

Assessment of Computational Thinking – A Croatian Evidence-Centered Design Model

Nikolina BUBICA¹, Ivica BOLJAT²

¹*Moko ica Middle School, Dubrovnik, Croatia*

²*Faculty of Science, University of Split, Split, Croatia*
e-mail: nikolina.bubica@skole.hr, boljat@pmfst.hr

Received: February 2021

Abstract. The new Croatian Informatics curriculum, which introduces computational thinking concepts into learning outcomes has been put into practice. A computational thinking assessment model reflecting the learning outcomes of the Croatian curriculum was created using an evidence-centered design approach. The possibility of assessing the computational thinking concepts, abstraction, decomposition, and algorithmic thinking, in an actual classroom situation and examples of such assessment is increasingly coming to the forefront of computer science educational research. Precisely for that purpose, the research was conducted. Research data are collected through the test and questionnaire of 407 pupils (10 middle schools, age 12), analysed by exploratory factor analysis and non-parametric tests. Results showed that the presented model was suitable to assess the understanding of the concepts of abstraction and algorithmic thinking, independently of the previous experience with programming languages and pupil's gender, while assessment of decomposition needs more work and improvement, some recommendations are provided. Also, it received positive feedback from pupils and teachers what implicated that such an assessment model could help teachers in building a real-time measurement instrument.

Keywords: educational testing, computational thinking, curriculum-based assessment, item analysis, programming, factor analysis.

1. Introduction

From the autumn of 2018, we have been witnessing the application of a new informatics/computer science (CS) curriculum which is part of a major educational reform in the Republic of Croatia. The reform represents a change in the teaching and learning process and introduces new concepts, such as computational thinking (CT), into K-12 computer science education (CSE). The evaluation of CT then becomes a challenge in CSE, demanding a systematic approach. Technology is ubiquitous in our lives, and regardless of pupils' future occupations, their age, or the type of technology they use, they are increasingly expected to possess some generic CT competencies, such as the

ability to solve problems in everyday life, disaggregating complex problems into simpler ones and generalizing solutions; increasingly, these generic competencies will in practice be technology mediated. Appropriate pedagogical practices which emphasize the constructivist approach to learning and put pupils at the heart of the learning process should develop competencies such as independence, self-confidence, responsibility, and entrepreneurship (Ben-Ari, 1998; Moon, Do, Lee, and Choi, 2020). From this perspective, learning experiences should be based on the belief that pupils are best taught by helping them participate actively, that they are ready to make great efforts and apply their creativity, and that teamwork and collaboration are a powerful motivation for learning (Leron and Hazzan, 1998; Ambrosio and Almeida, 2014; Dagienė and Futschek, 2019).

A fundamental question today is how to respond to such challenges. Leaders in computer science education increasingly emphasize the need to modify existing computer science curricula to include the development of specific CT concepts such as abstraction and decomposition (Csizmadia, *et al.*, 2015; Yadav, Aman; Stephenson, Chris; Hong, Hai, April 2017).

In what follows, an ECD (Evidence-centered design) model for creating quality tasks for assessing CT concepts aligned with the new Croatian informatics curriculum will be presented. During the spring of 2018, the research was conducted in ten Croatian middle schools to examine the appropriateness of using online tool, consisted of tasks created according to a given model, in a real class situation. The standard statistical characteristics of the applied tool as well as its ability to really measure the adoption of CT concepts (abstraction, algorithmic thinking, decomposition) were examined. As there were different programming languages used in the teaching process, it was important to explore how well the applied tool managed to avoid the influence of the specific programming language used during the teaching process and put emphasis on the CT concepts instead on the syntax of the programming language.

2. Theoretical Background

2.1. About CT

In research literature, computational thinking, along with mathematics, engineering, and reading literacy, is often seen as one of the standard types of literacy for pupils in general today (Wing, 2006); it can be understood as “the process of formulating problems and their solutions, but in ways that solutions are presented in a form that enables them to perform effectively with some information processing agent” (Wing, 2010, p. 1). The idea of computational thinking reaches back to Papert’s work on Logo programming and children’s usage of computers, in which he emphasized the value of programming for the development of procedural thinking (Papert, 1980). Papert believed that by using technology and programming to create “micro-worlds”, pupils would develop skills that they could then transfer to a non-programming context outside of the teaching environment. A little later, and to this day, the constructivist approach to

teaching became prominent; under this approach, the pupils are expected to create their own sustainable constructs of knowledge (Ben-Ari, 1998), and in programming specifically, this construction is linked to the ability to predict and understand what is going on when some computer program is executed.

There is still considerable confusion over definition of computational thinking, and a whole series of questions and challenges that need to be addressed. For some, the term refers to a universal competence of every child, which together with analytical skills serves as the basis for each child's school learning (Wing, 2006). Some point out that computational thinking concepts are used in disciplines other than computer science, for example in problem-solving tasks (Bundy, 2007). Denning (2009) discusses whether computational thinking is exclusive to the field of computer science and claims that "CT is one of the key practices of computer science ... not unique to computing and is not adequate to portray the whole of the field" emphasizing that although computational thinking should be considered an important part of computer science, computer science is a much broader term. Also, computational thinking has its place in some other areas, outside of computer science. Guzdial (2008) considers computational thinking to be a 21st-century form of literacy that is necessary for a whole series of academic fields; using computational thinking concepts like abstraction, for instance, pupils can learn how to lower complexity by reducing some details of the observed phenomena or problem and still preserve the basic meaning (Yadav, Stephenson, and Hong, 2017).

In discussions of computational thinking, it is important to define and describe its connection to algorithmic thinking; Denning points out that "computational thinking means interpreting the problem as an information process for which we are then trying to find an algorithmic solution" (Denning, 2010). To create an operational definition of computational thinking, the ISTE (International Society for Technology in Education) and CSTA (Computer Science Teachers Association) analysed feedback from about 700 surveyed computer science teachers and researchers. The result was formulated in an operational definition of computational thinking for K-12 education as a "problem-solving process that includes:

- Formulating problems in a way that enables us to use a computer and other tools to help solve them.
- Logically organizing and analysing data.
- Representing data through abstractions such as models and simulations.
- Automating solutions through algorithmic thinking (a series of ordered steps).
- Identifying, analysing, and implementing possible solutions with the goal of achieving the most efficient and effective combination of steps and resources.
- Generalizing and transferring the problem-solving process to a variety of problems" (ISTE and CSTA, 2011).

When talking about teaching and learning of computational thinking, perhaps the most interesting aspect is the role of programming. Programming knowledge includes the ability to read and write in a specific programming language and to think computationally (Roman-Gonzales, 2014). Computational thinking and programming are not the same concepts, but they are strongly related. Programming can help foster computational

thinking (Lye and Koh, 2014), but computational thinking can also be applied to different types of problems that do not directly involve programming tasks (Wing, 2008).

It is questionable how much actual programming, if any, is needed to acquire computational thinking. There is no universally accepted answer yet, but practice indicates different ways of programming involvement across contexts in learning and teaching of computational thinking. Some define computational thinking as a fundamental, ubiquitous problem-solving tool and suggest several activities and projects to address it from this perspective (Astrachan, Hambruch, and Peckham, 2009). Most approaches suggest various ways of incorporating programming into teaching and learning of computational thinking, from those in which programming is the fundamental computational thinking skill to those that integrate computational thinking in and through various general education courses.

