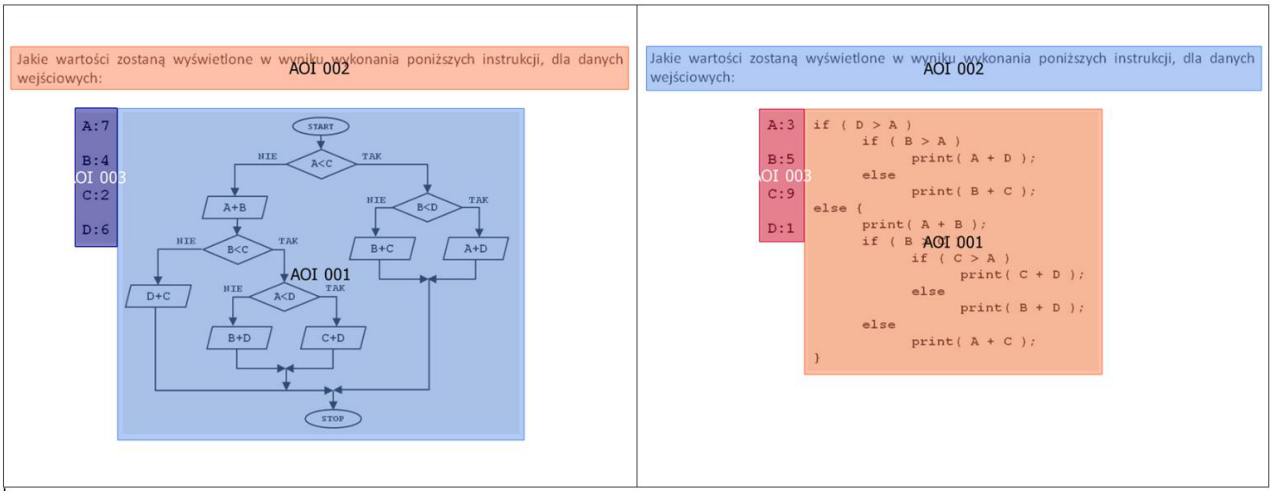


**FIGURE 1.** Areas of interest (AOIs) - True-False algorithms, Medium level, left Flowchart right Pseudocode.

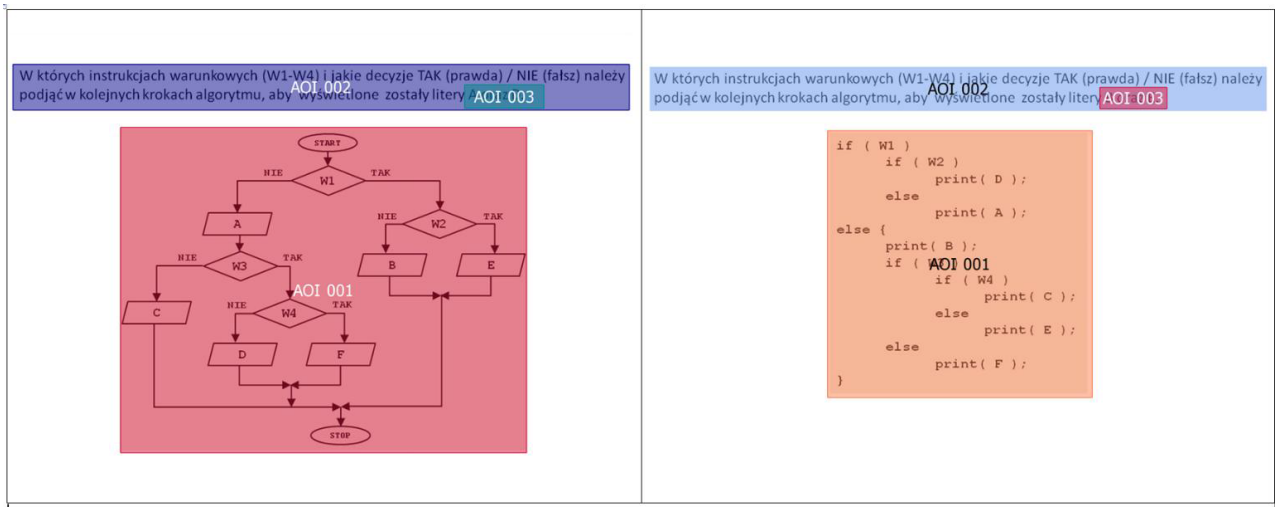


**FIGURE 2.** Areas of interest (AOIs) - Output-Answer algorithms, Medium level, left Flowchart, right Pseudocode.

Study participants also included novices, namely students of computer science who had started learning programming a few months earlier and had just completed an introductory programming course (CS1).

### B. RESEARCH QUESTIONS AND HYPOTHESIS

We made a general hypothesis that structured flowcharts would prove more efficient than pseudocode under all experimental conditions adopted. In particular, we assume that solving a task presented in the form of structured flowcharts: H01: takes less time, H02: reduces the number of algorithm re-analyses, H03: reduces the number of ''returns'' to (re-referencing of) input data, H04: generates fewer incorrect answers, H05: gives students more confidence that they have understood the algorithm.

We also predict that H06: flowcharts are more efficient in terms of time required to analyse algorithms as their complexity increases. In addition, we strive to answer the following research questions: (1) does the students' level of programming skills affect dependent variables associated with hypotheses H01 to H05, and (2) will the same trends be observed for both types of assignment.

### C. EYE MOVEMENT PARAMETERS

Areas of interest (AOIs) were marked on all the boards of all the assignments to provide detailed information about the distribution of the respondents' visual attention. AOIs were determined for the following areas: (AOI 001) assignment instruction, (AOI 002) algorithm, (AOI 003) input data (see Figure 1 and Figure 2), and AOIs for individual pseudocode lines and flowcharts (see Figure 3).

