Informatics in Education, 2024, Vol. 23, No. 4, 735–765


DOI: https://doi.org/10.15388/infedu.2024.25

# A Transformation of the Way of Thinking

## Key Principles in Programming Education Inspired by Pascal

Valentina DAGIENĖ, Gintautas GRIGAS, Tatjana JEVSIKOVA

*Vilnius University, Institute of Data Science and Information Technologies*
*e-mail: valentina.dagiene@mif.vu.lt, gintautas.grigas@mif.vu.lt, tatjana.jevsikova@mif.vu.lt*

**Abstract.** The work of Niklaus Wirth, designer of the Pascal programming language, has led to the introduction of programming in schools in many countries often leading to a transformation in the way of thinking. In this article, we provide a retrospective analysis of the Lithuanian experience driven by Pascal and discuss the main ideas about teaching programming originating from this experience. We conducted a qualitative study by means of interviews with experts involved in the development of programming education during its early phases to examine their memories and perspectives.

Programming education in Lithuania started with the Pascal-inspired *Young Programmers' School by Correspondence*, founded in 1981, which had a great influence on the Lithuanian programming elite. For this purpose, a compiler for a subset of Pascal was developed for students taking their first steps towards programming, or more precisely, algorithmic thinking.

Many innovations were developed and brought into practice. The ones that have proved their worth and have not lost their relevance are the subject of this article. These include assessing program text readability, cultivating programming style, program reading tasks, creative thinking tasks, problem-solving approaches, detailed compiler error messages, automatic error fixing, and compiler advice to novice programmers. While some concepts became obsolete with technological advancements, others remain relevant, directly or as inspiration for new ideas, forming the basis of this study.

**Key words:** Pascal, Young Programmers' School, programming education, informatics.

### 1. Introduction

Pascal, a distinguished programming language that served in education for many years, was designed in 1968 at ETH Zurich by Niklaus Wirth, with the goal of encouraging novices to apply good programming practices by using *structured programming*. Pascal had a big influence on programming education in many countries. In Lithuania, Pascal was chosen as a language to communicate with big machines and express algorithms, especially for secondary school students. It was the backbone of the *Young Programmers' School by Correspondence* (Dagys *et al.*, 2006).

Informatics, including programming, started to be taught in all Lithuanian schools in 1986, but interest in teaching algorithms and programming had been shown almost a

decade earlier. Eager teachers stepped forward to teach it on their own. However, there were only a few of them.

The notion of making programming accessible to all students in Lithuania was first proposed in the late 1970s by Laimutis Telksnys. After long discussions and experiments, Pascal was chosen as programming language. A compiler was developed by the first two authors and Alma Petrauskienė, and in 1981 the Young Programmers' School by Correspondence was established. Numerous innovations stemmed from this initiative, which are crucial to emphasize and analyse in contemporary times.

This article retrospectively analyses the experience of Lithuania in the emergence of programming education, highlighting the fundamental principles that came into practice with the adoption of Wirth's Pascal.

The research questions of this study are the following:

- **RQ1.** What were the most important innovative ideas that arose during the early stages of the introduction of programming education in Lithuania?
- **RQ2.** What factors influenced the success of programming education, its emergence, and establishment in Lithuanian schools?

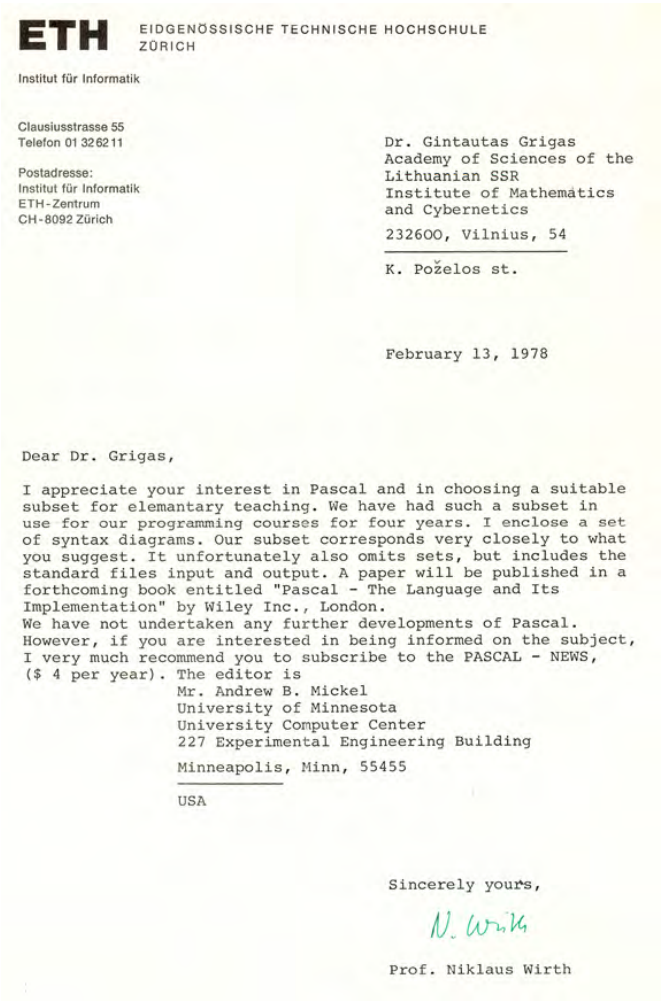
To address these research questions, a retrospective analysis of experiences, memories, communications, and publications has been conducted by utilizing reflection and qualitative analysis methods.

First, in Section 2, the background is presented, and the most important initiatives in programming education are reflected on. In Section 3, the methodology of our qualitative study is outlined. During the interviews, 11 experts – the first programming teachers, teachers who taught in the Young Programmers' School, and the younger-generation teachers who taught programming with Pascal, provide their insights. The results of the expert interviews are presented in Section 4. Finally, the findings of the whole study are discussed.

## 2. Background and Experience Analysis

In 1977, Laimutis Telksnys raised the idea that it would be useful to allow all Lithuanian school students to get acquainted with programming (at least in clubs or extracurricular activities). A working group at the *Institute of Mathematics and Cybernetics* was set up by initiative of the second author to prepare a curriculum and tools, and representatives from various universities where programming was already taught were invited.

The first important issue that arose was the programming language for teaching, with Algol 60 and Fortran being the dominant languages at that time. The new language PL/1 was spreading rapidly. PL/1 had a compiler for computers used in the Soviet Union in 1968–1998 – *United System of Electronic Computing Machines* or *ES EVM* (compatible with IBM System/360 and IBM System/370). Lithuania was still occupied and could not buy foreign-made computing machines. PL/1 language subsets PL/$k$ with $k = 1, \ldots, 8$ were developed for teaching in Canada and the US (Holt *et al.*, 1977). Pascal was known



ETH EIDGENÖSSISCHE TECHNISCHE HOCHSCHULE ZÜRICH

Institut für Informatik

Clausiusstrasse 55
Telefon 01 326211

Postadresse:
Institut für Informatik
ETH-Zentrum
CH-8092 Zürich

Dr. Gintautas Grigas
Academy of Sciences of the
Lithuanian SSR
Institute of Mathematics
and Cybernetics
232600, Vilnius, 54
K. Požėlos st.

February 13, 1978

Dear Dr. Grigas,

I appreciate your interest in Pascal and in choosing a suitable subset for elemantary teaching. We have had such a subset in use for our programming courses for four years. I enclose a set of syntax diagrams. Our subset corresponds very closely to what you suggest. It unfortunately also omits sets, but includes the standard files input and output. A paper will be published in a forthcoming book entitled "Pascal - The Language and Its Implementation" by Wiley Inc., London.
We have not undertaken any further developments of Pascal. However, if you are interested in being informed on the subject, I very much recommend you to subscribe to the PASCAL - NEWS, ($ 4 per year). The editor is
Mr. Andrew B. Mickel
University of Minnesota
University Computer Center
227 Experimental Engineering Building
Minneapolis, Minn, 55455
USA

Sincerely yours,

N. Wirth

Prof. Niklaus Wirth

Fig. 1. A letter from Niklaus Wirth about Pascal subset for teaching in schools

very little in Lithuania at this time, and the computing machines produced and widely used in the Soviet Union did not have a Pascal compiler.

Thus, at that time, there was a fierce debate among scientists in Lithuania with respect to which language to use for teaching: PL/1 or Pascal. The circumstances favoured PL/1. It was agreed to start writing educational material and lessons to teach programming. However, as the lessons started to be written, the weaknesses of PL/1 and the strengths of Pascal gradually became apparent. After many discussions and a lot of experimentation in lesson writing, Pascal was adopted.

We initiated a communication with Niklaus Wirth asking for advice, and for the source code of the Pascal compiler, suitable for teaching. To our surprise, the answer to our letter came shortly afterwards (see Figure 1), and we received *Pascal-S*, a subset of Pascal's

compiler for students, written in the programming language. But to compile it, it was necessary to have a full-language compiler. The ES EVM computers did not have one, but there was a PL/1 language compiler. So we used the PL/1 language to develop the Pascal-S compiler.

While teaching the students, we noticed that the data types that were very useful for teaching programming and that were part of Pascal (enumeration and set data types) were not included in Pascal-S. We asked Wirth whether it would be appropriate to include these data types in the subset of Pascal we were designing, and he agreed to it.

The Pascal language was designed with the intention that structured programs written in it could not only be executed by computers but also comprehended easily by humans. Leveraging this philosophy, we utilized Pascal to install what we termed "programming culture" or style, encompassing algorithmic thinking and creativity. When assessing students' programs, we not only considered the outcomes produced by the computer but also evaluated the clarity and readability of the program text. Later, a relationship between programming style and errors was investigated (Grigas, 1995).