2.2. *CT Evaluation*

Since there is still no formal consensus definition of CT, it can be imagined that its evaluation remains even more in question. Mostly, to evaluate CT, it is necessary to find evidence of a deeper understanding of a CT-relevant problem solved by a pupil, that is, to find evidence of understanding how the pupil created their coded solution. For example, if we consider abstraction, we must find ways to evaluate how pupils apply it in their solutions while trying to solve a problem (ISTE and CSTA, 2011). Despite a lack of consensus thus far, a few relatively widespread approaches for evaluating the development of CT have emerged. They can serve as a foundation for a general approach. One such CT evaluation method combines different methods of collecting feedback: analysing project portfolios, analysing interviews, and developing project design scenarios, in paper or digital form (Brennan and Resnick, 2012); it assesses the fluency of computer-based practices of testing and debugging, experimenting and repetition, abstraction and modulation, and reusing and remixing/scaling at three levels: low, medium and high. Dorling and Walker studied the practice of teaching CT in the classroom and proposed a framework for evaluating the “computing progression pathway”, which recognizes the major areas of CS and outlines specific levels of adoption (Dorling and Walker, 2014). Dr. Scratch presents an example of automatic online CT evaluation of Scratch projects, that is, creations made in the Scratch graphical programming environment, which can involve music, animation, art, games, and simulations (Moreno-León, Robles, and Román-González, 2015). At this point, this online tool is working with Scratch 1.4, 2.0 i 3.0 files. The system assigns a CT score automatically and detects bad programming habits and errors to help learners develop their CT skills (Dr Scratch, 2021).

Another interesting approach to CT evaluation applicable in formative and summative evaluation is the Bebras challenge. It is an international initiative promoting computational thinking in pupils of all ages, performed at school using computers or mobile devices, integrated into regular classes, or in the form of a special competition. Bebras tasks are puzzling stories that incorporate a puzzle or task relying on some key CT concept, designed to promote computational thinking in pupils of all ages. There is

certainly evidence that Bebras tasks are an effective tool in promoting problem-solving and computer thinking concepts (Dagienė and Stupurienė, 2016; Araujo, Santos and Andrade, 2017; Curran, 2019). However, evaluating computational thinking through Bebras tasks is demanding (Dagienė and Stupurienė, 2016), and it is debatable whether CT concepts can be effectively evaluated using Bebras challenges tasks (Araujo *et al.*, 2017), especially since various aspects of developing good Bebras tasks still lack adequate research.

Today, it is possible to find several other published computer-based or paper-pencil CT tests usable in different contexts; however, effectiveness of those approaches is still being investigated (Werner, Denner, and Campe, 2012; Román-González, 2015). An interesting approach to CT assessment combines tasks from specific CT assessment (CTt) and Bebras challenge assessments (Wiebe, *et al.*, 2019). By combining the two assessments, a subset of tasks was selected to create a promising assessment that could be applicable in a reasonable time, but also be suitable for pupils with no previous programming experience. The results showed that the use of this instrument in pre- and post-testing can reveal the effectiveness of such assessment. Also, the instrument can collect quality information on the development of pupils' computational thinking during the teaching process. The authors emphasize the need for further research on the issue of validity and further work on the selection of an appropriate subset of tasks (Wiebe, *et al.*, 2019).

The research presented in this paper was strongly influenced by the ECD approach to assessment. Evidence-centered assessment design is an approach to constructing educational assessments in terms of evidentiary arguments (Almond, Steinberg, and Mislevy, 2002; Mislevy, Almond, and Lukas, 2003; Mislevy and Haertel, 2006; Bubica and Boljat, 2018; Grover, 2020). It focuses on the evidence of competencies as a foundation for the construction of excellent assessment tasks. ECD could be described with layers: Domain Analysis, Modelling, Conceptual Assessment Framework, Assessment Implementation and Delivery, which incorporate actions like analysing assessment domain, selection and administration of the tasks, interaction with pupils to present or capture work products, evaluation of the responses from the task and accumulation of the evidence across them. This approach highlights knowledge and skills complexity and other features and behaviours that should be valued, such as basic ICT knowledge, possible working products (written/digital product or a spoken answer in which pupils might produce evidence) and observations, and variable features connected to assessment (Mislevy and Riconscente, 2005). The creation of assessment that can better reflect and measure what is happening in the classroom and getting results of assessments that are heavily supported by evidentiary arguments are highlighted as possible advantages of this approach (Hendrickson, Ewing, Kaliski, and Huff, 2013).

The work conducted within the PACT (Principled Assessment of Computational Thinking) project, also incorporated evidence-centered design (ECD) to represent general CS practice through design patterns, especially to create a structured description of the domain (in which learning is occurring), evidence (for use in a task), and argument (that emerges from the evidence). Such patterns should emphasize application and review of design skills while solving computational problems, rather than evaluating

the knowledge of the concepts necessary to apply such skills. The SRI International Education group through the PACT project proposed application modes for every layer to create a comprehensive practice of CT assessment (Bienkowski, Snow, and Rutstein 2015). They presented design patterns for major computational thinking practices and developed templates for assessment task development for computational thinking practices in the context of ECS (Exploring Computer Science).

The principled assessment design methodology as well as ECD was also applied in the “VELA assessment” (Grover, 2020), where an instrument was developed to measure the adoption of introductory programming concepts (grades 6 to 8). Assessment design patterns as well as the measurement instrument have been created to measure the understanding of concepts such as variables, expressions, loops, conditions, and abstractions. These concepts represent the basic concepts of most introductory programming courses, regardless of the applied programming language or environment. Besides these concepts, the VELA assessment also included tasks with Scratch script which, in addition to the concepts mentioned above, were used to assess the understanding of concepts such as events. Supporting rubrics and design patterns have been developed that can also serve other teachers as good templates for developing their own assessments and ensure broad use of the instrument. Creating quality assessments is extremely important for pupils and teachers. Well planned assessment and quality teachers’ feedback can improve the acquisition of competencies and enables pupils to gauge their own progress (Pellegrino, 2020).

Recognizing the importance of computational thinking and its introduction in K-12 education, CS teachers are faced with the challenge of incorporating these concepts into their teaching in primary and secondary school (Csizmadia *et al.*, 2015). There is an increasing awareness of the fact that teacher-training programs should include discussions of computational thinking and provide recommendations and examples of best practices for incorporating CT concepts into the teaching process (Yadav, Stephenson, and Hong, 2017). There is a need for better cooperation between teacher educators and computer science teachers to enhance the education of CS teachers in this regard (Yadav, Gretter, and Good, 2017).

2.3. Research Questions

In this work, the results of the conducted research regarding created model of CT assessment will be presented and discussed. Specific research questions regarding CT concepts (algorithmic thinking, abstraction, and decomposition) were:

- Is the proposed CT assessment independent from specific programming language and approach?
- Is the proposed CT assessment suitable for measuring computational thinking concepts, abstraction, algorithmic thinking, and decomposition?
- Is the proposed CT assessment tool better aligned with the Croatian informatics curriculum than the selected tasks of the Bebras challenge and is it thus more suitable for use in educational practice in the Republic of Croatia?

3. Methods

3.1. Informatics Education in Croatia

The teaching of Informatics in primary education in the Republic of Croatia has been present since 1990, in the form of an elective subject for pupils from fifth to eighth grade. In secondary schools, informatics was introduced ten years earlier in the form of a regular subject through one, two, three or four years of schooling, depending on the type of secondary school. In some secondary schools, the subject is called Computer Science.

It must be emphasized that with the development of new technologies and thus an increasing number of applications that support the use of technology in everyday life, informatics/computer science teaching has increasingly become a service for learning about information and communication technology, while basic, often demanding content such as programming lose the battle with easily accessible, less demanding, and often more fun content such as text processing, editing pictures and videos, making presentations, etc. Those Informatics/CS curriculums were defined by specific content that should be included in teaching, which, due to rapid IT progress, makes it very fast outdated and inadequate background for teachers to organize and plan their work. Even so, all earlier versions of the Informatics/CS curriculum included, in various forms, the content of programming.