Using the SMI BeGaze$^{TM}$2.4 software, information was generated concerning the primary indicators of visual activity called Key Performance Indicators (KPIs), the names and meaning of which was derived from [38]. Two AOI-related eye tracking parameters were analysed and interpreted for the purposes of examining study hypotheses: (1) Dwell Time (DT), i.e. the duration of watching a selected AOI (where time is the sum of fixation and saccade times) and (2) Glances Count (GC), that is the number of glances towards the AOI

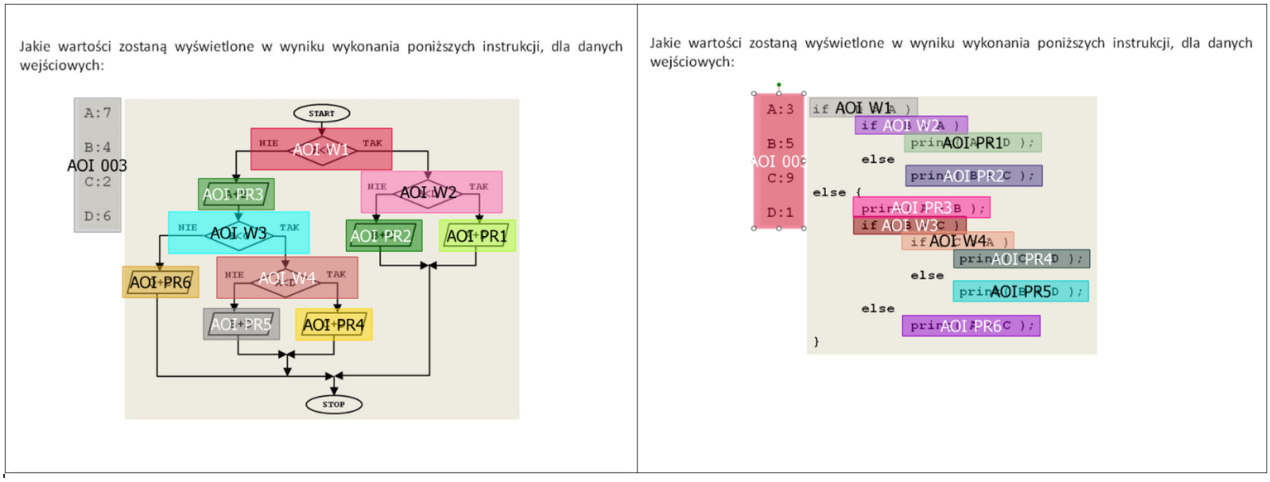


**FIGURE 3.** Areas of interest (AOIs) – lines and blocks, Output-Answer algorithms; left Flowchart, right Pseudocode.

in the case of externally triggered saccades. We chose these ocular indicators because we assumed that the most important dependent variable was the time spent on algorithm analysis. In order to maximise the sensitivity of this variable, study participants were allowed to view the algorithms for as long as necessary to understand them. We also assumed that the more difficulty the respondents had with interpreting the algorithm, the more often they returned to code lines or flowchart blocks they had already ''viewed'' and ''jumped'' between the algorithm area and the input data area.

### D. DESIGN

The main independent variable was the form used to present the algorithms, i.e. structured flowcharts versus pseudocode. The level of programming skill (Expertise) was an additional independent variable in our model - the participants were divided into 2 groups (Novice versus Non-novice). The main dependent variables directly related to the hypotheses were: 1. time associated with algorithm analysis: the number of milliseconds (Dwell Time) spent looking at the algorithm area (AOI 001) (see Figure 1 and Figure 2); 2. number of algorithm re-examinations: the number of additional return glances (Glances Count) on the areas of single lines or single blocks of algorithms (see Figure 3); 3. number of return times to input data: the number of glances (Glances Count) per area (AOI 003), an analysis which was only performed for the True-False task (see Figure 1); 4. percentage of correct answers to questions concerning algorithms; 5. the level of confidence associated with the correctness of the answers provided: measured for each answer according to the Likert scale ranging from very low (1) to very high (5). The data collected was mainly analysed using a mixed model ANOVA, while significant effects were analysed using Tukey's post-hoc HSD test. Variance analysis was performed independently for the two types of assignment (True-False and Output-Answer) and independently for each level of algorithm complexity (Simple, Medium, Complex) for all dependent variables. Within-subject factors include two forms of algorithm representation (Flowchart (FC) and Pseudocode (PC)), while between-subject factors are two levels of expertise (Novice and Non-novice). The analysis encompassed the main effects of Form (FC vs. PC) and Expertise (Novice vs. Non-novice) as well as the interaction effect of Form*Expertise.

### E. PARTICIPANTS AND PROCEDURE

The study group consisted of 115 subjects aged 19 to 26 (M=21.72, SD=1.99), of which there were 97 males and 18 females. All participants were students majoring in computer science at the same university. They participated in the study on a voluntary basis. For technical reasons, the oculographic data of one individual was discarded and the results of 114 students were accepted for further analysis. The participants' experience in the field of programming varied. They were divided into two groups (Novice vs. Non-novice) based on the level of their programming skills (Expertise). The Novice group comprised 1st year students (N=58, number of years of learning to program M=1.50, SD=0.82), while 2nd and 3rd year students were assigned to the Non-novice group (N=56, number of years of learning to program M=2.49, SD=0.84).

The eye tracking experiment took place in a room that ensured the same conditions for all subjects in terms of lighting, temperature, and acoustic isolation.

Each examination took place individually, with only one participant and the experiment supervisor present in the room at any given time. Prior to the experiment, each participant read the outline of the experiment procedure and solved sample model tasks. The study proper, which involved solving assignments displayed on a computer screen, was recorded by an eye tracker. This part of the experiment began with measures ensuring that the eye tracking measurements were performed correctly, i.e. by setting up the correct position of the subject and calibrating the device.

Questions were displayed on the screen in random order. The participants did not use any writing utensils. The time

for solving the assignments was not limited; each respondent worked at their own pace. The respondents provided verbal answers and, after giving them, rated their level of confidence in the correctness of the relevant solution. After this stage was completed, each participant filled out an electronic questionnaire which collected demographic data.

### F. RESEARCH MATERIALS

The experiment involved testing two types of algorithm (differing in terms of the manner of providing answers) at three levels of complexity: Simple, Medium, and Complex. The algorithms were displayed in the form of pseudocode and flowcharts. The test consisted of 12 questions, six of which were of the True-False (TF) type and six of the Output-Answer (OA) type. There were four tasks at each level of complexity (Simple, Medium, and Complex); six questions were presented in the form of a flowchart (FC) and 6 in the form of pseudocode (PC). Examples of Medium level assignments are shown in Figure 1 and Figure 2.