Furthermore, we took the initiative to co-author several exercise books aimed at teaching programming, in which a significant portion of the exercises comprised "reading" tasks (see Appendix A). These exercises were designed to enhance students' ability to interpret and understand existing programs, thereby fostering a deeper understanding of programming concepts.

Our goal was to bring computer programs closer to humans so that they are not just perceived as code that only the computer and author can understand. In Lithuanian, a program is not even called code. For example, the international project *Code Week* in Lithuanian is called "Programavimo savaitė" (Programming week). In this context, the concept of a correspondence school for young programmers was conceived. Subsequently, these objectives were introduced at a conference in Sofia, Bulgaria (Grigas, 1989), held simultaneously with the first *International Olympiad in Informatics* (IOI), and published in the journal *Informatics* (Grigas, 1990). We view programming not as an end in itself, but rather as an intermediary tool to foster creative thinking, intelligence, and work discipline.

The main goals and tools of teaching concerned with computers and informatics in school were the following: (a) Creative Thinking; (b) Problem Solving; (c) Programming; (d) Programming Languages; and (e) Computers and Software.

Each item may be considered as a goal. Each item but the first one may be considered as a tool to achieve the goal indicated by the preceding item.

A computer together with its operating system and other software may be considered as an ultimate goal or as a tool for all its applications (i.e., for the automation of different information processing tasks). It is a tool for the execution of the programming language commands. A language itself may be considered as an ultimate goal (by many teachers of programming, unfortunately) or as a tool for another goal – programming as such. Programming itself may be treated not only as a goal but also as a tool to achieve yet another goal – problem-solving or creative thinking.

The subject and its didactics may be built with any of these goals in mind, using the items following them as tools. The higher the goal the greater the possibility to develop the


Fig. 2. Textbooks and exercise books on how to use Pascal for teaching programming in schools

intellectual capabilities of the student; the lower the goal the more possibilities to convey empirical or technical knowledge.

Teaching was accompanied by experimentation. The training course was updated approximately every other year. But one thing remained constant throughout, namely the programming culture, which was assessed by analysing the texts of the programs written by the students. Such an assessment requires more effort from the teacher than checking the results obtained from the computer. However, the students learn to write more comprehensible program texts and to make fewer mistakes. The programs they create acquire a lasting value for the creation of other programs.

If students need to program in another language in the future, the programming culture inherited from Pascal will help them there, too. This means that the first programming skills acquired using Pascal are worth acquiring in any case.

We (the first two authors and a few other researchers) have authored several books focused on Pascal or using Pascal as the language for teaching programming (see Figure 2). In 1986, the inaugural informatics teaching journal was launched: a compact black-and-white booklet comprising roughly 100 pages (see Figure 3). Catering to both educators and inquisitive school students, the magazine circulated three issues annually until 2001, when it evolved into an international scientific journal titled *Informatics in Education*.

With the introduction of informatics, including programming, into all Lithuanian schools, the universal, free-for-all initial teaching of programming has lost its relevance. However, the experience has survived and is being passed on to schools and universities. The localization of Free Pascal and the Component Pascal compilers can be seen as a continuation of Lithuanian programming traditions.


Fig. 3. The first informatics didactics journal was established in Lithuania in 1986

Other activities include the *Bebras Challenge* (Dagienė and Futschek, 2008; Dagienė and Stupurienė, 2016; Dagienė and Sentance, 2016), aiming to promote informatics and computational thinking among school students all over the world. The Bebras Challenge has involved around 4 million students from nearly 70 countries by records of 2024.[1]

One more activity is localization, making the interface of software clear and understandable to humans, as well as all the texts and other resources.

We wrote here mainly about our work, and less about that of Niklaus Wirth. But if it had not been for his enormous contributions to the creation of Pascal and the teaching of programming, our work would not exist.

### 2.1. *Establishment of the Young Programmers' School by Correspondence*

From 1979 to 1981, a small group of researchers from the Institute of Mathematics and Informatics led by the second author organised programming lessons and invited 10 schools from different parts of Lithuania to take part in a teaching experiment.

On 27 January 1981, the Young Programmers School by Correspondence named *JPM* (acronym for "Jaunųjų programuotojų mokykla" in Lithuanian) was launched. So 1981 is treated as the foundation year of the JPM.

The Ministry of Education of Lithuania approved the teaching materials and tasks for publication in the most popular youth daily newspaper of that time ("Komjaunimo tiesa," Figure 4).

The JPM training course has been held every other year for more than 20 years. Although teaching programming in schools started later than in other countries (such as Poland, Estonia, Russia, and Canada), the interesting tasks attracted many eager learners, making

[1] https://www.bebras.org/statistics


Fig. 4. The daily newspaper in which the programming lessons were published several times per month from 1981 to 1985. The logo of the Young Programmers' School is visible on the left side: JPM – letters J, P, and M combined.

the JPM a popular institution in the country. This success was also due to the excellence of Wirth's Pascal.

Ordinary postal service (surface mail) was used for communication between teachers and school students. In the early 1980s, the JPM was the only educational institution enabling to get a primary acquaintance with algorithms and computer programming for most students in Lithuania, especially for those living in provinces.

To convey the foundations of contemporary programming methodology to the students of the JPM, theoretical knowledge is necessary as well. However, for most children, theory is less attractive than practical activities. Thus, the basic principles of the theory were delivered indirectly through problem solving. The set of programming problems was chosen in accordance with the requirements driven by good programming style and creativity (Grigas, 1990).

All the teaching materials of the JPM until 1993 consisted of several teaching chapters: (1) names, variables, values, assignment statements, and sequences of statements; (2) branches of actions; (3) repetitions of actions; (4) programs and their executing by computers; (5) logical values; (6) functions and procedures; (7) recursion; (8) discrete data types; (9) real numbers and records; (10) arrays; (11) programming methodology (i.e., style); and (12) program design.

A challenge in introducing students to programming was choosing integers as the primary data type in the first chapters, with real numbers introduced as late as possible. This sequence for teaching numerical data types was motivated by the following reasons: (1) the results of operations with data types of countable (discrete) values can always be exactly defined; (2) the arithmetic of integers is easily understood by younger students; and (3) integers are used in the majority of interesting problems. The Pascal operations `div` and `mod` are widely used in algorithms for such problems.

Since 1981, there have been significant changes in the way informatics is taught, particularly programming, driven by the increasing availability of computers in educational institutions and the introduction of informatics as a compulsory subject in secondary


ŠEŠTOJI PAMOKA

Fig. 5. The fragment of the newspaper with the sixth lesson introducing a loop

schools. These changes had a considerable effect on the aim and motivation of teaching at the JPM (Dagys and Klupšaitė, 1993; Dagienė, 1999).

The primary course of programming used to be published in the daily newspaper (see Figure 5), one lesson per week during four months (1981 January – May, 1983 September – December, and 1985 September – December). The lessons covered all the material in programming needed for beginners: text and tasks for self-control as well as certification tests. The JPM lessons in the newspaper were the only possible way to get primary acquaintance with computers and programming for many youngsters at that time.

The primary course of programming covered only four chapters of the JPM curriculum: all operations with integer values, assignment statements, a sequence of statements, and conditional and loop statements. Program execution by a computer was discussed as well. Students were taught to solve interesting and attractive problems by algorithmic approaches (Dagys, 1994). Tests were presented about once per month so that the students studying the first part of this course could carry out four or five tests in total.

Since 1986, informatics as a new subject has been introduced into all schools in Lithuania. All students have a chance to learn basic ideas of programming in the corresponding lessons. This led to some changes in the work of the JPM. The first part lost its scientific popularization component, because all the students learned about programming and computers in schools. There was no more need to publish programming lessons in the newspaper.

In the 1993/1994 school year, the JPM was reorganized once again (Dagys, 1994). The main reason for the reorganization was the above-mentioned increase in the number of computers in schools and the arranging of national *Olympiads in Informatics*. The studies were divided into two parts (courses): (1) constructs of algorithms and programming, and (2) methods of algorithms. The enrolment into the JPM School has become to be performed once a year.

Tasks in the first part were not difficult to solve. Reading (analysing) tasks made up a quarter of each student's homework. All the tasks aimed to develop reasonable thinking.

The first part of the course covered the whole curriculum of the JPM and consisted of five training chapters and a test: (1) algorithms, variables, and assignment statements; (2) control structures: conditional, compound, and loop statements; (3) functions and procedures, and recursion; (4) scalar (simple) data types; and (5) data structures. Homework for each chapter consisted of ten tasks: to write and analyse algorithms. The tasks to students and their solutions to teachers were sent by land mail. The solutions were analysed, mistakes were thoroughly indicated, and these solutions had to be returned to the students. Simultaneously the students received the homework task for the next chapter. The solution of each task consisted of three parts: (1) a verbal description of the solution idea; (2) an algorithm; and (3) an informal verification of the algorithm.

Since the second course of the JPM was meant for the most advanced students to teach them algorithms, they had to be selected in one way or another. The selection was supposed to be performed by a test that contained fifteen to twenty tasks. However, only half of them were randomly selected, and then reviewed and evaluated. Each task was scored 10 points independently of its complexity. A test was regarded as passed if it scored at least 75 % of the points. Only a successful student could continue his studies in the second course of the JPM.

The second part consisted of five homework tasks classified according to the nature of the algorithms' methods. Usually, there were five topics (they could be slightly different each year), e.g., (1) large numbers; (2) units of measurement (regular and irregular), number systems, and calendars; (3) searching for solutions, backtracking methods, and puzzles, (4) codes and ciphers, and finding and correcting mistakes in data; (5) data sorting; (6) dynamic programming; and (7) graph algorithms.