In March 2018, the Croatian Ministry of Education published a new CS curriculum for K-12 education (Ministry of Education, 2018). The curriculum was a hopeful prospect for CS teachers, since most of them felt restrained by the old, outdated curriculum. It was created according to learning outcomes instead of prescribed content, enabling the realization of learning and teaching directed at each individual pupil and the development of their potential. For example, instead of teaching pupils how to shape specific document features in Microsoft Word, such as header, number of pages, styles, etc., a learning outcome associated with a realistic task is released, such as creating digital content on a given topic, without defining the application to be used. Thus, the focus shifts from content to learning outcomes that are realized with specially designed activities. This approach provides flexibility and freedom to teachers in designing the learning and teaching process.

Moreover, this Informatics/CS curriculum finally incorporated computational thinking as an important part of informatics education in general in the Republic of Croatia. The role of CT and programming in the curriculum includes involving pupils in logical thinking, modelling, abstracting, and problem-solving, under the principle that solid ICT (Information-communication technology) education, based on computational thinking and creativity, should prepare pupils for understanding the changes in the world around us (Brodanac *et al.*, 2016).

As mentioned earlier, although programming was included in all earlier versions of informatics curriculum, the situation in on-site schooling showed diversity in content being thought, programming languages being used and even in the duration of teaching programming. In addition to these differences, the common thing was that while

teaching programming content, great emphasis was on the application of appropriate programming language commands, and less on problem-solving skills, data handling, modelling, abstraction, and similar difficult-to-measure concepts such as computational thinking concepts.

In the new Croatian curriculum, learning outcomes were developed according to several documents, but mostly the Croatian National Educational Standards (Ministry of Education, 2006), CSTA Computer Science Standards (CSTA, 2016), and CS curriculum (Ministry of Education, 2018). The National Educational Standards define the way in which computer science is involved in Croatian primary, secondary, and higher education. The CS Curriculum and CSTA Computer Science Standards define CS learning outcomes for each degree of adoption or acquisition of the skills at each educational level. Learning outcomes are expressed in detail within the Bloom taxonomy (Anderson, Krathwohl, and Bloom, 2001) at different adoption levels: satisfactory, good, very good, and exceptional (Ministry of Education, 2018; Brođanac, *et al.*, 2016).

The curriculum is still briefly and insufficiently in use to assess its quality and evaluate the success of its application. Nevertheless, a survey conducted among informatics/computer science teachers at the very beginning of the new curriculum application revealed some interesting information and indicated areas that need to be further researched and supported (Table 1).

Among 817 teachers (72% primary teachers, 28% secondary teachers) who participated in the survey, the vast majority confirmed that they teach programming in their work although most of them consider it to be very demanding content for pupils. Most teachers confirmed that they were already familiar with the concept of computational thinking and that they use it in their work.

Still, the analysis of terms which they mostly connected with the term of computational thinking (Fig. 1) showed that in their work there was higher emphasis on developing programming skills than adopting computational thinking concepts like abstraction or decomposition.

Furthermore, in the discussions on the definition, concepts, teaching approaches and evaluation of computational thinking, evaluation stood out as a topic that teachers know the least and needed better support (43%) (Ministry of Education, 2018). These results point to the fact that there is still some dilemma and some misunderstanding among teachers about the very concept of computational thinking and that in the years to come it will be better shown how successfully the new curriculum has responded to the challenges that informatics teachers face daily.

Table 1
Teachers' opinions: What is CT? (Ministry of Education, 2018)

Teachers' basic opinions about CT	N = 817
I teach programming as a regular part of curriculum.	91%
I think programming is demanding content for pupils.	82%
I am familiar with the meaning of computational thinking.	93%
I already involved computational thinking in my teaching process.	91%

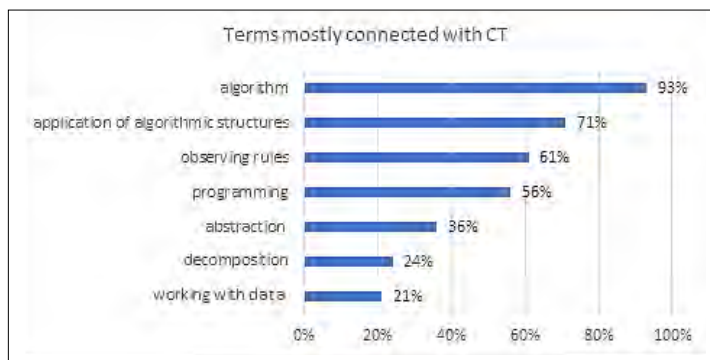


Fig. 1. Terms mostly connected with CT (Ministry of Education, 2018).

The creation of new informatics/computer science curriculum was part of a comprehensive educational reform which, in addition to the development of new subjects' curricula, also developed cross-curricular topics that connect topics of universal human values and competencies for life in the 21st century. In this way, these topics are daily present in a special way in the educational work of the entire educational vertical. One of the cross-curricula topics was the Use of Information and Communication technology which included efficient, appropriate, timely, responsible, and creative use of information and communication technology in all subjects, areas and at all levels of education. Quality application of the ICT cross-curricula topic in the years to come could free informatics education from the main role in the development of digital literacy and enable greater concentration on topics such as computational thinking.

3.2. Model of CT Assessment – Croatian ECD Model

The proposal for a computational thinking assessment approach presented in the next paragraphs uses ECD as an orientation towards multiple activities necessary to create useful documentation: domain analysis, domain modelling, construction of a framework (pupil model, task model, evidence, and measurement model of evidence), and assessment implementation and delivery. Unlike the SRI PACT strategy, which assess computational thinking using an interdisciplinary STEM approach, this model is aligned with the Croatian Informatics⁷/CS curriculum and proposes a method of assessment that is applicable in the real classroom environment. Furthermore, while some assessment approaches rely on the programming language or environment used during teaching process, e.g. Scratch programming environment (Grover, 2020), this model advocates independence of the programming language and offers the development of tasks that are equally appropriate / understandable to pupils who have met different programming languages e.g. Scratch and Python. Such an approach is desirable because the new Informatics⁷/CS Curriculum doesn't define programming language or environment to be used in teaching process, so in real educational practice teachers use different languages like Scratch, Python, Logo, C, Kodu (Microsoft Research, 2009).

3.2.1. Domain Analysis and Modelling

The basic goal of the domain analysis under the ECD approach is to find and explore all relevant materials concerning target learning outcomes as defined in the new Croatian CS curriculum. These learning outcomes were the basis for the assessment process of computational thinking and programming domain learning in pupils aged 11–12 (5th–6th grade) which will be looked at in this research (see Table 2).

In the ECD approach, domain modelling is the process meant to identify elements that describe the domain under assessment and is represented through five categories: fundamental and additional knowledge, skills and features, possible working products, variable features, and possible observations. Is it necessary to conduct computational thinking assessment using some programming tool or environment, or can it be done generically and abstractly? The question of the connection between computational thinking and programming must be defined in relation to the specific context of the assessment. There are different approaches to incorporating programming into the processes of teaching and resulting computational thinking assessment. We differentiate these approaches according to the role that programming and computational thinking fulfil in the course curriculum (Astrachan *et al.*, 2009). In this research, computational thinking assessment was achieved through an approach independent of a particular programming tool or environment, which served for assessing the adopted learning outcomes in real classroom situations in middle school education. Independence of assessment tool of

Table 2
CT and programming learning outcomes for pupils aged 11–12 (5th–6th grade)
(Ministry of Education, 2018)

	Satisfactory level	Good level	Very good level	Excellent level
Computational thinking and programming domain (B) Pupils' age 11 (5 th grade)	B.5.1. 1 use a programming language to design programs using input and output values and repetition			
	The pupil states how to run the software (programming) tool, recognizes the parts of the interface and blocks (commands) of the software tool that can execute an instruction. Composes a simple set of instructions using blocks / commands.	The pupil recognizes the basic segments of program development: input – processing – output. He builds a simple set of instructions that represent a solution to a problem using input and output values and assignment statement.	The pupil, with the help of the teacher, develops a solution to a problem using an iteration structure with the determined number of iterations	The pupil independently develops a solution to the problem using an iteration structure with the determined number of iterations.
	B.5.2. create an algorithm for solving simple tasks, check the validity of the algorithm, detect and correct mistakes.			
	The pupil describes the concept of an algorithm and recognizes the basic steps for solving a problem.	The pupil analyses the problem and devises and shows the steps to solve the given problem (graphically, orally or by text).	The pupil critically checks the correctness of his algorithm using default input values.	The pupil re-examines and rearranges his algorithm until the algorithm represents an exact solution to the given problem.