All algorithms had the same structure of conditional blocks, with two blocks at Simple level, 4 blocks at the Medium level, and 6 blocks at the Complex level. TF assignments were modelled after the tasks used by [24]. They posed questions concerning the Boolean states for each conditional block. The questions were as follows: ''What YES (true) / NO (false) decisions should be made in which conditional instructions (W1...) in the next steps of the algorithm in order to display ...''. Output-Answer assignments were modelled after the tasks used by [37] in their study. They were worded as follows: ''What values will be displayed as a result of executing the following instructions, for input data ...''. The same typeface and font size were used in the flowcharts and pseudocode, and the sizes of all blocks were also the same. The algorithms were displayed centrally on a computer screen.

### G. EYE TRACKING DEVICE

The course of the experiment was recorded in real time using an iViewX Hi-Speed eye tracker manufactured by Senso-Motoric Instrument (SMI). It is an instrument designed for non-invasive testing at high sampling rates (i.e. 500/1,250 Hz) in a laboratory setting. The workstation includes a computer used to manage the experiment, 2 computer screens (one each for the test subject and the supervisor), and an eye tracking module. The device permits the subject to hold their head still without restricting their field of vision. The tasks were displayed on a 23'' LCD screen with Full HD 1920 × 1080 resolution. A 13-point calibration process was performed prior to the session. Ultimately, a final validation was performed to check for any calibration inaccuracies. The experiment was conducted using SMI Experiment Suite$^{TM}$360 software.

## V. RESULTS

Two-way mixed ANOVA with one within-subject variable Form (FC vs. PC) and one between-subject factor Expertise (Novice vs. Non-novice) was performed for each dependent variable. The main effects of the Form (FC vs. PC) and Expertise (Novice vs. Non-novice) and interaction effect Form*Expertise was analysed.

### A. TIME OF SOLVING ALGORITHMS

Variance analysis revealed a significant main effect (FC vs. PC) at each level of algorithm complexity and for both assignment types, with the exception of the Simple level in the Output-Answer assignment. Average time of flowchart analysis was significantly shorter compared to pseudocode at all three levels of complexity, except for the Simple level in the Output-Answer assignment (see Table 1). This means that, in line with our predictions, respondents took less time to understand structured flowcharts. No significant main effect related to level of expertise (Novice vs. Non-novice) emerged in any case, while a significant interaction effect between assignment form and the subjects' expertise (Form*Expertise) was observed for both assignment types at the Complex level (True-False: $F(1, 112) = 4.92$, $p = 0.029$, Output-Answer: $F(1,112) = 4.84$, $p = 0.030$). The post-hoc analysis of the interaction effect was performed using Tukey's HSD test. In terms of the True-False assignment, as per our expectations, Novices took the longest to solve the Complex-level pseudocode assignment (M_PC=52.935 s); the time required in their case was significantly longer ($p=0.035$) than the time Non-novices devoted to this assignment form (M_PC=40.704 s). In general, the times spent by the Novices (M_FC=19.419 s) and Non-novices (M_FC= 20.045 s) in solving flowchart-based assignments were the only ones which did not show significant differences ($p=0.999$). In terms of the Output-Answer assignment, significantly less time ($p<0.001$) was needed by Novices when solving the assignment using the flowchart (M_FC=28.753 s) compared to pseudocode (M_PC=45.720 s). It also deviated at the level of statistical trend ($p=0.059$) from the time needed by Non-novices to solve the pseudocode-based assignment (M_PC=40.472 s). The relationships discussed are also shown in Figure 4.

### B. RATIO TIME

In order to verify whether structured flowcharts continued to be more efficient than pseudocode with an increase in algorithm complexity, ratios were designed that compared pseudocode to flowchart in terms of the time needed to solve the given assignment. At each level of algorithm complexity, the ratio of the higher time value (flowchart or pseudocode) to the lower time value was calculated for every participant. If the time value was lower for pseudocode, the ratio was marked as negative; meanwhile, if it was lower for flowcharts, it was marked positive. Positive ratios indicated the predominance of flowcharts, and negative ratios the predominance of pseudocode. ANOVA with one within-subject variable Complexity (Simple vs. Medium vs. Complex) and one between-subject factor Expertise (Novice vs. Non-novice) was performed for each type of assignment

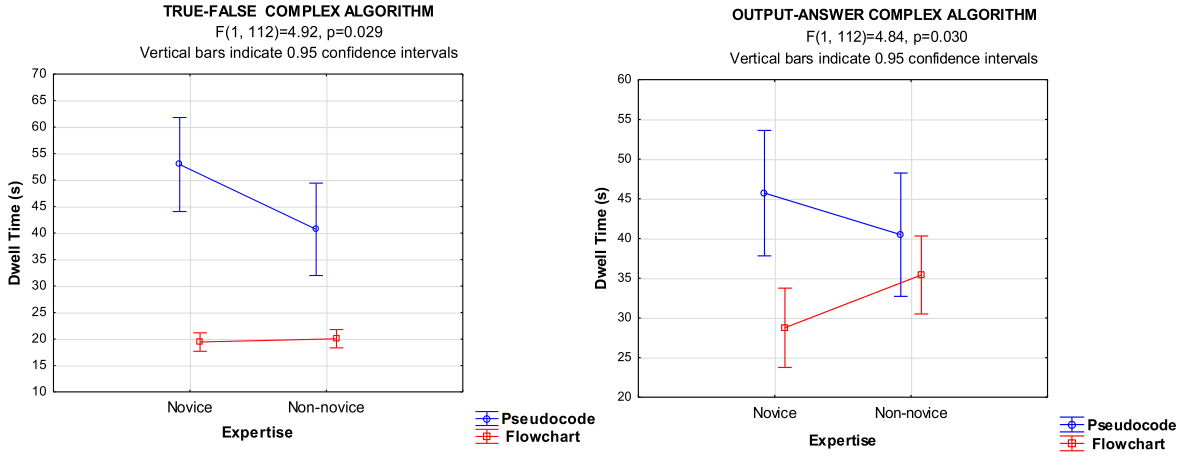


**FIGURE 4.** Interaction effect Expertise*Form for Dwell Time (s): left True-False Complex algorithm, right Output-Answer Complex algorithm.