Three types of tasks were presented: to create an algorithm, to analyse an algorithm, and to develop a complete program. Five tasks were presented in one homework. The solution of each task consisted of two parts: 1) verbal description of an idea of solution, and 2) algorithm itself. Students who gained 60 % of the points were considered to finish the JPM successfully.

During the 34 years of the JPM's existence, over 7 000 school students were introduced to programming basics. The JPM is one of the long-existing schools with programming available for all school students. The main issues can be highlighted following the long experiences in teaching programming:

- Attention to the programming and algorithmic style as a part of information culture. Meaningful names of variables, procedures, and other objects are selected and commented.
- Introducing so-called reading tasks. Analysing algorithms is proven an important way of learning programming (Cheah, 2020; McGill and Volet, 1997; Nielson *et al.*, 1999; Robins *et al.*, 2003). By reading a task, a more complicated algorithm can be introduced.
- Priority is given to tasks that require creativity in programming.
- Teaching programming rather than a programming language. The latter is just a tool for writing programs, not a teaching goal.

The School of Young Programmers triggered several scientific and methodological articles, books, contests, and other activities. The team at the research institute was enthu-

siastically inspired by the idea of contributing to computing education at schools despite the lack of even the most elementary resources: in the entire institute there was only one typewriter with Lithuanian font, and a huge electronic computing machine type VS EVM without terminals; the timetable to use this machine was strictly enforced, a researcher could get access to the computing machine for about two hours weekly.

### 2.2. *Development of Pascal Compilers*

The Pascal subset compiler we developed was intended for beginning programmers who might be presenting their work to a computer for the first time. The compiler provided many error messages with explanations, printed the original data and the results side by side, and corrected proofreading errors found during syntactic analysis. There were even some features implemented, such as that if the value to be assigned to a variable could not be calculated, the compiler would assign a default value and continue executing the program. This was done with the aim of detecting as many errors as possible (computer execution time was expensive). Such errors and their correction were reported to the program author.

In the late 1980s, when the JPM was being developed, ES EVM computers were common. As mentioned above, they did not have a Pascal compiler, but we needed one for the JPM. Therefore, a team at the Institute of Mathematics and Informatics took it upon themselves to develop a Pascal compiler for teaching purposes.

The capabilities of the ES EVM computing machines were not great. In consultation with Niklaus Wirth, we decided to develop a Pascal subset compiler incorporating the elements needed for teaching programming. To reduce the workload, we decided to make an interpreting compiler consisting of two parts: a compiler that translates the program from Pascal into an intermediate language and an interpreter that executes the intermediate language. The compiler was implemented by the two Vilnius University students Alma Baliūnaitė (Petrauskienė) and Valentina Piekaitė (Dagienė), under the guidance of Gintautas Grigas, and the interpreter by the two students Remigijus Petrauskas and Arvydas Salda, under the direction of Vladas Tumasonis. We wrote the compiler in PL/1, but decided to program the interpreter in Assembler, fearing that the PL/1 language would be too slow. Programming in Assembler was much more complicated and time-consuming. We implemented the interpreter in PL/1 for a temporary job and found that it was fast enough and there was no longer any temporariness.

The compiler has been customized to cater primarily to beginner programmers but remains suitable throughout the entire learning process. In comparison to Pascal S, it included features such as set and enumeration types, and it provided sample information essential for novice programmers venturing into programming and computing for the first time. Furthermore, the compiler displayed input data and corresponding results side by side, enabling students to discern the relationship between them. Additionally, the compiler not only detected errors in the program but also attempted to rectify many of them (see Figure 6).

The first computers received by Lithuanian schools were BK-0010. These machines did not have a monitor and came with a mini-TV. They had only 64 kB of RAM, 48 kB

Fig. 6. Explanations: (1) The specific letters of the Lithuanian alphabet were obtained by printing a letter without a diacritical mark and adding a mark similar to a diacritical mark (an apostrophe or a comma) in its place. (2) An unlimited number of programs can be given to the computer in one task. (3) Headings. (4) Print not only the final data (results) but also the original data so that the student can see from which original data the results were derived when looking for errors. (5) At the beginning and end of the original data set, the rows of asterisks identifying the columns of the perforator are printed, so that if students need to correct the data, they can see which columns of the perforator have been punctured. (6) The compiler reports the error detected, corrects it, and looks for new errors. (7) The maximum number of errors to be searched can be set. (8) After the first error is detected, the values of the variables at the time are printed; this may help to find the cause of the error. (9) The compiler message that the error may have been corrected incorrectly and may have caused secondary errors. (10) Undefined values of variables that were found.

of which were used for the operating system. So only 16 kB remained available, and it was a challenge to fit the compiler and its necessary ancillaries into this space. However, Albertas Dinda, a teacher at the *Zigmas Žemaitis Secondary School* in the small town Švenčionys, decided to make such a compiler freely available. Only a small subset of Pascal was indeed realized (Dinda, 1990). There were only integer, Boolean, and char data

types. And there were quantitative restrictions: the maximum number of names was 16, the total maximum number of characters used in symbolic constants was 256, the Write and Writeln procedures could not specify formats (a fixed number of positions was allocated: integer – 18, char – 1, Boolean – 6), and there was a minimal program writer (editor). All this was called *MIKROPAS* (a combination of MIKRO and Pascal).

Although the compiler was very limited, it was sufficient for an introductory 13-lesson course (Dagienė, 1989), based on the newspaper lessons of the JPM published in 1981, 1983, and 1985. This was achieved according to the following principle: to put into the product (in this case, the training course and the compiler) all that is needed and nothing that is not.

In 2001, Component Pascal was adopted in Lithuania, a component building system *BlackBox* with a Component Pascal compiler was localized, and a teaching module was developed and suggested for schools (Grigas and Jevsikova, 2001).

Component Pascal, building upon the traditions of Pascal, Modula-2, and Oberon, emerged as a versatile language with a broad scope (Oberon Microsystems, 2001). Its key features encompass block structure, modularity, separate compilation, robust static typing coupled with comprehensive type checking extending across module boundaries, type extension facilitated by methods, dynamic module loading, and integrated garbage collection. Through type extension, Component Pascal adopts an object-oriented approach where objects represent variables of abstract data types containing private data and associated procedures. Abstract data types are defined as extensible records. With a focus on complete type safety and a dynamic object model, Component Pascal positions itself as a language that can be used for component-based development.

Close to the Pascal programming language used in schools, it incorporated modern features for learning object-oriented and component programming. In the Lithuanian localization, the Component Pascal system brought big advantages for teaching and learning purposes due to the ability to use all Lithuanian alphabet letters (including those with diacritic marks) in the names of variables and other objects. This feature made algorithms even closer to human language by the ability to use mnemonic names (Jevsikova, 2007).

A Free Pascal compiler was localized into Lithuanian and methodological materials developed (Institute of Mathematics and Informatics, 2004). The localized system supported the use of the entire Lithuanian alphabet allowing for a natural choice of names (identifiers) in the program.

### 2.3. *Short Context-Based Reading Tasks*

When teaching algorithms, we prioritized cultivating an algorithmic style. Our focus was on presenting algorithms in a clear and structured manner, ensuring that their texts were well-layed-out and accompanied by appropriate comments to enhance readability. By adopting this approach, we sought to influence the broader education of school students. While others primarily taught how to write algorithms, we took a novel approach by emphasizing the importance of learning how to read them.

The selection of tasks at the distance education school is very important: they must cover as many theoretical problems as possible, teach students algorithms and programming

methods, and what is most important, allow them to acquire the skills to use them. Modern computing concepts need to be conveyed to students in an attractive way, what motivated us to create and adapt interesting tasks from a wide range of fields. The main goal was to attract students' attention with intriguing tasks reflecting different areas of life and science, and to encourage them to solve these tasks in order to learn to program.

While developing the methodology of teaching programming concepts and algorithms for the JPM, we have raised the principle that it is of high importance to choose and classify the sets of tasks that would actualize the purposes of both teaching programming/algorithms and pedagogics. To foster programming skills, we needed well-developed short tasks on computing concepts that aim to develop algorithmic thinking. A task should have an easy-to-understand problem statement, and at the same time be complex enough to allow creativity in the solution process.

The tasks must be selected carefully, considering the different aspects of each problem. Two large groups of problems were distinguished: (1) so-called *reading tasks* (for analysing the text of the program/algorithm and understanding the main ideas behind the solution) and (2) *writing tasks* or program/algorithm development tasks.

Some tasks connected with programming and algorithms are similar to mathematical problems. However, the solutions to traditional mathematical problems are usually short and written in a single way; that is why it is not necessary to analyse here how the solutions should be written. The constructs of programming languages are often longer than the mathematical ones and have a great variety of means of expression. Therefore, it is not easy for a student to develop an algorithm.

One of the most comfortable and effective ways of learning algorithms is to learn to read and analyse algorithms written by others. The main issue in the presentation of algorithm reading tasks is to avoid so-called *passive reading*. Therefore, it should be demanded to accomplish certain exercises during the reading.

The reading tasks involve analysing and performing various small tasks, such as finding errors, filling in blanks, modifying existing text, determining whether multiple algorithms solve the same problem, and so on. The reading tasks are particularly beneficial for novice programmers who do not yet possess algorithmic writing skills.

Programming teaching materials were enriched with reading tasks, and most of the programming manuals we produced at that time had separate chapters on reading tasks. For example, our book of programming exercises (Dagienė and Grigas, 1992) has 178 reading tasks. Let us give a few examples.