Continued on next page

Table 2 – continued from previous page

		Satisfactory level	Good level	Very good level	Excellent level
Computational thinking and programming domain (B)	Pupils' age 12 (6 th grade)	<p>B.6.1. design, [code-] trace, and adjust programs which contain selection and conditional repetition structures; anticipate the behaviour of simple algorithms which can be displayed in the form of a diagram, words of natural language, or a programming language</p>			
		<p>The pupil creates track and rearranges programs that contain branching and conditional iteration structures and predicts the behaviour of simple algorithms that can be represented by a diagram, spoken language or programming language</p>	<p>The pupil independently or with the help of the teacher analyses the given problem and suggests which algorithmic solution. He presents the solution of the problem in spoken language words, diagrams, or commands of the programming language, and independently plans and arranges a series of instructions as a solution to the problem by applying algorithmic structures of sequence and branching.</p>	<p>The pupil independently proposes a program/algorithm as a solution to the problem, predicts the behaviour of the algorithm and checks the correctness of the algorithm by monitoring its behaviour or by executing the program with given examples. On his own or with the help of a teacher, he puts together a series of instructions for solving a problem using conditional repetition.</p>	<p>The pupil independently creates a program / algorithm as a solution to a problem that includes a series of instructions (commands) using all algorithmic structures, provides appropriate input (test) examples and critically checks the correctness of the solution and rearranges its solution if necessary.</p>
		<p>B.6.2. explore and solve more complex problems by dividing them into smaller subproblems</p>			
		<p>The pupil describes the problem and recognizes some steps/parts in solving the problem</p>	<p>The pupil, with the help of the teacher, develops a plan for solving problems and recognizes sub-problems in it, minor problems that he has already encountered, or problems that he knows how to solve.</p>	<p>The pupil analyses the possibility of including the solution of sub-problems in the solution of a more complex problem, analyses and suggests possible changes / adjustments to the solution of sub-problems.</p>	<p>The pupil independently finds and creates a solution to a complex problem with the help of subproblems and critically evaluates and rearranges the solution if necessary.</p>

the programming languages used in education must ensure that pupils who have learned a particular programming language will not therefore achieve better results in this assessment. More precisely, the independence of the programming tool or environment enables wider application of the assessment tool and highlights computational thinking concepts rather than the syntax of a given programming tool or environmental affordances/constraints.

Although, such a tool could be used with pupils that have no programming background, it is primarily intended to fairly assess pupils' CT competencies independent of the programming language. Is it necessary to require the independence of the assessment tool of the programming language? Isn't it natural to expect that the basic programming skills should be independent of the programming language itself? Research related to working with novice programmers has shown that novices often approach programming 'line by line' instead of using meaningful programming structures. They have difficulties in various issues related to the construction of algorithms or computational solutions because, although they know the syntax and semantics of individual commands, they

do not know how to combine them into valid algorithms or computational solutions (Fincher, 1999). Due to all the above, when assessing programming, teachers usually put more emphasis on the application of commands or certain algorithmic structures in the selected programming language. When assessing computational thinking, it is crucial to make a qualitative departure from the assessment of programming skills, and to put in the foreground the assessment of problem-solving methods, data management, application of abstraction, etc.

Under the ECD approach, we had to explore and define activities such as applying algorithmic structures of sequence and iteration in a computational solution, operating with input and output values in computational solutions, etc. Further, we had to recognize what additional knowledge and features had to be considered (manipulating files/folders, searching the internet, downloading files, signing in/signing out), as well as variable features (computational solution difficulty level, presentation of computational solution, whether the pupil has come up with a solution, evaluated the solution, or compared multiple solutions, etc). As per the ECD model the proposed assessment framework can serve as a reference to assist assessment designers in designing and validating their task models. Every assessment designer should validate their work with questions regarding assessment instrument relevance, specificity, and scalability and questions related to item statistics and item complexity.

This domain analysis identified key documents and recognized computational thinking as the fundamental approach that develops ability to solve problems and ability to program:

“...The emphasis is on the process of creation of the application, from the initial idea to the final product, and not exclusively on the adoption of the syntax and semantics of the programming language. Activities and contents of outcomes from the Computational thinking and programming domain develop innovation, creativity and entrepreneurship and provide valuable knowledge that can be incorporated into future professional life.” (Ministry of Education, 2018)

Modelling of the assessment domain represents the foundations for the development of a future assessment tool and describes its main features (Table 3).

In what follows, we will present this assessment framework with information about the evidence, pupil, and task models, observable characteristics, measurement models, and test specifications.

3.2.2. Assessment Framework

Pupil model

Although programming was a part of the informatics curriculum in Croatian middle schools, there were differences in the programming languages used in the teaching process as well as in the length of the teaching of programming content.

This research was conducted in 2018 (March/April) when Informatics was an elective subject for pupils from the fifth to the eighth class of middle school. Informatics

Table 3
Modelling of the assessment domain

Basic knowledge, skills, and characteristics	<p>Understanding that a computational solution can be designed for multiple purposes.</p> <p>Ability to create solutions by combining smaller parts that lead to solving the initial (bigger, more complex) problem.</p> <p>Ability to subsequently edit the design of a computational solution.</p> <p>Ability to encode a complete solution.</p> <p>Ability to use algorithmic structures (sequence, branching and repetition) in the solution.</p>
Additional knowledge, skills, and characteristics	<p>Knowledge of a particular programming language (not required).</p> <p>Ability to login to the web pages with personal user data (login with an AAI account).</p> <p>Basic web browsing skills (opening a default web page, basic navigation within a given web site).</p>
Possible working products	<p>Computational solution.</p> <p>Comparison of multiple computational solutions or strategies.</p> <p>Description or explanation of the computational solution.</p> <p>Predicting the results of executing a computational solution.</p>
Characteristic features	<p>Presentation of a computational solution with graphical representations and an algorithmic solution written in a language like a spoken language that resembles a pseudocode in structure.</p> <p>Each evaluation task has a common context, i.e., a puzzle that develops from the initial to the final task according to the principle from the easiest to the most difficult).</p> <p>In each evaluation task, the main character of the puzzle encounters a problem that needs to be solved to successfully pass through the maze. Tasks describe, check, correct existing and create new ways of navigating the maze.</p>
Variable features	<p>Computational solution levels.</p> <p>Representation of a computational solution.</p> <p>Did the pupil come up with a solution, did he evaluate the solution, or compare computational solutions?</p>
Possible observations	<p>The degree to which a computational solution addresses a problem.</p> <p>The level of complexity of the computational solution.</p> <p>Correctness of the computational solution.</p> <p>Appropriateness of using algorithmic structures in the solution.</p> <p>The degree to which the debugging process finds and / or corrects errors.</p>

has become an obligatory subject for fifth and sixth-grade pupils with the beginning of the following school year (2018/2019) with the application of the new informatics curriculum.

The result showed that in the everyday process of learning programming pupils were engaged in programming tasks, mostly in Scratch, Python or Logo, with diversity in the length of teaching those content, from 12 to 25 hours (of 70 per school year) (Table 4).