**TABLE 1.** ANOVA for Dwell Time (s) involved in analysing the algorithms.

| | Flowchart | | Pseudocode | | | |
|---|---|---|---|---|---|---|
| | M | SD | M | SD | F | p |
| True-False task | | | | | | |
| Simple | 8.59 | 3.61 | 12.53 | 7.00 | 36.72 | 0.000 |
| Medium | 13.43 | 6.49 | 24.50 | 12.20 | 91.11 | 0.000 |
| Complex | 19.74 | 6.59 | 46.71 | 33.94 | 87.35 | 0.000 |
| Output-Answer task | | | | | | |
| Simple | 15.12 | 7.45 | 14.39 | 7.20 | 1.00 | 0.050 |
| Medium | 19.41 | 10.17 | 25.60 | 17.16 | 15.11 | 0.000 |
| Complex | 32.14 | 19.09 | 43.05 | 29.83 | 16.56 | 0.000 |

In this and other tables: M – Mean, SD – Standard Deviation, F – F-Snedecor test, p – p-value, probability

for ratios established in this manner. The main effect associated with the Complexity variable was significant for each type of assignment (True-False: $F(2,224) = 21.41$, $p < 0.001$; Output-Answer: $F(2,224) = 9.52$, $p < 0.001$). For both assignment types, variance analysis revealed no significant main effect for the Expertise factor and no interaction effect between algorithm complexity and level of programming skill (Complexity*Expertise). The average factor values amounted to, respectively: for the True-False assignment, 1.09 for simple algorithms, 1.84 for medium algorithms, and 2.30 for complex algorithms; for the Output-Answer assignment, −0.03 for simple algorithms, 0.76 for medium algorithms, and 0.92 for complex algorithms. These averages would amount to 0.00 if there was no prevalence of flowcharts over pseudocode or vice versa. The relationships discussed are also presented in Figure 5. Further post-hoc analyses for the True-False assignment showed that all mean values of the ratios differed to a significant degree. Significance levels were $p<0.001$ (Simple, Medium); $p<0.001$ (Simple, Complex), and $p<0.001$ (Medium, Complex), respectively. For the Output-Answer assignment, the Simple level ratio differed significantly from both the Medium level ($p=0.004$) and the Complex level ($p<0.001$). Meanwhile, the Medium level ratio was not significantly different from the Complex level ratio ($p=0.799$).

### C. NUMBER OF ALGORITHM RE-ANALYSES

The variance analysis resulted in a statistically significant main effect related to the form of algorithm presentation (FC vs. PC), i.e. the average number of returns to already analysed algorithm areas (returns to areas of individual lines or individual blocks (see Figure 3)) was significantly lower in the case of flowcharts compared to pseudocode in terms of both assignment types and at all three levels of complexity, except for the Simple level in the Output-Answer assignment (see Table 2).

For both assignment types, variance analysis revealed no significant main effect for the Expertise factor (Novice vs. Non-novice) and no significant interaction effects between the form of algorithm presentation and the programming skills of the respondents (Form*Expertise) at any level of algorithm complexity.

### D. NUMBER OF RETURNS TO INPUT DATA

The analysis was only performed with respect to the True-False assignment due to the manner of solving the Output-Answer assignment which required referencing the input data for each conditional block. A statistically significant main effect related to the form of algorithm presentation was observed at each level of algorithm complexity, while the average number of return times to input data was significantly lower ($p<0.000$) in the case of flowcharts compared to pseudocode (see Table 3). Variance analysis revealed no significant main effects for the Expertise factor (Novice vs. Non-novice) and no significant interaction effects between the form of algorithm presentation and programming skills (Form*Expertise) at any level of algorithm complexity.

### E. CORRECT ANSWERS

The respondents made fewer errors in the case of flowcharts, i.e. the average percentages of correct answers obtained with respect to algorithms presented as flowcharts were higher than in the case of pseudocode, with the exception of the Output-Answer task at the Complex level (see Table 4). For both assignment types, variance analysis revealed no

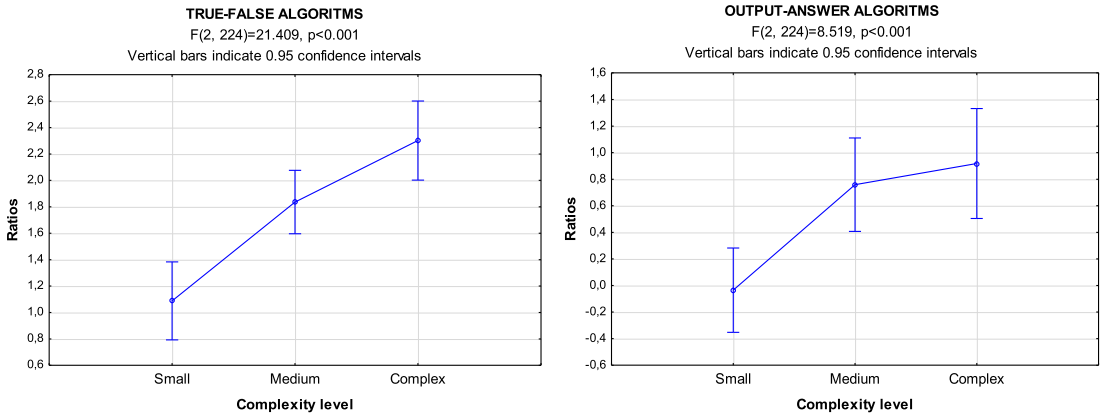


**FIGURE 5.** ANOVA for Ratios time; left True-False algorithms, right Output-Answer algorithms.

**TABLE 2.** ANOVA for the number of times the algorithm areas (PC lines or FC blocks) were re-analysed.

| | Flowchart | | Pseudocode | | | |
|---|---|---|---|---|---|---|
| | M | SD | M | SD | F | p |
| True-False task | | | | | | |
| Simple | 16.36 | 6.09 | 23.75 | 11.75 | 42.83 | 0.000 |
| Medium | 27.39 | 22.54 | 41.65 | 22.34 | 81.48 | 0.000 |
| Complex | 37.67 | 33.02 | 76.32 | 47.95 | 111.09 | 0.000 |
| Output-Answer task | | | | | | |
| Simple | 21.58 | 11.37 | 23.97 | 12.36 | 3.52 | 0.050 |
| Medium | 28.25 | 29.43 | 40.86 | 28.24 | 40.28 | 0.000 |
| Complex | 40.57 | 39.42 | 62.56 | 42.30 | 46.01 | 0.000 |

**TABLE 3.** ANOVA for the number of return times to the input data areas, True-False task.

| | Flowchart | | Pseudocode | | | |
|---|---|---|---|---|---|---|
| | M | SD | M | SD | F | p |
| Simple | 4.75 | 2.19 | 6.95 | 3.34 | 41.54 | 0.000 |
| Medium | 4.53 | 1.89 | 8.08 | 3.84 | 80.08 | 0.000 |
| Complex | 4.98 | 2.10 | 9.54 | 5.66 | 70.02 | 0.000 |

significant main effect for the Expertise factor (Novice vs. Non-novice) and no significant interaction effects between the form of algorithm presentation and the programming skills of the respondents (Form*Expertise) at any level of algorithm complexity.