*Exercise 1.* To calculate the time displayed after one minute, the following procedure is used. Insert the missing line in place of the ellipses (the "...").

```
1  procedure time(h, min: integer; var h1, min1: integer);
2  begin
3     min1 := min + 1;
4     h1 := (h + min1 div 60) mod 24;
5     ...
6  end;
```

*Exercise 2.* The lengths of the sides of a triangle, denoted by a, b, and c, are known. A function is then written to verify if the triangle is equilateral. Simplify the function by eliminating a component from the logical expression.

```
1  function triangle(a, b, c: integer) : boolean;
2  begin
3     triangle := (a = b) and (b = c) and (a = c)
4  end;
```

*Exercise 3.* A program has been developed to print the prime factors of a given number. The program is intended to print the prime factors of the given number. However, it contains errors. Correct them by replacing the two reserved words.

```
1  program factors;
2     var a, k: integer;
3  begin
4     read(a);
5     for k := 2 to a do
6        if a div k = 0 then
7           begin
8              write(k);
9              a := a mod k
10          end
11 end.
```

## 2.4. *Comparison of Pascal with Other Programming Languages for Education*

Pascal was designed to teach programming. Its constructs are simple, easy to remember, and Pascal programs are easy to read. The language has been widely used in schools, programming clubs, extra-curricular studies, contests, and informatics olympiads. C++ was developed for the industry. The language has tools for developing large scale programs, such as the Windows and Linux operating systems. In short, Pascal has elements of an algorithmic language, C++ has elements of a computing language.

Over time, C++ has been increasingly used in schools. Some education policymakers thought that if you were going to be programming in C++ in the future, you did not need to learn Pascal. However, it is not about learning a programming language, but about understanding programming concepts. In Pascal, programming constructs are clear and systematic, making it much easier for a novice to understand and learn them.

For many years, the IOI has used several languages: Pascal, C++, and Basic. A detailed analysis of how students performed in these languages was carried out in the sixth International Olympiad, involving students from 49 countries. The data on the programs written by the students and their evaluation are described in detail by Grigas (1995). The success of students, measured by the number of medals, is analysed by a number of factors: usage of programming language, length of program, usage of functions and procedures, character density, line density, usage of particular reserved words, etc. Let us use these data to compare student success using Pascal and C++.

Table 1
Number of medals awarded depending on the programming language chosen

| Programming language | Number of participants | Relative number of medals | Relative average of medals |
|---|---|---|---|
| Pascal | 136 | 156 | 1.15 |
| C++ | 27 | 26 | 0.96 |
| Basic | 13 | 0 | 0 |
| Total | 176 | 182 | |

The number of medals received is a natural indicator of the success of participants in the Olympiad. The number of bronze ($b$), silver ($s$), and gold ($g$) medals we converted into the relative number of medals $m$ using the formula $m = b + 2s + 4g$. Dividing $m$ by the number of participants gives the average relative number of medals per participant shown in Table 1.

The difference between Pascal and C++ is 0.19 in favour of Pascal. The interpretation of these findings is subjective in nature. Therefore, we conclude our analysis here and leave it to the reader for further consideration.

## 3. Methodology

In this section, we describe the methodology used for the qualitative study, which has been performed to delve into the memories and insights of the experts involved in programming education development at early stages by utilizing semi-structured interviews with open-ended questions. The data collected via the interviews was processed by applying an inductive thematic analysis approach. Descriptive statistical methods and correlation analysis were used to analyse the outcomes derived from the qualitative data.

### 3.1. *Instrument, Participants, and Data Collection*

The experts for the interviews have been selected to cover the widest range of experiences and perspectives with respect to the early stages of programming education in Lithuanian schools, contributing to the richness of the collected data. The selected experts represent three groups:

1. *Pioneers (4 experts)*, i.e., teachers, who started during the first years of informatics introduction in schools, who contributed to first initiatives to teach programming.
2. *JPM teachers (4 experts)*, who taught in the JPM school.
3. *Younger generation teachers (3 experts)*, who have worked with Pascal in secondary schools.

In total, 11 experts were interviewed. They are denoted by the following codes: the first two letters denote a group (PE – teachers-pioneers, JPM – teachers in the Young Programmers' School, and YT for younger-generation teachers) and a sequence number, e.g., 03, after a dash.

The interviewed teachers represent different cases in terms of their age, background, and teaching experience.

The instrument for data collection included four open-ended questions, contributing to the two research questions posed in this paper:

1. Write about yourself and your experience of teaching programming.
2. What ideas from the origins or early stage of programming education are the most important and innovative, in your opinion?
3. What factors, do you think, have contributed to the emergence and establishment of programming education in schools?
4. What features of Pascal would you identify as the most significant for teaching programming?

The selected experts shared their perceptions and insights with respect to the interview questions between winter and spring 2024. The interviews were conducted in written form by contacting the experts by e-mail or phone. The participation of the experts was on a written informed consent basis. They were informed about the aim of the research being conducted, and each of them agreed to provide answers to the interview questions to be analysed in an anonymized way.

On average, the text of the answers to the interview questions was 7636 characters / 986 words per interview.

### 3.2. *Data Analysis*

The interview data underwent thorough qualitative analysis involving coding and categorization. The primary analytical approach was thematic analysis as a method for identifying, analysing, and reporting patterns (themes) within qualitative data (Braun and Clarke, 2006). Thematic analysis not only aims to summarize the data but also to identify and interpret key aspects of it, guided by the study's research questions (Clarke and Braun, 2017). After the authors familiarised themselves with the interview texts, a coding process was performed to identify codes and categorize themes. The interview data of this study was analysed using an inductive approach to thematic analysis, i.e., a bottom-up data-driven analysis (Clarke and Braun, 2017).

The coding process was iterative (Byrne, 2022): after the first usage of the code, all interviews were re-coded to use the best of the codes that emerged from the data. First, codes were organized into codes and sub-codes and categorized according to the three interview questions. Then, the codes were reorganized according to the patterns identified in the data and the themes were derived. To deal with subjectivity, the organization of codes and themes was discussed by three experts. To ensure effectiveness and reliability in the data analysis, the coding was performed using the MAXQDA software tools.

## 4. Results of the Qualitative Study

### 4.1. *Early Innovative Practices*

Regarding the second question of the interview (the first research question of our study), the experts identified several most important early innovative practices in programming

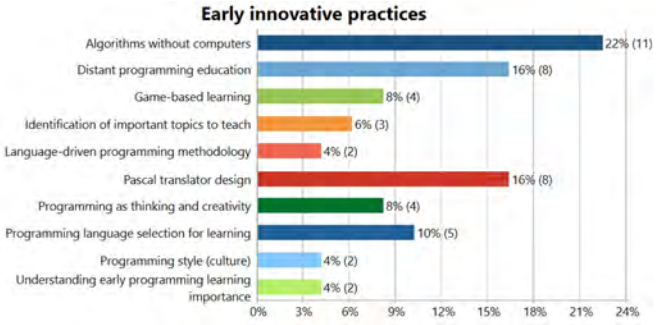



Fig. 7. Early innovative practices in programming education, as identified by experts, frequency, % (number of segments)

education. Figure 7 presents denoting codes, segments and frequencies of mentions). Then, the meaning of the codes, based on experts' interview texts, are described.

Experts most frequently mentioned *algorithms without computers* as an innovative idea during the early stages of programming education development. This idea was identified by all groups of experts. "The strangest and probably most innovative idea was to program on paper, without any computer that could show syntax errors, structure of the code, execute the algorithm with different initial values, and allow step-by-step execution with the mapping of intermediate variable values. [...] We checked the algorithms and looked for errors by running them in our heads." (PE-04). The experts mention that algorithms or programming without a computer puts more emphasis on thinking and the algorithm itself: "It was in the early stages that I really enjoyed programming without a computer. With a computer, the focus is more on the mechanical work of the computer rather than on the beauty of the algorithm and its logic. The emphasis is more on abstract thinking." (PE-03).

This idea, popular nowadays, too, appeared naturally due to the lack of computer availability and provided many advantages in the development of algorithmic thinking of students. The idea is related to another innovative idea during the early stages, identified by experts: *programming as thinking and creativity*. The experts stress that algorithmic thinking helps people in all areas of life. Most of the experts mention creativity in programming, e.g.: "Programming is first and foremost about creativity" (PE-02), and thinking: "The important thing was to focus on thinking, on the understanding of data structures and their selection, rather than on finding a way to write a functioning program faster." (JPM-03).

The above-mentioned innovative ideas are closely related to *game-based learning*, also identified by experts as an important innovative idea that appeared at the early stages. The experts recall that in the JPM, during face-to-face activities organized for students, there were game-based algorithmic activities for students adopted as early as 1983. The activity also involved collaborative learning. One year later, the idea was implemented as a competition for the groups of participants: "In 1983, in Užupis, in the pouring rain, Kęstutis taught a game I had never seen before. It was so interesting that while we were playing, we thought – what if next year we could organize a programming competition between groups? Each group would create a function that chooses the next move of the

game, and we would create a program that organizes a tournament of the functions in the game: each function competes against the other twice. One time one function makes the first move of the game, and the second time the other. The winner gets a point. The group whose function wins the tournament is the winner and is rewarded." (JPM-02).

"The distant mode of teaching (in the JPM) did not directly test students' theoretical knowledge, but instead assessed their use of programming constructs and techniques to solve puzzle-like tasks." (JPM-04). These types of activities later evolved into well-known international initiatives, such as the Bebras Challenge.

The second most frequently mentioned innovative idea was the *Pascal compiler*, designed in Lithuania: "When Albertas Dinda created and distributed the Pascal language compiler, programming education went to a much higher level." (PE-02). During the various stages of the design of the compiler, the main principles that should help it to fit into the small memory of the BK-0010 were formulated (PE-04), and included reliability, error logging, translation until the first error, avoiding duplication, using existing libraries on the computer, saving RAM, and others. Since they were not cost-effective, long integers were implemented since they were not cost-effective. This was a compiler, not an interpreter, with an integrated working environment, which was also an important idea. The compiler mentioned by the expert is described in more detail in Section 2.2.