As there were differences in concepts included in the teaching process it was important to offer an assessment model that will be appropriate for teachers regardless of which programming language, they use in the teaching process. An assessment's independence should emphasize computational thinking concepts rather than the ability to work with a specific tool or environment. For that reason, this research was conducted among pupils who already, to some extent, learned programming with different programming languages, mostly Python, Logo, and Scratch.

Table 4
Concepts included in teaching process

Concepts included in teaching process (12–25/70 hours per year)	Pupils involved
Simple branching and working with output values	80.61%
Drawing commands in turtle graphics	78.18%
Loops with a defined number of repetitions	70.91%
Algorithm	71.21%
Complex branching and working with input values	56.67%
Editing a block diagram of an algorithm	42.12%
Got acquainted with the concept of nested branching	34.85%
Concept of subprogram	28.79%
Conditional repetition loops	23.33%

Task model

In the process of developing an assessment framework, identifying possible working products (artifacts) and observations is very important. Doing so should answer some basic questions regarding assessment, such as:

- Should pupils' working product be computational solutions, comparison of multiple solutions, or even description/explanation of existing solutions?
- Should we consider problem complexity or computational solution correctness and effectiveness?
- Should we look for appropriate use of algorithmic structures in the solution?
- Should we observe the degree to which the solution addresses the problem and/or the degree to which the pupil finds and corrects the errors?

Answers to these questions depend on the pupil's age, learning outcomes included in the assessment, the assessment domain and purpose, and the real classroom situation in which the assessment is applied; in turn, these answers provide the basis for the creation of future assessment tasks. In this research, pupil assessment tasks were created for novice programmers, that is, pupils who studied programming for a total of 12 to 40 hours over two years. During this period, they got acquainted with the basic (low-level) programming content (algorithmic structures, working with input / output, basic code tracking skills, editing and creating an algorithmic solution). It can also be seen from the previous section that not all pupils were familiar with the same teaching content.

To help improve pupil motivation and ability to understand the task, each task represents a puzzle where the pupil helps a main character solve a problem (Lee and Ko, 2011). Puzzles are meant to assess one or more computational thinking concepts concealed in the puzzles and selected and aligned with the expected learning outcomes (see Table 2) and domain modelling (Table 3). CT practice and concepts which express abstraction, algorithmic thinking, and decomposition used in this research are presented in Fig. 2. This classification describes the way in which abstraction, algorithmic thinking, and decomposition are used in assessment. It represents authors' personal view influenced by the requirements of the Croatian informatics curriculum (Ministry of Education, 2018),

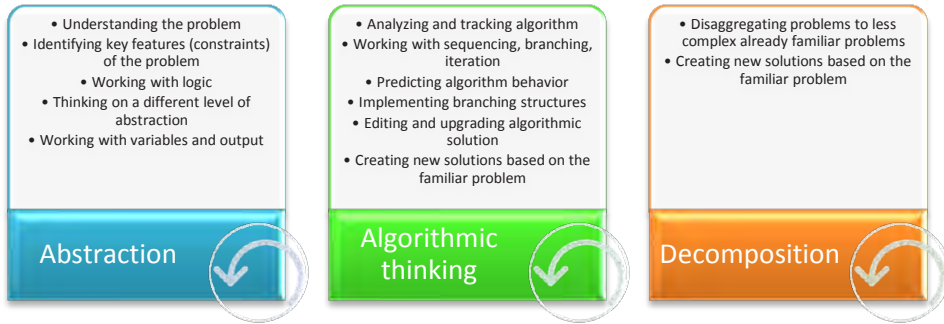


Fig. 2. Practices and concepts: abstraction, algorithmic thinking, and decomposition, respectively used in the assessment.

authors' attitudes and experience as well as other relevant literature (Bienkowski, Snow, Rutstein, and Grover, 2015, December; Brennan and Resnick, 2012; ISTE and CSTA, 2011; Grover, 2020). This classification is the basis for creating design patterns (Table 5) and the structure of the assessment tool (Table 6).

This assessment model aims to create an approach which will create enough space to evaluate computational thinking concepts per se. All the tasks of the measuring instrument are interconnected by a common context: the same story, in which the main character, Maja, tries to find a flower in the Action Labyrinth. The assessment tool is implemented in the form of an online test (10 questions/tasks) in which the option to return to previously solved tasks is turned off to prevent finding the solution to the current task in any of the following tasks. Tasks are arranged by difficulty, from easiest to most difficult while maintaining the same context, throughout the assessment tool. The types of questions used in the assessment tool were:

- Multiple choice questions: mostly used for identification of fundamental misconceptions or an unsustainable mental model (misunderstanding of a variable as a changeable value, misunderstanding how an algorithmic structure works (for example if/if...else.../if...elif...), misunderstanding how a iteration structure works, misinterpretation of a given logical expression,...).
- Short answer questions.
- Essay question used to gather pupils' algorithmic solutions: mostly for upgrading the given algorithmic solution or creating new algorithmic solution (part of the solution).

Modelling the assessment area also describes the design patterns that will be used in the assessment (Table 5). Design patterns contain all or only some constructs (knowledge, skills, practice) that we want to assess and can be used as evidence (knowledge, practice) or serve as a basis for making evidence. Patterns can also be used as activities or tasks that aim to encourage the demonstration of certain competencies. They are general enough that they can direct assessment regardless of how it is implemented (computer assessment, traditional paper-pencil assessment, simulation assessment, game assessment, etc.).

Table 5
Design patterns used in the assessment tool

Concept	Design patterns
Abstraction	(A1) describe the basic features of the problem (A2) identify the limitations of the given problem (A3) evaluate logical expressions (A4) use logical operators (A5) create a logical expression for a given condition (problem) (A6) distinguish between constant and variable quantities in the algorithm/solution (A7) apply a variable in the algorithm to monitor the variable characteristics of the problem (A8) define and/or monitor the change in the value of the variable in the algorithm/solution
Algorithmic thinking	(AL1) monitor and predict the execution of an algorithm solution that does not contain loops (AL2) predict the execution of an algorithmic solution containing loops (AL3) recognize/identify the basic feature of the loop (repetition) and how to stop the loop (AL4) identify/identify parts of the algorithm that contain decisions (AL5) recognize/describe/distinguish how a simple and complex branching structure works (AL6) create a new algorithm (AL7) upgrade the algorithm due to an observed error or fulfilment of a given problem requirement
Decomposition	(D1) Identify parts of a given problem that are easier to solve, or we already know the solution to that part (D2) distinguish the sub-units of the algorithmic solution (D3) present the subprogram as the sub-whole of the algorithmic solution

Table 6 presents the assessment tool structure with the target and underlying CT concepts and practice included in each task. First column represents the order of tasks in the tool and second column type of the question used. Third column shows the target and underlying CT concepts used in a particular task of the tool. It is almost impossible to consider that when solving a task represented by an algorithmic solution, we don't expect the analysis and tracking of the algorithm in action. However, just as in mathematics, for example, when evaluating the solution of a linear equation with an unknown, we also rely on the skill of computing with basic mathematical operations, so when assessing computational thinking we will have to recognize the concepts and practices of computational thinking that are primarily assessed by the chosen task (target CT concepts and practices) as well as which concepts and practices are present in the background in the same task (underling CT concepts and skills). Those CT concepts and practices are chosen according to the descriptions of abstraction, algorithmic thinking and decomposition (Fig. 2), and design patterns of the task model (Table 5). Last column describes each task with design patterns used in the task (target and underlying).