### F. CONFIDENCE LEVEL

In the case of flowcharts, the respondents were more confident they had solved the task correctly. Variance analysis revealed a statistically significant main effect related to the form of algorithm presentation (FC vs. PC), i.e. mean levels of the Confidence variable differed significantly in favour of flowcharts for both types of assignment and at each level of algorithm complexity, except for the Simple level in the Output-Answer assignment (see Table 5). For both assignment types, variance analysis revealed no significant main effects related to the Expertise factor (Novice vs.

**TABLE 4.** ANOVA for the percentage (%) of correct answers to the questions about the algorithms.

| | Flowchart | | Pseudocode | | | |
|---|---|---|---|---|---|---|
| | M | SD | M | SD | F | P |
| True-False task | | | | | | |
| Simple | 100.00 | 0.00 | 97.37 | 16.08 | - | - |
| Medium | 100.00 | 0.00 | 85.09 | 35.78 | - | - |
| Complex | 99.12 | 9.37 | 69.30 | 46.33 | 44.87 | 0.000 |
| Output-Answer task | | | | | | |
| Simple | 91.23 | 28.41 | 91.23 | 28.41 | - | - |
| Medium | 85.09 | 35.78 | 76.32 | 72.40 | 4.29 | 0.041 |
| Complex | 64.04 | 48.20 | 77.19 | 42.14 | 6.27 | 0.014 |

Non-novice). There was an interaction effect at the level of statistical trend between the form of algorithm presentation and the level of programming skills (Form*Expertise) with respect to the True-False assignment at the Complex level ($F(1,112) = 3.34$, $p = 0.070$). The post-hoc analysis using Tukey's HSD test showed that Novices rated their confidence level significantly lower ($p<0.001$) with respect to assignments presented in the form of pseudocode (M_PC=4.00) compared to the flowchart version (M_FC=4.77). The situation was similar in the Non-novice group (M_PC=4.14, M_FC=4.64), the difference between confidence levels was smaller but also significant ($p<0.001$). The relationships discussed are also presented in Figure 6.

## VI. DISCUSSION

The analysis of our study results indicates that the way an algorithmic task is represented (flowchart vs. pseudocode) affects the process of its solution. In general, we observed a similar relation between the variables for both types of assignment used in our experiment (True-False and Output-Answer) and our results are similar to those obtained by [24]. The subjects took significantly less time to analyse algorithms in the case of flowcharts (versus pseudocode), made significantly fewer errors (with some exceptions), and had a considerably greater confidence in the correctness of their solution. With respect to ratio time, our results were also

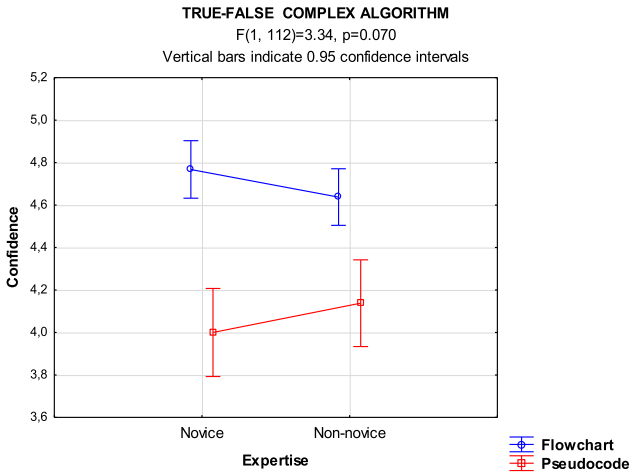


**FIGURE 6.** Interaction effect Expertise*Form for Confidence level, True-False Complex algorithm.

**TABLE 5.** ANOVA for confidence level related to the answers to the questions about the algorithms.

| | Flowchart | | Pseudocode | | | |
|---|---|---|---|---|---|---|
| | M | SD | M | SD | F | p |
| True-False task | | | | | | |
| Simple | 4.82 | 0.40 | 4.69 | 0.53 | 11.66 | 0.001 |
| Medium | 4.78 | 4.31 | 4.31 | 0.78 | 51.38 | 0.000 |
| Complex | 4.70 | 0.51 | 4.07 | 0.78 | 74.90 | 0.000 |
| Output-Answer task | | | | | | |
| Simple | 4.50 | 0.64 | 4.60 | 0.56 | 2.97 | >0.05 |
| Medium | 4.36 | 0.75 | 4.08 | 0.88 | 10.94 | 0.001 |
| Complex | 4.08 | 0.73 | 3.75 | 0.92 | 14.96 | 0.000 |

similar to [24] (especially in True-False assignments), i.e. as the complexity level of the algorithms increased, their analysis in the case of pseudocode took the subjects an increasing amount of time compared to structured flowcharts. The mean values of all ratios (ratio time) differed significantly in pairs. As a result, we confirmed the observations of [37] that the more complex an algorithm becomes, the more efficient it is to use flowcharts.

Although the results provide a strong case in favour of flowcharts, there were some exceptions. Of particular interest seems to be the outlier for the correct answer variable in the Complex/Output-Answer task (see Table 4). The differences that appeared were statistically significant and indicated a pseudocode advantage.

We assumed that this was not directly related to the form (flowchart vs. pseudocode) in which the task was presented, but was the result of a wrong decision in condition W1 (flowchart) and the subjects' incorrect execution of an arithmetic operation. In the case of the pseudocode 1·2 should have been multiplied, while in the case of the flowchart it was 6·9. Which seems to be a more difficult task and likely to generate errors. To test our assumptions related to arithmetic operations we conducted an additional analysis of the value of average fixation duration for selected areas of interest. Many eye tracking investigations confirmed that longer fixation duration on the AOI is connected with difficulty in interpreting the information present or a greater involvement in its exploration, and by contrast the AOIs that are comprehensible, and those that do not contribute significant semantic informativeness, have a shorter duration of fixation. The paired t-test was performed for the average duration of fixation (AFD) variable for AOIs, in which the calculations (AOI marked as PR5) and data (AOI marked as AOI3) were stored. For both AOIs in question, the average fixation duration was significantly longer for flowchart-based assignments (PR5: M_FC=304.33 ms, M_PC= 242.02 ms, t= 2.818, p=0.006; AOI3: M_FC=277.55, M_PC=235.11, t= 8.999, p<0.001).