The third most mentioned innovative idea were early initiatives of *distance programming education*, which refers to the JPM analysed in Section 2.1. Distance programming education initiatives included *distance education through correspondence*: "Students solved tasks published in the popular newspaper." (PE-02), and distance education through *television broadcasts* that were related to the JPM activities and broadcast programming lessons (YT-02). "Students who completed an interesting task were invited to appear on a TV show to present it." (PE-02). As JPM-04 remembers, "In 1986–1988 [...], I hosted the Lithuanian TV program 'Informatics and Computing Technology,' and in the studio, I had to interview almost impromptu (after a short rehearsal) the students who had been invited to the program, or [...] the teachers who had brought them on board." PE-03 remembers these initiatives as "one of the most exciting adventures of my life."

*Programming language selection for learning* was identified as an important innovative idea: "An important idea was the choice of the programming language for teaching." (PE-04). Experts stress that "When teaching programming to beginners, what matters is not a powerful language, but a language that is suitable for teaching and has the necessary features." (JPM-03). The selection of a programming language was driven by the JPM activities, and the selected language was Pascal. This choice is regarded as the best decision at that time: "Selecting Pascal has helped for the JPM teaching methodologies to clarify programming concepts that are less obvious in other programming languages: data types, loops, hierarchical structures, functions, and procedures." (PE-04). Choosing Pascal for teaching and learning programming contributed to the *language-driven programming methodology* development. One former teacher in the JPM, working now in programming education and students' preparation for the Olympiads in Informatics (JPM-03), reflects: "It was my first programming language, which not only inspired my understanding of programming principles, but also helped shape my methodological approach to the field."

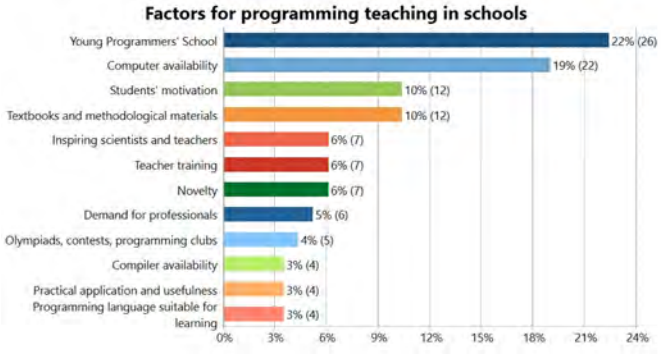



Fig. 8. Factors for the emergence and establishment of programming education in schools, as identified by experts, frequency, % (number of segments)

Innovative ideas that were developed in teaching programming later in schools, such as *programming style (culture)*, also originate from selecting Pascal and the work of the JPM (JPM-01, JPM-03).

Language selection, JPM work, compiler design, and all the initiatives taken together contribute to the *identification of important topics to teach*. The most attractive for students were the ideas of introducing recursion (PE-01), computer graphics, and 3D design (YT-01). It was identified that it is important to teach algorithmic thinking, programming, computation, text processing, computer graphics, and website design, which was included in the informatics course in schools (PE-05).

Society understood the *importance of the early introduction to programming* in education, and this was also one of the innovative ideas of the initial period. Computers were becoming an important part of life and it was important to get the right qualifications as early as possible, starting from the school bench (PE-04). "The earlier children start to learn to program, the better their performance in olympiads and competitions, the better their academic performance, and the easier it is for them to enter the world of work." (PE-02).

### 4.2. *Factors for the Emergence and Establishment of Programming Education in Schools*

As for the third interview question and second research question of this study, there were the following main factors for the appearance and establishment of programming education in schools identified by the experts (see Figure 8).

The most frequently mentioned factor is the success of the JPM (see Section 2.1): "The success of the Young Programmers' School, which shows that students are receptive to programming, and whose graduates outperform experienced programmers in programming competitions, is, I believe, the success factor that led to the emergence of teaching programming as part of general education." (JPM-02).

This school attracted especially those students who were interested in maths and the novelties that were coming along with computers. It was the only way to help those students interested in programming at that time, providing support and methodological

materials (PE-03). Later, those students took part in the Olympiad of Informatics (JPM-01). The experts mention the great impact of the school on *students' motivation* to learn to program (11 mentions by 7 experts in all groups). A valuable role in supporting students' motivation was the JPM summer camp, where students could meet with each other, see their teachers in person, solve engaging tasks, and be involved in community-building activities. As mentioned by most of the experts, some of those experts had studied in this school themselves (7 mentions). Students' motivation was one of the main factors for programming education establishment in schools, and mostly related to the JPM activities.

Besides students' motivation, *teacher training* was essential and evolving alongside. When talking about the effect of the JPM factor, most experts usually mention the JPM as an informatics teacher training activity (5 mentions): "The JPM also produced an entire generation of teachers who were able to code. This made programming easy to get into informatics in schools." (PE-01). Teachers, involved in the JPM learned themselves, prepared methodological materials, and valued it as an important practice in their career (JPM-01–04). Respondents also mentioned a lot of training activities for teachers during the first years of the informatics school subject: compulsory monthly seminars for teachers (YT-02), "When the subject of informatics was introduced into Lithuanian schools, I had to attend many different qualification courses." (PE-02).

The second most frequently mentioned factor for programming introduction in school was *computer availability*: "I think that the most important factor that has helped the teaching of programming at school is the rapid development and availability of computers, the rapid and widespread diffusion of computers in all areas, and the cost of computers falling drastically." (PE-03). In the early stages, "It was easy to show the advantages of programming over a calculator." (PE-01). With the proliferation of computers, it has become possible to run solutions on the computer and discover even more interesting aspects of the tasks (JPM-3). First, the appearance of computers stimulated the process of learning and teaching programming. Second, the limitations of the first available computers free from a variety of software has also led to learning programming. Computers appeared in schools in 1987 (PE-01, PE-02), and "the first computers in our schools were practically 'bare' if we ignore BASIC or FOCAL." (PE-01). There was a natural need to program. Some experts also compare this to later processes: "When IT became an everyday item on almost everyone's desk and in every pocket, the reverse happened." (PE-01).

The experts also mention the *compiler availability* factor. First, locally developed compilers were used, especially for teaching in the JPM. Later, computers with Pascal compilers came to schools, and programming teaching and learning advanced to the next level. "In addition to the operating system, utilities, and the programs I have already mentioned, the DVK-2M also included a word processor [...] and, much to my delight, a Pascal language compiler. The most skilled student could sit down at the computer and program." (PE-01). Turbo Pascal came to schools with IBM computers (PE-02). As YT-01 recalls, "The user-friendly and modern Borland programming environment and step-by-step execution capabilities have contributed greatly to its popularity. And of course, the speed of compilation and execution was also very important. Computers were slower...."

The *novelty* of the technologies and ideas was one more important factor mentioned by the experts (7 mentions): "All sorts of innovations have appeared, if not every week, then

every month or year. The later it got, the more innovations appeared." (PT-01). *Practical application and usefulness* attracted students and teachers: "Children were very interested in programming, because they saw the practical benefits in various fields – computing, information modelling, moving image creation, and others." (PE-01). As a related factor, experts mention the natural *demand for professionals* (6 mentions by 4 experts) as contributing to the establishment of programming education in schools: "With the awareness of the use of computers in business and institutions, there was a growing demand for people who know how to use them (programmers)." (JPM-02).

The pioneers mention the lack of literature at the early development stages. However, due to the efforts of research enthusiasts, the *textbooks and methodological materials* appeared rapidly in the national language, and supported the teachers as well as students. The experts mention early textbooks on the principles of the compiler, a book on formal languages (PE-01, YT-01). With the JPM activities, the methodological materials for teachers and students became available (PE-03). There were textbooks translated into Lithuanian, by JPM teachers and researchers, e.g., "I translated (from Russian) the second part of the textbook 'Fundamentals of Informatics and Computing' and the second part of the teacher's book of the same name (1987), which, with the consent of the Ministry of Education at the time, also included a chapter on Pascal, which was not in the Russian original [...] I also translated from English [...] a book for educators 'Computers in School' (1989, which was ahead of our reality in terms of its content [...]." (JPM-04). The periodical journal "Informatika" contained mostly programming teaching materials (JPM-04, YT-01). The fundamental nature of the textbooks and teaching materials is reflected by the younger-generation teacher's response (YT-02): "In 1991, the textbook was published by Dagienė Valentina and Gintautas Grigas: 'Informatics, Tentative teaching material for grades X-XII.' A year later, in 1992, the same authors also published a programming task book. These two tools can still be referred to today, and they have not lost their foundations for informatics education in school."

The establishment and successful development of programming education are supported by various initiatives, being organized in the early stages and today: *olympiads, contests, and programming clubs* (5 mentions by 4 experts).

The success of programming education in schools was based on the *selection of a programming language suitable for learning* (PE-02, PE-03, JPM-04, YT-03). Pascal was chosen and used for teaching since 1979 in the JPM (PE-03, JPM-04). "Perhaps most importantly, Gintautas Grigas' foresight led to the choice of Pascal for both the JPM and later for the general education school for teaching the informatics course." (JPM-04). This quotation shows the relationship with the next factor.

The early initiatives, including the JPM work, language selection, compiler design, preparation of teaching materials, and spreading ideas for establishing teaching programming in schools would have been impossible without *inspiring scientists and teachers*. This factor is mentioned by 6 experts (7 mentions).