Feedback for multiple choice and short answer questions was formulated automatically, while essay questions were manually evaluated by the researcher and an independent teacher specialist. One of the most important phases in developing a task model is an analysis of the evidence of the existence of CT concepts which we are looking for in pupils' solutions. Here, prior to each assessment, the teacher must think about the intended outcome of the assessment that they plan to develop, and which concepts,


Table 6
Structure of the measurement tool

Task number	Type of question	Concepts: Abstraction (A); Algorithmic thinking (AL_T); Decomposition (D)		Design patterns
		Target CT	Underlying CT	Included in the task Target/Underlying
Task 1	Short answer question	Understanding the problem (A)	Analysing and tracking algorithm (AL_T) Working with sequencing, branching, iteration (AL_T)	A1+A3 (AL2+AL3+AL4)*
Task 2	Multiple choice question	Identifying key features (constraints) of the problem (A)	Predicting algorithm behaviour (AL_T)	A2+A3 (AL2+AL3)*
Task 3	Essay question	Working with logic (A)	Implementing branching structures (AL_T)	A3+A5 (AL2+AL3+AL4+AL5+AL7)*
Task 4	Multiple choice question	Thinking on a different level of abstraction (A)	Understanding the problem (A)	A1+A2 (AL2+AL3+AL4)*
Task 5	Multiple choice question	Working with branching (AL_T)	Analysing and tracking algorithm (AL_T)	AL4+AL5 (AL1)*
Task 6	Multiple choice question	Predicting program behaviour (AL_T)	Analysing and tracking algorithm (AL_T)	AL2 (AL3)*
Task 7	Essay question	Editing and upgrading algorithmic solution (AL_T) Understanding of the problem (A)	Analysing and tracking algorithm (AL_T) Working with variables and output (A)	A2+AL7 (A6+A7+AL2+AL3)*
Task 8	Multiple choice question	Disaggregating problems to less complex already familiar problems (D)	Analysing and tracking algorithm (AL_T)	D1 (AL2)*
Task 9	Essay question	Creating new solutions based on the familiar problem (AL_T, D)	Thinking on a different level of abstraction (A) Working with variables and output (A)	D3 (A1+A3+A7)*
Task 10	Short answer question	Predicting program behaviour (AL_T) Working with variables and output (A)	Analysing and tracking algorithm (AL_T)	A2+A8+AL2 (AL3)*

skills, and practices are relevant to that chosen outcome. In this process, the expected evidence of the pupil’s partial or complete knowledge to be included in the model will be recognized.

Through one task example, we will demonstrate our process of discovery of evidence of pupils’ knowledge and present our assessment of that evidence. It is important to emphasize that every well-planned assessment could reveal the existence of nonvalid pupil mental models (mental representations of the observed concept that contain some

Maya decided to keep track of the number of steps she makes on her journey through the labyrinth. She recorded the number of steps in the value named `steps_number` (variable is positioned above the labyrinth in the picture). At the beginning of each movement through the labyrinth, the value of the `steps_number` is set to zero (0).



`steps_number`

Action Labyrinth:
 Set `steps_number` to 0
 While *not* (`flowerup` or `flower_right`) do:
 Walk

Action Walk:
 While *not* `obstacle_up`:
 Do 1 step up ↑
 If *not* `obstacle_right`:
 Do 1 step right →
 else:
 If *not* `obstacle_left`:
 Do 1 step left ←

Upgrade Maya's movement directions in the way that the `steps_number` value always presents the total number of steps taken. In the appropriate place in the Action Walk, you need to write your version of the commands which will increase the value of `steps_number` by 1 each time a step is taken.

Write your version of the Action Walk

Fig. 3. Task example (Task 7).

error or misconception that makes it difficult or impossible to upgrade the mental model and build/upgrade new knowledge about the observed concept). Such information is of great importance for the teacher as well as the pupil. For the teacher, it points out the topics in their teaching to which they should give greater emphasis in the future. For the pupil, it points to the problems in their understanding, and to which concepts they possess but need to change and correct in further work. Fig. 3 represents one task from the assessment tool, which will be used to demonstrate the gathering of evidence and measurement model.

Model of evidence and measurement

Design and application of high-quality assessment is very demanding and time consuming. In the ECD approach teachers themselves create assumptions or hypotheses about the pupils' knowledge that will be demonstrated through assessment. Algorithmic solutions are always difficult to evaluate. In the process of creating a model of evidence, it is crucial to explore all possible evidence of a pupil's knowledge without losing sight of the different ways in which it could be expressed within the context and the requirements of the task itself. Also, a coherent and precisely formulated model of evidence is a prerequisite for drawing valid conclusions regarding results (Kane, 2013). Table 7 presents evidence of different ways pupils' learning could be expressed while solving the task from Fig. 3.

The evidence varies from a situation where the pupil does not even try to do anything through several partial solutions and finally to a fully correct solution. The scoring

Table 7
Task evidence and measurement example (task 7)

Evidence	Detected pupils' problems	Score
<ul style="list-style-type: none"> • No change was made in the Action Walk 	<ul style="list-style-type: none"> • No understanding of the problem, problems in algorithmic thinking 	0
<ul style="list-style-type: none"> • There was some change made in the Action <i>Walk</i> which included change in the number of steps (go 4 steps, go 3 steps...), but the change didn't include counting the total number of steps made while walking 	<ul style="list-style-type: none"> • Problems in working with variables (abstraction), partial understanding of the problem 	0
<ul style="list-style-type: none"> • Incrementing the counter by 1 was done, but in the wrong place in the algorithm (for example in the Action Labyrinth) 	<ul style="list-style-type: none"> • Problems in algorithmic thinking (code tracing), partial understanding of the problem 	1
<ul style="list-style-type: none"> • Incrementing the counter by 1 was done, but not in all the necessary places in the algorithm (for example only while going right or while going up) 	<ul style="list-style-type: none"> • Partial understanding of the problem, problems in algorithmic thinking (code tracing) 	2
<ul style="list-style-type: none"> • Correct and necessary changes for incrementing the counter were made in all expected places in the algorithm. 	<ul style="list-style-type: none"> • No problem detected 	3

or measurement model is presented in the same table. While creating an evidence model, it is very important to focus on evidence of knowledge connected to the task; thus, we avoid evaluation of any displayed knowledge in the task solution that does not have any link to the task itself.

When creating a task and planning its assessment, it is important to determine which educational outcomes should be included in the task, and, accordingly, which evidence of pupil knowledge should be recognized and assessed. Table 7 presents such evidence, identifying for each type of evidence the recognized problem in the pupil's understanding or mental model. For example, let us consider a task where Maja can move through the labyrinth according to a predefined movement rule defined by the Labyrinth Action and the Walk Action. The pupil needs to upgrade the existing solution so that the total number of steps that Maja makes can be recorded. If the pupil makes no change in the initial (presented) solution, they probably do not know where or how to make the expected changes to the program. That is, this seems to be a case of complete incomprehension or misunderstanding of the problems and thus obviously a problem with algorithmic thinking ability with this particular task and set of rules. Further, if the pupil did make a change in the number of steps, defining it as a constant value rather than changing the number of steps with the help of a variable that at some point increments by one, it was assumed that the pupil did not recognize variables as the abstract concept which could track Maja's movement through the labyrinth (indicating problems in working with variables, or abstraction generally). Nevertheless, in such a case, the pupil obviously understood that some change regarding movement through the labyrinth had to be made (partial understanding of the problem).

Next table (Table 8) demonstrates mental models recognized from pupils' answers. This task proved to be one the most difficult for pupils (difficulty index 89.74), but with great strength to distinguish those who have adopted very well the observed concepts from those who expressed difficulties in understanding the problem, algorithmic thinking skills, and working with variables (discriminant index 0.41).