For more detail on the average duration of fixation, we also performed a t-test against groups of correct and incorrect responses (see Table 6) for each of the AOIs discussed. As can be seen from Table 6, significant differences between the groups only occurred for the flowchart and AOI related to the multiplication operation (PR5). But the result obtained was surprising, as it turned out that those who gave an incorrect answer had a significantly shorter average fixation time (M=178. 94 ms) than those who gave the correct answer (M=374.77). The next step of data analysis showed that 21 of the 41 people who answered incorrectly did not fixate at all in the PR5 area, and the average fixation time of the remaining 20 people was similar (M=366.83 ms) to those who gave the correct answer. Analysis of the responses given by these 21 subjects showed that 17 of them had already made the wrong decision at the initial stage of solving the task, i.e. condition W1, which was associated with a lack of fixation in AOI PR5.

In summary, the analysis of the eye tracking data firstly supported our assumption that performing the 6·9 multiplication was more difficult for the subjects than performing the 1·2 calculation, this was indicated by the significantly longer average fixation time in AOI PR5 and secondly confirmed that the subjects misinterpreted the W1 condition in the flowchart task.

The eye tracking technique enabled the re-analyses of algorithms to be measured based on the number of fixation regressions in areas of individual code lines and algorithm blocks. We observed that the respondents were significantly less likely to re-analyse flowcharts than pseudocode for each combination of variables (with the exception of the OA assignment at the Simple level).

The eye tracking analysis of re-referencing input data was performed only for True-False assignments because Output-Answer assignments required referencing input data for each conditional block and each conditional instruction. Regardless of the level of algorithm complexity, the respondents were significantly more likely to return to the input data in the case of pseudocode than in the case of flowcharts. We can find a parallel here with the studies of [39] who observed an increase in the number of fixations related to transitions between areas of pseudocode and problem description as the

**TABLE 6. T-test for the average duration of fixation (AFD) involved in selected AOIs for correct and incorrect answers.**

| | Correct answers | | Incorrect answers | | | |
|---|---|---|---|---|---|---|
| | M | SD | M | SD | t | p |
| Flowchart | | | | | | |
| PR5 | 374.77 | 194.42 | 178.94 | 277.64 | 4.41 | 0.000 |
| AOI003 | 282.14 | 53.47 | 269.39 | 63.06 | 1.14 | 0.255 |
| Pseudocode | | | | | | |
| PR5 | 245.29 | 63.36 | 230.96 | 76.25 | 0.97 | 0.336 |
| AOI003 | 232.38 | 33.85 | 244.38 | 45.80 | -1.46 | 0.148 |

problem difficulty increased, as well as with the research of [40] who found that students with lower programming skills were more likely to return to the initial values of variables needed to perform calculations when solving algorithmic tasks.

The analysis of the results also showed the existence of relations between the form of assignment and the level of the respondents' programming skills as regards the duration of solving assignments at the Complex level for both types of tasks. In the True-False assignment, Novices devoted significantly more time to analysing the pseudocode assignment compared to the Non-novice group; in the Output-Answer task, this relation was revealed at the level of statistical trend. No such effect was observed with respect to the other variables i.e. re-referencing input data, algorithm re-analyses, or the percentage of correct answers. Meanwhile, as regards the degree of confidence, an interaction effect between the variables in question (assignment form and skill level) was revealed at the level of statistical trend in the TF Complex task. Novices rated their confidence level far lower with regard to assignments presented in the form of pseudocode compared to their flowchart versions. A similar phenomenon was observed with respect to Non-novices, but in their case the difference in rating was smaller.

We note that our observations only apply to the analysis of structured conditional instructions and only to already designed algorithms. In the future, we are considering exploring more diverse algorithms (e.g. using loops) at the design stage, also using eye tracking technology.

## VII. CONCLUSION

The purpose of our experiment was to verify the efficiency of solving algorithmic assignments presented in the form of pseudocode and structured flowcharts. The tasks in both representation categories were divided into two types (True-False and Output-Answer) and had 3 levels of complexity: (Simple, Medium, and Complex).

In light of the results obtained, it should be concluded that our research hypotheses H01–H05 regarding the advantage of flowcharts over pseudocode were verified positively (with a few exceptions) for all levels of complexity and for both types of assignment. Solving a task presented in the form of structured flowcharts compared to pseudocode: (1) takes less time, (2) reduces the number of re-examinations of the algorithm, (3) reduces the number of "returns" to input data, (4) generates fewer incorrect answers, and (5) gives the respondents more confidence that they have understood the algorithm. Furthermore, significant interactions were observed in the case of complex algorithms related to the form of algorithm presentation and the level of programming skill in relation to the duration of task solving and the confidence level.

Our results indicate that the use of structured flowcharts in various educational materials, including textbooks, should be encouraged. Debatable conclusions concerning the effectiveness of teaching algorithms using pseudocode and flowcharts have contributed to reducing the role of graphic presentation of algorithms and, over time, even resulted in a downright elimination of flowcharts. It turns out that it is uneconomical to include block diagrams in traditional computer science textbooks, hence publishers are more likely to include pseudocode. It seems that the best solution in this situation is to provide additional algorithmic learning aids via flowcharts which are available to students electronically (online).

Our results confirm that flowcharts can help novice programmers understand how control proceeds - especially in complex conditional instructions, and it can be assumed that also in various types of loops (especially in nested loops). We therefore advocate that teachers use flowchart-based programming environments aimed at novices, which have functionalities that support concurrent representation of flowcharts and pseudocode to the greatest extent possible. We mean, for example pseudocode and flowchart are displayed concurrently, flowchart generates a pseudocode and the pseudocode generates a flowchart, synchronized highlighting and visual execution of flowchart and pseudocode.

In conclusion, our research supports the use of teaching methods that optimally combine both forms of algorithm presentation. These issues should be revisited, given that teaching to program and algorithmics fosters the development of computational thinking skills.