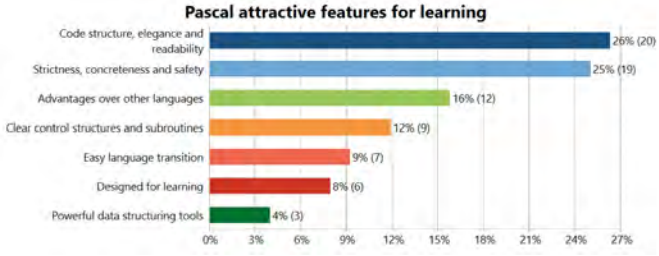



Fig. 9. Most attractive features of Pascal for teaching programming, as identified by experts, frequency, % (number of segments)

### 4.3. *Pascal's Most Significant Features for Teaching Programming*

As mentioned in previous subsections, the selection of a suitable programming language for learning (Pascal) was one of the innovative practices and important factors for informatics becoming part of general education.

The main features the experts identify as most important for teaching programming are presented in Figure 9.

Pascal is valued for *code structure, elegance, and readability*. These features were mentioned by all experts. "The structure of Pascal programs is elegant." (JPM-02), and strictly hierarchical (PE-04). "These elements make it easier and quicker to understand the structure and operation of algorithms. They also make the code easier to read and understand." (PE-04). "The clear structure of a program written in Pascal makes it easier to analyse, debug, and improve it (a feature that was used in 1982 to prepare students for the IOI when they were given programs that had just been written by their peers and were asked to review and improve them, i.e., to fix them so that they would work properly)." (JPM-04). The readability of the written program was related also to compliance with elementary logic, including reserved words, operation symbols, etc. (JPM-02). Programs written in Pascal were close to human language (PE-03), and thus students in baccalaureate programs nowadays are taught to write programs in pseudocode that is close to Pascal (JPM-03).

The above-mentioned important feature is tightly related to others identified by the experts: *strictness, concreteness, and safety*. "Pascal is strict and concrete. I liked these features, especially when learning with pen and paper." (PE-01). Strict syntax and semantics help to avoid programming errors (PE-01, PE-02, PE-04, JPM-02). Error messages were clear and logical (PE-01). Safety is also ensured by the important feature of strict data type separation, as mentioned by the majority of the experts, e.g., "definition of variable data types (without the 'default' types based on the first letter of the variable name)" (JPM-04). The local and global variables are separated, and the variable definition section is clearly defined (PE-01, JPM-02).

Pascal is characterized by *clear control structures and subroutines*. This includes structured conditional statements (PE-04), loops of different types (PE-04), separation of functions and procedures, and the possibility of nesting them one into another (PE-01, PE-04, JPM-02).

Table 2
Code organization according to the themes

| **Society readiness and technological advancements** | **Early innovative national initiatives** |
|---|---|
| Computer availability | Young Programmers' School |
| Students' motivation | Textbooks and methodological materials |
| Inspiring scientists and teachers | Pascal compiler design |
| Novelty | Teacher training |
| Demand for professionals | Olympiads, contests, programming clubs |
| Compiler availability | Programming language selection for learning |
| Practical application and usefulness | |
| **Programming methodological approaches** | **Pascal attractive features for learning** |
| Algorithms without computers | Code structure, elegance and readability |
| Distance programming education | Strictness, concreteness, and safety |
| Programming as thinking and creativity | Advantages over other languages |
| Game-based learning | Clear control structures and subroutines |
| Identification of important topics to teach | Easy language transition |
| Language-driven programming methodology | Designed for learning |
| Understanding early programming learning importance | Powerful data structuring tools |
| Programming style (culture) | |

The experts state that Pascal had many *advantages over other programming languages* available at that time: "Pascal was very suitable, because it was very modern, new, and had many good features, outperforming FORTRAN, BASIC, and FOCAL." (PE-01). C, which appeared almost at the same time, was close to Assembler (PE-01). "Assembler or Fortran were not appropriate languages for teaching programming concepts." (PE-04). The experts mention advantages of Pascal over languages, which are most popular today, e.g., "Several features of Pascal that are not found in popular modern programming languages (C++, Java, C#, Javascript/TypeScript, PHP, or Python) stand out." (JPM-02). JPM-02 and PE-01 list these main features as assignment statement, code structure, data type separation, variable declaration part, and separation of functions and procedures. "For example, Python has no data type control; the same variable can take on completely different types in the same program." (JPM-03, the same stated by PE-01, JPM-02).

*Easy language transition*, as stated by 6 experts, is an important feature of Pascal to be the first language for learning, e.g., "However, once you have learned the principles of writing Pascal-based code, it is easy to switch to most other programming languages." (PE-04). Even contemporary popular programming languages, such as Python, are suggested to be the second programming language after Pascal (PE-01). This easy transition is possible,

Table 3
Pearson's correlations between the four thematic variables

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | – | | | |
| 2 | 0.370 ($p = 0.1314$) | – | | |
| 3 | **0.782** ($p = 0.0022$) | **0.739** ($p = 0.0047$) | – | |
| 4 | **0.611** ($p = 0.0230$) | **0.571** ($p = 0.0332$) | **0.677** ($p = 0.0111$) | – |

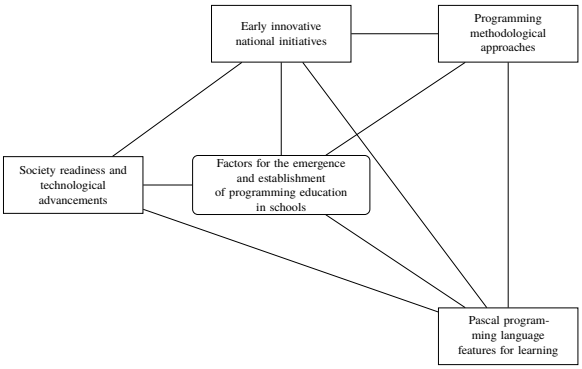



Fig. 10. The thematic model

because Pascal was *designed for learning*. "Pascal was designed for teaching, so each time the student had to make, for example, a 'queue,' a 'stack,' or a 'list' himself." (PE-01). This explains the presence of *powerful data structuring tools*. "Pascal had powerful data structuring tools that allowed us to describe practically any data structure." (PE-01).

### 4.4. *Thematic Model: Interrelationships Between Groups of Factors*

In the previous subsection, the codes derived from the expert interviews were presented according to the three key interview questions. However, they are interrelated with each other. Codes, restructured according to the themes

1. Society readiness and technological advancements;
2. Programming methodological approaches;
3. Early innovative national initiatives; and
4. Pascal's attractive features for learning

are presented in Table 2. The themes of were derived around research question RQ2, which relates to the factors that contributed to the emergence and establishment of programming education in schools. The correlation table presents the Pearson correlation between variables corresponding to the themes, $p$-value: 1-tailed, with significant correlations marked in bold (see Table 3). In the scheme shown in Figure 10 the main themes and their interrelations are presented.

All the categories examined are structured around RQ2. All other questions of the interviews supported this question and provided opportunities to look for relationships. Early national innovative initiatives were related to and fostered by technological advancement and society readiness, including the interest and motivation of teachers and students, inspired by enthusiastic scientists, novelty, capabilities and applicability of computers, and demand for professionals. This supported the introduction of innovative national initiatives like the JPM, the design of the original Pascal compiler, and the preparation of textbooks, exercise books, and other methodological materials for programming education, in parallel with teacher training and informatics olympiads, contests, and other activities. One of the

important factors was programming language selection, which is related to the theme of the Pascal features for learning. Innovative national initiatives and Pascal language features are related to the methodological approaches that appeared in the initial stages and are still relevant today, e.g., algorithms without computers, distance programming education, and game-based learning. Pascal drove the formation process of programming methodology. Focussing on algorithms in programming was possible due to the features of Pascal, such as structure, strictness, elegance and readability of the program, ability to structure data, and others.

## 5. Conclusion and Discussion

The impact of Niklaus Wirth, celebrated (e.g., by receiving the Turing award) for his development of Pascal and other programming languages, has been essential in shaping the landscape of programming education in schools globally. As nations have pursued their individual paths of exploration, this led to shifts in cognitive paradigms.

Significant societal changes seldom unfold abruptly, emerging not on a single day or within a specific year, but rather through a gradual process. Ideas take root, conversations ensue, and concepts evolve over time, maturing through discussion and debate before potential implementation. Only then do tangible outcomes begin to manifest, though not always in predictable ways. The same incremental progression can be observed in the evolution of programming education within Lithuanian schools.

An important role in shaping the methodology for teaching programming was undertaken by a group of scientists from the Institute of Mathematics and Informatics, now known as the Data Science and Information Technologies Institute of Vilnius University. In 1979, they designed comprehensive instructional materials, comprising tasks, exercises, tests, and corresponding answers, aimed at facilitating programming education. Subsequently, the Ministry of Education of Lithuania endorsed the pilot implementation of this methodology in ten schools. Thus, in 1981, the JPM was inaugurated, marking a significant milestone in programming education.

The scientists at the institute developed a Pascal compiler specifically tailored for the Russian electronic computing machines ES EVM. This compiler was designed with a focus on learning, enabling pupils and teachers to input their initial programs and receive comprehensive feedback on the computer's execution process and any identified errors. Notably, the compiler was equipped to automatically correct certain mistakes and provide detailed reports on the changes made. This functionality was particularly crucial due to limited access to the computer system and the sizable queue of programs awaiting execution, which often resulted in lengthy delays in obtaining results, sometimes spanning an entire day. Additionally, in response to a request from the Petropavlovsk Pedagogical Institute in Kazakhstan, the compiler was adapted to support the Russian language (Dagienė *et al.*, 1983). This may well have marked the first case of a Lithuanian software being translated into another language.