Table 8
Mental models recognized by analysis of pupils' solutions – task 7

Mental model	Number of pupils	
The pupil does not know where or how to upgrade the program.	284	80.7%
The pupil does not recognize the variables as values by which the movement of the Maya through the labyrinth can be traced, He understands that some change needs to be made in addition to the movement itself.		
The pupil recognizes the need to use the variable to track Maya's movement through the labyrinth but does not recognize the appropriate place in the program for such a change.	40	11.4%
The pupil recognizes the need to use the variable and successfully applies it in some of the expected places.	11	3.1%
The pupil successfully applies the variables in a specific problem.	17	4.8%

This kind of analysis of evidence helps us in the creation of an evidence model for similar tasks. We could create several task examples where it is necessary to create improvements in character movement, adding some new possibilities like skipping steps, rotating, and so on. While analysing pupils' answers, it is crucial to know in which computational concepts we can expect to find appropriate evidence of acquisition of target concepts. For example, one task's goal could be to reveal the pupil's algorithmic thinking level by comparing two computational solutions or by creating a standalone computational solution in detail. Another task's goal could be to look for the pupil's ability to analyse and understand the problem or to decompose it into less complex parts. Both tasks are trying to find out whether the pupil is familiar with and can manipulate different levels of abstraction.

3.3. Assessment Implementation/Delivery

In our model, as noted, the background of each task is a problem situation involving Maja's search for a flower in the labyrinth. Task context must be preserved throughout the assessment to facilitate understanding, and easier tasks come at the beginning of the tool. To facilitate understanding of the moving instructions all algorithmic solutions used in the tasks must be written in a language very similar to a spoken language with the structure that resembles a pseudo language. In addition, the solution to a given task, or part of it, may be incorporated into later tasks. For that reason, the model excludes the possibility of returning to previously solved tasks for reconsideration or re-solving, as the solutions of earlier tasks are incorporated into later tasks (Fig. 4).

This model of assessment was first tested in exploratory research during the 2016/2017 school year with an online assessment adapted for the Python programming language. Further, pilot research was conducted during the 2017/2018 school year, in which implementation of this CT assessment model was enabled by using online testing within the Loomen LMS (Learning Management System). Pupils' access to Loomen is based on

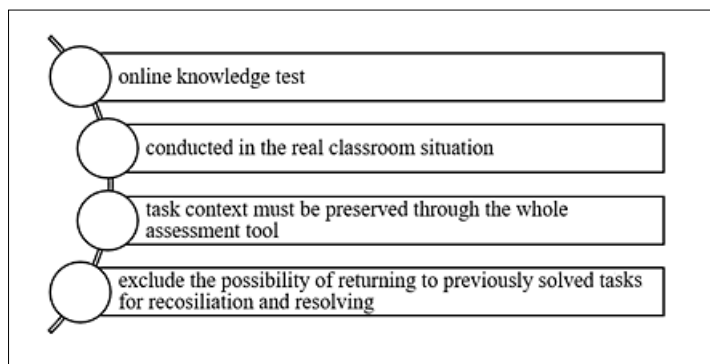


Fig. 4. Delivery of assessment tool.

their unique user data, provided to every middle and secondary pupil in the Republic of Croatia, so that the authenticity of the research participants' data was preserved. Thus, Loomen was also used in the main study. Pupils were invited to join the assessment through an email invitation from their CS teacher, who agreed to participate.

Since the study material was in accordance with the current CS Curriculum in Croatia, it is in line with the requirements of “in situ” research, so no special consent from participants was necessary.

4. Results

4.1. Statistical Characteristics of the Assessment

Basic features of this assessment model were initially tested through introductory research, as noted, with a measuring instrument adapted to the Python programming language. The promising results of this initial assessment encouraged the creation of the new tool- and environment-independent assessment model and a valid CT assessment model based on the ECD approach (Bubica and Boljat, 2018).

Research using the final version of the assessment tool (March/June 2018) was conducted among 407 pupils from ten middle schools in eight cities. After analysis of the assessment results, 27 pupils who submitted no solutions were excluded from the research. In addition, 28 pupils were excluded as they spent less than 10 minutes finishing their assessment and most of the tasks were unanswered. It was assumed that these pupils did not take the assessment seriously as they did not submit their solutions at all or they completed the assessment in less than ten minutes, which wasn't even enough for the first reading and understanding of assessment tasks. Thus, the analysed results refer to 352 pupils (193 males, 159 female). Research data concerning basic test statistics such as difficulty and discrimination index were explored. This assessment tool showed a satisfactory degree of internal reliability ($\alpha = 0.630$). In terms of Cohen, Manion and Morrison (Cohen *et al.*, 2000) in general seven tasks had acceptable difficulty

Table 9
Test statistics regarding difficulty and discrimination index

Task	Difficulty index	Discrimination index	Task	Difficulty index	Discrimination index
1	59.42	0.32	6	57.10	0.29
2	28.12*	0.29	7	89.74**	0.41
3	44.03	0.38	8	51.19	0.36
4	59.42	0.11	9	85.72**	0.45
5	46.68	0.27	10	93.90**	0.16

* probably very easy task; ** probably very difficult task

(33.00–67.00), one task was probably too easy (28.12) and three tasks seemed to be very difficult (85.72–93.90) (see Table 9).

The initial assumption of this research is that this assessment tool should highlight three basic concepts: algorithmic thinking, abstraction, and decomposition. The degree of task intercorrelation was explored in order to find out whether there was a way to group these tasks by the relationships among them, based on some common term in the background. The main characteristics of the research, the number of respondents ($N = 352$) and the number of tasks (10), met the basic prerequisites of factor analysis (35:1 > 30:1) (Hair, Black, and Babin, 2014). Factor analysis was applied to detect possible interrelationships between tasks (Cohen *et al.*, 2000), using the Kaiser-Meyer-Olkin measure of sampling adequacy parameter ($0.756 > 0.5$) and Bartlett's test of sphericity ($p \leq 0.001$). The results showed that tasks were sufficiently related (determinant = $.406 > 0.00001$, $p \leq 0.001$) but not excessively interdependent (intercorrelation < 0.8).

Overall, values for eight tasks mostly satisfied the criterion of good task association (that is, ~ 0.2 – 0.8 , $p < 0.05$). Two tasks (task 4, task 10) showed a greater correlation with other tasks than most tasks did, but at an insufficiently significant level ($p > 0.05$). It should also be noted that these tasks had the lowest index of discrimination (< 0.16).

Results of the factor analysis confirmed the initial assumption of three components in which assessment tasks could be grouped what will be addressed further in the discussion section.

To reinforce the reliability of the evaluation tool, an additional grader was introduced for essay assignments, which were not evaluated automatically. The additional grader was an experienced teacher, who independently evaluated essay tasks (task 3, task 7, task 9) according to the presented model of evidence and measurement. Comparison of the researchers' and the additional grader's grading showed a very high degree of matching in all three tasks (task 3: 96%; task 7: 97%; task 9: 97%).

4.2. Influence of Some Factors (Gender, Academic Achievement, Programming Experience) on Assessment Results

It has been researched whether there is a correlation of factors such as gender, math and general academic achievement with the success in this assessment given that such

factors have shown correlation with success in programming in some studies (Wilson and Shrock, 2001; Bubica and Boljat, Predictors of Nvices Programmers’ Performance, 2014). Pupils’ data concerning their general academic and math achievement showed that this research sample did not represent a normal distribution, so a nonparametric test was used to explore correlations among data (one-sample Kolmogorov-Smirnov test $p = .001$). Results regarding general academic achievement showed very strong positive correlation with math achievement (Spearman’s $r = 0.803$, $p \leq 0.001$). Factors like math and general academic achievement had medium positive correlations with achievement on this CT assessment (Spearman’s $r = 0.425$, $r = 0.429$, $p \leq 0.001$).