## REFERENCES

[1] A. Yadav, H. Hong, and C. Stephenson, "Computational thinking for all: Pedagogical approaches to embedding 21st century problem solving in K-12 classrooms," *TechTrends*, vol. 60, no. 6, pp. 565–568, Nov. 2016, doi: 10.1007/s11528-016-0087-7.

[2] A. Threekunprapa and P. Yasri, "Unplugged coding using flowblocks for promoting computational thinking and programming among secondary school students," *Int. J. Instruct.*, vol. 13, no. 3, pp. 207–222, Jul. 2020.

[3] P. Hubwieser, M. N. Giannakos, M. Berges, T. Brinda, I. Diethelm, J. Magenheim, Y. Pal, J. Jackova, and E. Jasute, "A global snapshot of computer science education in K-12 schools," in *Proc. ITiCSE Work. Group Rep.*, Vilnius, Lithuania, Jul. 2015, pp. 65–83.

[4] N. Carlborg, M. Tyrén, C. Heath, and E. Eriksson, "The scope of autonomy when teaching computational thinking in primary school," *Int. J. Child-Comput. Interact.*, vol. 21, pp. 130–139, Sep. 2019, doi: 10.1016/j.ijcci.2019.06.005.

[5] C. Angeli and M. Giannakos, "Computational thinking education: Issues and challenges," *Comput. Hum. Behav.*, vol. 105, Apr. 2020, Art. no. 106185, doi: 10.1016/j.chb.2019.106185.

[6] A. Lamprou and A. Repenning, "Teaching how to teach computational thinking," in *Proc. 23rd Annu. ACM Conf. Innov. Technol. Comput. Sci. Educ. (ITiCSE)*, Larnaca, Cyprus, Jul. 2018, pp. 69–74, doi: 10.1145/3197091.3197120.

[7] C. Cabo, "Student progress in learning computer programming: Insights from association analysis," in *Proc. IEEE Frontiers Educ. Conf. (FIE)*, Covington, KY, USA, Oct. 2019, pp. 1–8, doi: 10.1109/FIE43999.2019.9028691.

[8] A. Gomes and A. J. Mendes, "Learning to program—Difficulties and solutions," in *Proc. Int. Conf. Eng. Educ.*, Coimbra, Portugal, 2007, pp. 1–5.

[9] A. Luxton-Reilly, Simon, I. Albluwi, B. A. Becker, M. Giannakos, A. N. Kumar, L. Ott, J. Paterson, M. J. Scott, J. Sheard, and C. Szabo, "Introductory programming: A systematic literature review," in *Proc. IEEE Frontiers Educ. Conf. (FIE)*, Covington, KY, USA, 2019, pp. 55–106, doi: 10.1145/3293881.3295779.

[10] M. C. Carlisle, T. A. Wilson, J. W. Humphries, and S. M. Hadfield, "RAPTOR: A visual programming environment for teaching algorithmic problem solving," in *Proc. 36th SIGCSE Tech. Symp. Comput. Sci. Educ.*, St. Louis, MO, USA, 2005, pp. 176–180.

[11] H. R. Ramsey, M. E. Atwood, and J. R. Van Doren, "Flowcharts versus program design languages: An experimental comparison," *Commun. ACM*, vol. 26, no. 6, pp. 445–449, Jun. 1983, doi: 10.1145/358141.358149.

[12] B. Shneiderman, R. Mayer, D. McKay, and P. Heller, "Experimental investigations of the utility of detailed flowcharts in programming," *Commun. ACM*, vol. 20, no. 6, pp. 373–381, Jun. 1977, doi: 10.1145/359605.359610.

[13] P. Mishra and A. Yadav, "Of art and algorithms: Rethinking technology & creativity in the 21st century," *TechTrends*, vol. 57, no. pp. 10–14, Mar. 2013.

[14] M.-J. Tsai, J.-C. Liang, and C.-Y. Hsu, "The computational thinking scale for computer literacy education," *J. Educ. Comput. Res.*, vol. 59, no. 4, pp. 579–602, Jul. 2021, doi: 10.1177/0735633120972356.

[15] L. Zhang and J. Nouri, "A systematic review of learning computational thinking through scratch in K-9," *Comput. Educ.*, vol. 141, Nov. 2019, Art. no. 103607, doi: 10.1016/j.compedu.2019.103607.

[16] J. M. Wing, "Computational thinking and thinking about computing," *Philos. Trans. Roy. Soc. London A, Math. Phys. Sci.*, vol. 366, no. 1881, pp. 3717–3725, 2008, doi: 10.1098/rsta.2008.0118.

[17] B. Arfé, T. Vardanega, C. Montuori, and M. Lavanga, "Coding in primary grades boosts children's executive functions," *Frontiers Psychol.*, vol. 10, p. 2713, Dec. 2019, doi: 10.3389/fpsyg.2019.02713.

[18] E. Relkin, L. E. de Ruiter, and M. U. Bers, "Learning to code and the acquisition of computational thinking by young children," *Comput. Educ.*, vol. 169, Aug. 2021, Art. no. 104222, doi: 10.1016/j.compedu.2021.104222.

[19] T.-C. Hsu, S.-C. Chang, and Y.-T. Hung, "How to learn and how to teach computational thinking: Suggestions based on a review of the literature," *Comput. Educ.*, vol. 126, pp. 296–310, Nov. 2018, doi: 10.1016/j.compedu.2018.07.004.

[20] X. Wei, L. Lin, N. Meng, W. Tan, S. Kong, and Kinshuk, "The effectiveness of partial pair programming on elementary school students' Computational Thinking skills and self-efficacy," *Comput. Educ.*, vol. 160, 2021, doi: 10.1016/j.compedu.2020.104023.

[21] R. Scherer, F. Siddiq, and B. S. Viveros, "The cognitive benefits of learning computer programming: A meta-analysis of transfer effects," *J. Educ. Psychol.*, vol. 111, no. 5, pp. 764–792, Jul. 2019, doi: 10.1037/edu0000314.

[22] P. Wright and F. Reid, "Written information: Some alternatives to prose for expressing the outcomes of complex contingencies," *J. Appl. Psychol.*, vol. 57, no. 2, pp. 160–166, Apr. 1973, doi: 10.1037/h0037045.