The establishment of the JPM sparked a surge in scientific and methodological publications, books, contests, and competitions. Despite our inspiring work, we faced significant

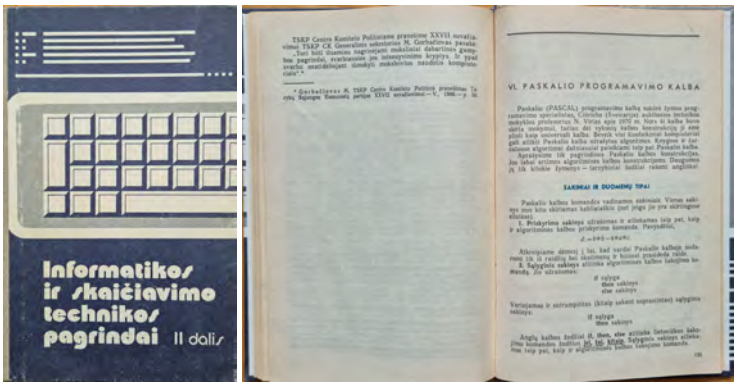



Fig. 11. The first informatics textbook for secondary school, with a chapter on Pascal

challenges due to the scarcity of even the most basic resources. For instance, the entire institute had only one typewriter with Lithuanian font, and computing machines were few and far between. Access to the mainframe computer was strictly regulated, with scientists allotted a mere two hours per week. Nevertheless, despite these limitations, our group of dedicated scientists pioneered an environment conducive to the development of informatics education.

Lithuania's pioneering work in programming methodology for schools garnered recognition throughout the Soviet Union in the late 1970s and early 1980s. The Soviet Union introduced mandatory programming education in schools in 1985. During this period, we not only translated the first textbook *The Basics of Informatics and Computing Techniques* into Lithuanian, but also included a chapter on Pascal (see Figure 11), a programming language that had not yet been thoroughly evaluated by Russian scientists at the time. This endeavour was challenging, as it required permission from Moscow.

Our focus was primarily on teaching programming and fostering algorithmic thinking. Despite the scarcity of computers in schools, which were limited in functionality, we emphasized teaching algorithms for problem-solving. Our approach prioritized critical, structural, and constructive thinking skills. We developed both theoretical methodologies and practical approaches for programming education in schools, creating hundreds of engaging and stimulating programming tasks.

Targeted and in-depth teaching of programming using Pascal in schools has helped students perform well in informatics olympiads. The Lithuanian team returned with medals from olympiads in informatics during the last three decades.

The first twenty years of informatics education (1986–2006) are presented in the book *A Road of Informatics* (Dagienė, 2006). Facts, events, dates, and numbers are illustrated with pages of pictures: images of publications, compact discs, and facsimiles of documents – everything that was the most important for that period – are shown.

The journey of informatics education in Lithuanian schools spans decades, evolving from its inception in the 1970s to the present day, with Pascal playing a pivotal role. To identify the main factors for the emergence and establishment of programming education

Fig. 12. Niklaus Wirth and Valentina Dagienė in Zurich in 2007

in schools and the impact of the features of Pascal in teaching programming, a qualitative study has been conducted, delving into the memories and insights of experts who shaped the early development of programming education. Semi-structured interviews with open-ended questions were employed to gather data, which underwent analysis using an inductive thematic approach. Furthermore, descriptive statistics methods and correlation analysis were applied to scrutinize the qualitative data. Carefully selected experts, comprising pioneering teachers, instructors from the JPM, and younger-generation educators with Pascal expertise, allowed us to collect and analyse valuable insights into the early stages of programming education in Lithuanian schools. The relationships between technological advancements and society readiness, early national innovative practices, Pascal's attractive features for learning programming, and the development of programming methodologies have been identified.

By the way, this article has been written by three generations of scientists: the pioneer of programming education in Lithuanian schools, 88-year-old Gintautas Grigas, his former PhD student, now Professor Valentina Dagienė (together with Niklaus Wirth in Figure 12), and her former PhD student, now Associate Professor Tatjana Jevsikova.

Teaching programming in secondary and even primary school is a multifaceted endeavour that raises intriguing questions and demands thoughtful consideration (Armoni, 2016; Duncan, 2019; Morris, 2017; Ross *et al.*, 2023; Sun *et al.*, 2022; Webb *et al.*, 2017). One pressing issue is the integration of programming into heavily-packed school curricula. How can educators effectively weave informatics including programming lessons into the educational framework (Gander *et al.*, 2013)? Striking the right balance between theory and practice is crucial. While understanding programming concepts is vital, students also need opportunities to apply their knowledge practically. How can educators design programming lessons that effectively integrate theoretical concepts with real-world applications (Dagienė *et al.*, 2021)?

Equipping teachers with the necessary skills and confidence to teach programming effectively is another challenge (Sentance and Csizmadia, 2017; Yadav *et al.*, 2022). Many educators may lack experience in programming instruction, particularly focusing on the

understanding of informatics concepts, necessitating robust professional development and ongoing support systems. Moreover, there is the imperative to ensure equitable access to programming education. Disparities in technology and resources across schools could widen existing educational inequalities. How can policymakers ensure that all students, regardless of their background or school resources, have equal access to programming instruction?

Lastly, assessment methods must evolve to accurately measure students' programming proficiency and problem-solving skills (Hu, 2024). Traditional assessment strategies may fall short in capturing the nuanced abilities cultivated through programming education. How can educators develop innovative assessment methods that align with the dynamic nature of programming education?

Tackling these challenging questions will require collaboration among educators, policymakers, industry experts, and other stakeholders to ensure that programming education in schools is inclusive, effective, and reflective of the evolving landscape of programming languages.

Niklaus Wirth has made a great leap in bridging the gap between computer programs and human understanding. His innovative approach brought about a paradigm shift in teaching programming, with a strong emphasis on clarity and readability program texts. Wirth developed a clear and logical framework of programming language concepts, clarified the relationships between data and algorithm constructs, and promoted comprehensibility, readability, accessibility, and user-friendliness of programs to programmers of all levels, including novices.

*Ethics Declaration*

The study has been conducted following ethical guidelines outlined by Vilnius University. Participating experts were informed in advance about the aims of the study, the purposes of the data and for whom it would be used, and written consent was collected. All data was anonymized to ensure confidentiality. Participants were assured of their right to withdraw. All the collected data was safely stored in encrypted facilities and was only used for the intended research purposes.

*Declaration of Interest Statement*

The authors report there are no competing interests to declare.

5.1. *Data Availability Statement*

The anonymized data can be obtained on a reasonable request e-mail to the corresponding author.

## A. Appendix

Set of Pascal-based books (textbooks, exercise books, and analyses of tasks of programming olympiads and contests) published in Lithuania for teaching programming and algorithms in schools in 1980–2002.


Grigas, G. (1979). Translator jezyka Pascal v operacionoj sisteme DOS/ES i ego ispolzovanije dlia učebnych celej. Novosibirsk: VC.

Baliūnaitė, A., Dagienė, V., Grigas, G. (1979). Transliator jazyka PASCAL v operacionoj sisteme DOS/EC i jevo ispolzovanyje dlia učebnych celej. Operativno-informacionyj material. Novosibirsk: SO AN SSSR.

Dagienė, V., Grigas, G., Petrauskienė, A. (1980). Programavimo kalba PASCAL. Vilnius: LTSR MA MKI.

Grigas, G. (1981). Funkcijos ir procedūros. Vilnius: MKI.

Grigas, G. (1982). Programavimo pradmenys. Vilnius: Mokslas.

Dagienė, V., Grigas, G. (1983) Programavimo pradmenų uždavinynas. Vilnius: LTSR MA MKI.

Augutis, K., Dagienė, V., Grigas, G. (1983). Duomenų tipų uždavinynas. Vilnius: LTSR MA MKI.

Dagienė, V., Grigas, G., Petrauskienė, A. (1983). Paskalio programavimo kalba. Vilnius: Mokslas.

Grigas, G. (1984). Duomenų tipai. Vilnius: MKI.

Dagienė, V., Grigas, G., Augutis, K. (1986). Šimtas programavimo uždavinių. Kaunas: Šviesa.

Dagienė, V. (1986). Programavimo pamokos ir uždaviniai. Vilnius: LTSR MA MKI.

Grigas, G. (1986). Duomenų tipai ir struktūros. Vilnius: Mokslas.

Grigas, G. (1987). Načala progrommirovanije. Moskva: Prosveščenije.

Grigas, G. (1987). Programavimo pagrindai. Kaunas: Šviesa.

Dagienė, V., Grigas, G. (1987, 1988). Programavimo uždaviniai. Vilnius: LTSR MA MKI.

Dagienė, V., Dagys, V. (1988). Algoritmų sudarymas I. Vilnius: RMTI.

Dagienė, V., Dagys, V. (1988). Algoritmų sudarymas II. Vilnius: RMTI.

Dagienė, V. (1989). Mokomės programuoti. Kaunas: Šviesa.

Tumasonis, V., Dagienė, V., Grigas, G. (1990). Paskalis. Programuotojo vadovas. Vilnius: Mokslas.

Dagienė, V. (1991). Lietuvos jaunųjų programuotojų olimpiados. Kaunas: Šviesa.

Dagienė, V., Grigas, G. (1991). Informatika. XI–XII klasei. Kaunas: Šviesa.

Dagienė, V., Grigas, G. (1992). Informatikos mokymas. Mokytojo knyga. Kaunas: Šviesa.

Dagienė, V., Grigas, G. (1992). Programavimo uždavinynas. Kaunas: Šviesa.

Tumasonis, V., Dagienė, V., Grigas, G. (1992). Pascal. Rukovodstvo dlia programista. Moskva: Radio i sviaz.

Dagienė, V., Grigas, G., Augutis, K. (1992). 100 zadač programirovanija. Moskva: Prosveščenije.

Dagienė, V., Grigas, G. (1993). Informatyka: Probna pomoc naukowa. Dla klas 10–12. Kaunas: Šviesa.

Dagienė, V., Grigas, G. (1994). Lietuvos jaunųjų programuotojų konkursai. Kaunas: Šviesa.

Dagienė, V., Klupšaitė, A. (1996). Duomenų tipų ir kompiuterinės grafikos uždavinynas. Kaunas: Šviesa.

Grigas, G. (1997). Duomenų tipai. Vilnius: Žuvėdra.

Dagienė, V. (1998). Informatika: Informacija, I dalis / Vadovėlis bendrojo lavinimo mokykloms 9–10 kl. Vilnius: TEV.

Dagienė, V. (1998). Informatika: Algoritmai, II dalis / Vadovėlis bendrojo lavinimo mokykloms 9–10 kl. Vilnius: TEV.

Dagienė, V., Skūpienė, J. (1999). Lietuvos moksleivių olimpiadų uždaviniai. I dalis. Vilnius: TEV.

Dagienė, V. (1999). Informatika: Kompiuteris, III dalis / Vadovėlis bendrojo lavinimo mokykloms 9–10 kl. Vilnius: TEV.

Dagienė, V., Grigas, G. (2000). Programavimo pradmenų uždavinynas / Realinio profilio vidurinėms mokykloms. Vilnius: TEV.

Dagienė, V., Skūpienė, J. (2001). Lietuvos moksleivių olimpiadų uždaviniai. II dalis. Vilnius: TEV.

Dagienė, V., Blonskis, J. (2001). Programavimo pradmenys / Vadovėlis XI–XII klasėms, Vilnius: TEV.

Dagienė, V. (2001). Informatikos pradmenys. II dalis. Algoritmai / Pataisytas ir papildytas leidimas, Vilnius: TEV.

Dagienė, V. (2001). Informatikos pradmenys. III dalis. Kompiuteris / Pataisytas ir papildytas leidimas, Vilnius: TEV.

Dagienė, V., Blonskis, J. (2001). Programavimo pradmenys / Vadovėlis XI–XII kl. Vilnius: TEV.

Dagienė, V. (2002). Informatika: Trumpas informatikos kursas. Vilnius: Gimtinė.

## References


Armoni, M. (2016). Computing in Schools. Computer Science, Computational Thinking, Programming, Coding: the Anomalies of Transitivity in K-12 Computer Science Education. *ACM Inroads*, 7(4), 24–27.

Braun, V., Clarke, V. (2006). Using Thematic Analysis in Psychology. *Qualitative Research in Psychology*, 3(2), 77–101.

Byrne, D. (2022). A Worked Example of Braun and Clarke's Approach to Reflexive Thematic Analysis. *Quality and Quantity*, 56, 1391–1412.

Cheah, C.S. (2020). Factors Contributing to the Difficulties in Teaching and Learning of Computer Programming: a Literature Review. *Contemporary Educational Technology*, *12*(2).

Clarke, V., Braun, V. (2017). Thematic Analysis. *The Journal of Positive Psychology*, 12(3), 297–298.

Dagienė, V. (1989). *Mokomės Programuoti*. Šviesa, Kaunas.

Dagienė, V. (1999). Programming-Based Solution of Problems in Informatics Curricula. In: *Communications and Networking in Education: Learning in a Networked Society*, Aulanko, Hämeenlinna, Finland, pp. 88–94.

Dagienė, V. (2006). *The Road of Informatics*. TEV, Vilnius.

Dagienė, V., Futschek, G. (2008). Bebras International Contest on Informatics and Computer Literacy: Criteria for Good Tasks. In: *Proceedings of the 3rd International Conference on Informatics in Secondary Schools (ISSEP 2008)*. Lecture Notes in Computer Science*: Vol.* 5090. Springer, pp. 19–30.

Dagienė, V., Grigas, G. (1992). *Programavimo uždavinynas*. Šviesa, Kaunas.

Dagienė, V., Sentance, S. (2016). It's Computational Thinking! Bebras Tasks in the Curriculum. In: *Proceedings of the 9th International Conference on Informatics in Schools (ISSEP 2016)*. Lecture Notes in Computer Science*: Vol.* 9973. Springer, Cham, pp. 28–39.

Dagienė, V., Stupurienė, G. (2016). Bebras – A Sustainable Community Building Model for the Concept Based Learning of Informatics and Computational Thinking. *Informatics in Education*, 15(1), 25–44.

Dagienė, V., Grigas, G., Petrauskienė, A. (1983). Transliator jazyka Pascal, prednaznačenyj dlia učebnych celej. *Programmirovanie*, (2), 87–89.

Dagienė, V., Hromkovič, J., Lacher, R. (2021). Designing Informatics Curriculum for K–12 Education: from Concepts to Implementations. *Informatics in Education*, 20(3), 333–360.

Dagys, V. (1994). The Work Principles of Lithuanian Young Programmers School by Correspondence. In: *Human Resources, Human Potential, Human Development: the Role of Distance Education*, Tallinn, pp. 182–184.

Dagys, V., Klupšaitė, A. (1993). Distance Teaching of Programming and Possibilities of E-Mail. *Informatica*, 4(3–4), 303–311.

Dagys, V., Dagienė, V., Grigas, G. (2006). Teaching Algorithms and Programming by Distance: Quarter Century's Activity in Lithuania. In: *Local Proceedings of the 2nd Conference on Information Technologies at School (ISSEP 2006)*, pp. 402–412.

Dinda, A. (1990). Paskalis Kompiuteriui BK-0010. *Informatika*, 15, 47–54.

Duncan, C. (2019). *Computer Science and Computational Thinking in Primary Schools*. Routledge.

Gander, W., Petit, J., Berry, G., Demo, G., Vahrenhold, J., McGettrick, A., Meyer, B. (2013). Informatics Education: Europe Cannot Afford to Miss the Boat. Technical report, Informatics Europe & ACM Europe Working Group on Informatics Education.

Grigas, G. (1989). Informatics and Creative Thinking. In: *Proceedings of the International Conference: Children in the Information Age*, Sofia, pp. 229–240.

Grigas, G. (1990). Some Aspects of Teaching the Art of Programming by Correspondence. *Informatica*, 1(1), 156–166.

Grigas, G. (1995). Investigation of the Relationship Between Program Correctness and Programming Style. *Informatica*, 6(3), 265–276.

Grigas, G., Jevsikova, T. (2001). Komponentinio programavimo taikymas informatikos mokyme. *Lietuvos matematikos rinkinys*, 41, 278–283. https://doi.org/10.15388/LMR.2001.34505.

Holt, R.C., Wortman, D.B., Barnard, D.T., Cordy, J.R. (1977). SP/*k*: a System for Teaching Computer Programming. *Communications of the ACM*, 20(5), 301–309. https://doi.org/10.1145/359581.359586.

Hu, L. (2024). Programming and 21st Century Skill Development in K–12 Schools: a Multidimensional Meta-Analysis. *Journal of Computer Assisted Learning*, 40(2), 610–636.

Institute of Mathematics and Informatics (2004). Lokalizuota "Free Pascal" programavimo sistema. Technical report.

Jevsikova, T. (2007). Understanding Cultural Aspects of ICT for Schools: Naming Issues. In: *Informatics, Mathematics, and ICT: a 'Golden Triangle'. Working Joint IFIP Conference*. Northeastern University, Boston, Massachusetts, USA, pp. 1–5.

McGill, T.J., Volet, S.E. (1997). A Conceptual Framework for Analyzing Students' Knowledge of Programming. *Journal of Research on Computing in Education*, 29(3), 276–297.

Morris, D. (2017). *Teaching Computational Thinking and Coding in Primary Schools*. SAGE Publications Ltd, London.

Nielson, F., Nielson, H.R., Hankin, C. (1999). *Principles of Program Analysis*. Springer.

Oberon Microsystems (2001). Component Pascal Language Report. Technical report, Oberon Microsystems Inc., Switzerland. https://blackbox.oberon.org/cp-lang.pdf.

Robins, A., Rountree, J., Rountree, N. (2003). Learning and Teaching Programming: a Review and Discussion. *Computer Science Education*, 13(2), 137–172.

Ross, M.S., Ronan, D., Erdil, D.C., Brylow, D., El-Hamamsy, L., Bruno, B., Parker, M.C. (2023). Computing Education. *ACM Transactions on Computing Education*, *23*(2).

Sentance, S., Csizmadia, A. (2017). Computing in the Curriculum: Challenges and Strategies from a Teacher's Perspective. *Education and Information Technologies*, 22, 469–495.

Sun, L., Guo, Z., Zhou, D. (2022). Developing K-12 Students' Programming Ability: a Systematic Literature Review. *Education and Information Technologies*, 27(5), 7059–7097.

Webb, M., Davis, N., Bell, T., Katz, Y.J., Reynolds, N., Chambers, D.P., Sysło, M.M. (2017). Computer Science in K–12 School Curricula of the 21st Century: Why, What and When? *Education and Information Technologies*, 22, 445–468.

Yadav, A., Connolly, C., Berges, M., Chytas, C., Franklin, D., Hijón-Neira, R., Warner, J.R. (2022). A Review of International Models of Computer Science Teacher Education. In: *Proceedings of the 2022 Working Group Reports on Innovation and Technology in Computer Science Education (ITiCSE-WGR 2022)*. Association for Computing Machinery, pp. 65–93.