[23] R. Kammann, "The comprehensibility of printed instructions and the flowchart alternative," *Hum. Factors, J. Hum. Factors Ergonom. Soc.*, vol. 17, no. 2, pp. 183–191, Apr. 1975.

[24] D. A. Scanlan, "Structured flowcharts outperform pseudocode: An experimental comparison," *IEEE Softw.*, vol. 6, no. 5, pp. 28–36, Sep. 1989.

[25] I. Vessey and R. Weber, "Structured tools and conditional logic: An empirical investigation," *Commun. ACM*, vol. 29, no. 1, pp. 48–57, Jan. 1986, doi: 10.1145/5465.5470.

[26] D. Hooshyar, R. B. Ahmad, M. Yousefi, F. D. Yusop, and S.-J. Horng, "A flowchart-based intelligent tutoring system for improving problem-solving skills of novice programmers," *J. Comput. Assist. Learn.*, vol. 31, no. 4, pp. 345–361, Aug. 2015, doi: 10.1111/jcal.12099.

[27] D. Giordano and F. Maiorana, "Teaching algorithms: Visual language vs flowchart vs textual language," in *Proc. IEEE Global Eng. Educ. Conf. (EDUCON)*, Tallinn, Estonia, Mar. 2015, pp. 499–504, doi: 10.1109/EDUCON.2015.7096016.

[28] U. Obaidellah, M. Al Haek, and P. C.-H. Cheng, "A survey on the usage of eye-tracking in computer programming," *ACM Comput. Surv.*, vol. 51, no. 1, pp. 1–58, Jan. 2019.

[29] M. E. Crosby and J. Stelovsky, "How do we read algorithms? A case study," *Computer*, vol. 23, no. 1, pp. 25–35, 1990.

[30] H. Uwano, M. Nakamura, A. Monden, and K.-I. Matsumoto, "Exploiting eye movements for evaluating Reviewer's performance in software review," *IEICE Trans. Fundam. Electron., Commun. Comput. Sci.*, vol. E90-A, no. 10, pp. 2290–2300, Oct. 2007, doi: 10.1093/ietfec/e90-a.10.2290.

[31] B. Sharif, M. Falcone, and J. I. Maletic, "An eye-tracking study on the role of scan time in finding source code defects," in *Proc. Symp. Eye Tracking Res. Appl.*, Santa Barbara, CA, USA, Mar. 2012, pp. 381–384, doi: 10.1145/2168556.2168642.

[32] T. Busjahn, R. Bednarik, A. Begel, M. Crosby, J. H. Paterson, C. Schulte, B. Sharif, and S. Tamm, "Eye movements in code reading: Relaxing the linear order," in *Proc. IEEE 23rd Int. Conf. Program Comprehension*, Florence, Italy, May 2015, pp. 255–265.

[33] R. Bednarik and M. Tukiainen, "An eye-tracking methodology for characterizing program comprehension processes," in *Proc. Symp. Eye tracking Res. Appl. (ETRA)*, San Diego, CA, USA, Mar. 2006, pp. 125–132, doi: 10.1145/1117309.1117356.

[34] T. Busjahn, C. Schulte, and A. Busjahn, "Analysis of code reading to gain more insight in program comprehension," in *Proc. 11th Koli Calling Int. Conf. Comput. Educ. Res.*, Koli, Finland, Nov. 2011, pp. 1–9.

[35] D. Binkley, M. Davis, D. Lawrie, J. I. Maletic, C. Morrell, and B. Sharif, "The impact of identifier style on effort and comprehension," *Empirical Softw. Eng.*, vol. 18, no. 2, pp. 219–276, 2013.

[36] M. Andrzejewska, A. Stolińska, W. Błasiak, P. Pęczkowski, R. Rosiek, B. Rożek, M. Sajka, and D. Wcisło, "Eye-tracking verification of the strategy used to analyse algorithms expressed in a flowchart and pseudocode," *Interact. Learn. Environ.*, vol. 24, no. 8, pp. 1981–1995, Nov. 2016, doi: 10.1080/10494820.2015.1073746.

[37] T. Crews and U. Ziegler, "The flowchart interpreter for introductory programming courses," in *Proc. 28th Annu. Frontiers Educ. Conf. (FIE)*, Tempe, AZ, USA, Nov. 1998, pp. 307–312.

[38] SMI. (2010). *BeGaze 2.4 Manual, Version 2.4*. Accessed: Jan. 21, 2022. [Online]. Available: https://www.tilburguniversity.edu/sites/default/files/download/BeGaze-2.4-Manual.pdf

[39] U. Obaidellah, T. Blascheck, D. T. Guarnera, and J. Maletic, "A fine-grained assessment on novice programmers' gaze patterns on pseudocode problems," in *Proc. ACM Symp. Eye Tracking Res. Appl. (ETRA)*, Stuttgart, Germany, Jun. 2020, pp. 1–5, doi: 10.1145/3379156.3391982.

[40] M. Andrzejewska and P. Kotoniak, "Development of program comprehension skills by novice programmers—Longitudinal eye tracking studies," *Informat. Educ.*, vol. 19, no. 4, pp. 521–541, Dec. 2020, doi: 10.15388/infedu.2020.23.

**MAGDALENA ANDRZEJEWSKA** received the M.S. degree in computer science from the AGH University of Science and Technology, Cracow, in 1996, and the Ph.D. degree in pedagogical sciences from the Educational Research Institute, Warsaw, in 2007. She is an Assistant Professor with the Institute of Computer Science and the Head of the Department of Educational Research and New Media, Pedagogical University of Krakow, Poland. She is involved in the teaching activity of various courses in the field of computer science, including programming techniques, object-oriented programming, algorithms, and data structures. She has authored and coauthored over 40 scientific publications. Her research interests include computer science education, learning and teaching programming, and using eye tracking technique to study of cognitive processes in learning and problem solving.

**ANNA STOLIŃSKA** received the M.S. degree in physics (specialization in physics with computer science) and the Ph.D. degree in pedagogy from the Pedagogical University of Krakow, in 2001 and 2009, respectively. She is currently a Professor with the Institute of Computer Science, College of Economics and Computer Science in Cracow. She is a Popularizer of IT education, an Active Activist, and a member of the Board of Directors with the Malopolska Branch of the Polish Information Processing Society. She has authored and coauthored over 50 scientific publications. Her research interests include human–computer interaction, teaching (didactics), e-learning (cognitive-based content personalization), and visual competence during learning process using eye tracker.