One of the important features of an assessment is to determine whether there are differences between the results on the given assessment achieved by the boys and girls. The research results showed that there was no correlation between the variables pupil’s gender and their success in CT assessment (Spearman’s $\rho = -0.051$, $p = 0.337 > 0.05$). Such a result is consistent with existing research (Authors, 2014; Pillay and Jugoo, 2005) and speaks in favour of the claim that the created assessment tool might be equally suitable for boys or girls. At the time when there is a significant debate about the lack of women in informatics and their lack of interest in STEM in general, it is important to find ways to increase girls’ interest in informatics during education, for example, at least by creating equal learning environments in both, teaching and assessment process (Writers, 2021).

Further analysis was done to explore the (in)sensitivity of the proposed CT tool regarding previous programming language knowledge. A normality test of pupils’ assessment results showed that there was not a normal distribution of data (Kolmogorov-Smirnov, skewness = 0.388, kurtosis = -0.041, $p \leq 0.001$), so nonparametric methods were used. Given that some of the pupils had their first programming encounter through the Scratch graphical programming environment and continued their work in Python or Logo, we examined the difference between the results with respondents placed in one of four groups depending on the programming language or combination of the programming languages they used while learning CT concepts (Table 10). The Kruskal-Wallis method showed that there was no significant difference between the scores achieved by pupils depending on the programming language or combination used.

This is consistent with the results of earlier research, and it also confirmed the initial criteria that the tool is independent of the observed programming languages (Allert, 2004; Byrne and Lyons, 2001; Pillay and Jugoo, 2005). Further grouping of data investigated the impact of learning an individual programming language on the pupils’ success.

Table 10
Difference between the scores depending on the programming languages used (Kruskal-Wallis)

Grouping of pupils according to the programming language learned	“first Scratch then Python”	“first Scratch then Logo,”	“only Logo”	“only Python”
mean rank	176.42	203.21	171.35	175.14

$\chi^2(2) = 2.458$,
 $p = 0.483$

The results showed no statistically significant difference between Scratch (Mann-Whitney 7581.500, $p = 0.240$), Logo (Mann-Whitney 14692, $p = 0.788$), and Python (Mann-Whitney 14692, $p = 0.788$) groups meaning that it is not expected that pupils who have learned particular programming language will therefore achieve better or worse results in this assessment.

4.3. Construct Validity of the Assessment Tool: Comparison to the Selected Set of Bebras Challenge Tasks

In previous sections, the correlation between the learning outcomes of the new CS Croatian curriculum and CT assessment was described. Given that the Bebras challenge was already present in schools in the Republic of Croatia, it was appropriate to determine if there was correlation between the two CT assessments and if possible, determine which of the two assessments is better aligned with the subject curriculum. In this way, the construct validity of the developed assessment tool was also examined. In a Bebras task there is the same main character, a beaver (*dabar* in Croatian), who is trying to solve a different presented problem/puzzle in a different context (story) each time. Every year, computer science education specialists prepare and distribute tasks for the Bebras challenge, keeping in mind that each task must contain CT appropriate to the pupils' age. In Croatia, the Bebras initiative was conducted as an online competition (15 questions) through the Loomen LMS (November 2016) distributed in middle and secondary schools ($N = 1892$). Table 11 compares the results of the observed assessments. As the observed concepts had to be aligned with the CS curriculum, an appropriate subset of Bebras challenge tasks which most closely coincided with the proposed CT assessment and Croatian CS curriculum was selected and the relationship between the two assessments was explored.

The Bebras challenge assessment consisted of fifteen tasks but only six of them were in line with the selected learning outcomes involved in this assessment (see Table 2). Concepts/skills such as decomposition, working with different levels of abstraction, creating new algorithmic solutions, and upgrading existing algorithmic solutions were not included in Bebras evaluation tasks. The selected Bebras challenge tasks showed moderate correlation with CT assessment ($r = 0.495$, $p = 0.001$), which speaks in favor

Table 11
Basic statistical features of Bebras and CT evaluations for the pupil samples

	Bebras assessment	CT assessment	Results of pupils who participated in both assessments	
			Bebras	CT
Pupil sample	$N = 1892$	$N = 358$	$N = 49$	$N = 49$
Reliability (Cronbach's alpha)	$\alpha = 0,48$	$\alpha = 0,63$	$\alpha = 0,325$	$\alpha = 0,722$
			Correlation (Spearman's r) 0,495 ($p = 0.01$)	

of the construct validity of the proposed CT assessment. Low values of the reliability coefficient ($N = 1892$, $\alpha = 0.48$; $N = 49$, $\alpha = 0.325$) indicated that the selected set of Bebras challenge tasks cannot be considered a reliable way of assessing CT concepts. Still, for more serious conclusions regarding the assessment of computational thinking with Bebras challenge tasks and their connection with the created assessment tool, research should be conducted on a larger subset of Bebras challenge tasks from several years as well as with greater sample of pupils involved.

5. Discussion

The main goal of this research was to assess acquisition of concepts of computational thinking: abstraction, algorithmic thinking, and decomposition, among pupils in middle school. According to the developed assessment model, based on the ECD approach, the appropriate tasks of the assessment tool were created. The assessment tasks were also aligned with the 6th-grade learning outcomes of the new informatics curriculum. Basic assessment statistics considering task structure and context (Table 11) showed that analysing and understanding algorithms, anticipating algorithm behaviour, and upgrading existing algorithms or creating completely new algorithms represented the greatest challenge for pupils. Further, those tasks, including concepts of algorithmic thinking and decomposition, also showed the highest degree of discrimination, that is, the great power of separating successful from less successful pupils, which is very important in any assessment.

Factor analysis was conducted to explore the possibility that the assessment tool could measure acquisition of CT concepts: abstraction, algorithmic thinking, and decomposition; the strength of the correlation of each task with each factor indicated by factor analysis was investigated. The obtained results were in accordance with the proposed model of the assessment tool (Table 6), which distinguishes target and underlying CT knowledge and practice. The results of factor analysis clearly distinguished factors from one another based on which type of practice and knowledge of algorithmic thinking prevailed (editing and upgrading algorithmic solutions, creating new solutions based on a familiar problem, predicting program behaviour). Target CT knowledge and practice correlated to these factors were grouped according to factor loadings (stronger, moderate, weaker), where according to sample size, factor loadings equal to or greater than 0.3 are considered significant (Hair *et al.*, 2014) (see Table 12).

According to the results of the factor analysis, the strongest connection with the first factor showed skills of algorithmic thinking, but also decomposition. The less powerful yet still significant correlation to the first factor demonstrated some skills that are related to abstraction (working with variables and output, understanding the problem, identifying key features of the problem). The correlation with the second factor showed skills and knowledge that are related to abstraction (understanding the problem, working with logic); the smaller but still significant correlation to the second factor showed the relevance of the skill of working with branching structure (algorithmic thinking). Although the factor analysis pointed out the existence of a third factor, due to the insuf-

Table 12
Grouping tasks according to factor analysis results

Factor loading	Factor 1: Abstraction	Factor 2: Algorithmic thinking	Ungrouped task*
Stronger factor loading (> 0.62)	<ul style="list-style-type: none"> • Editing and upgrading algorithmic solution (Algorithmic thinking, task 7) • Creating new solutions based on the familiar problem (Algorithmic thinking, decomposition, task 9) • Understanding the problem (Abstraction, task 7) 	<ul style="list-style-type: none"> • Understanding of the problem (Abstraction, task 1) • Working with logic (Abstraction, task 3) 	<ul style="list-style-type: none"> • Working with branching structure (Algorithmic thinking, task 4)
Moderate factor loading (0.52 <...< 0.62)	<ul style="list-style-type: none"> • Predicting program behaviour (Algorithmic thinking, task 6) • Disaggregating problems to less complex already familiar problems (Decomposition, task 8) • Working with variables and output (Abstraction, task 10) • Understanding the problem (Abstraction, task 7) 	<ul style="list-style-type: none"> • Working with a branching structure (Algorithmic thinking, task 5) 	
Lower factor loading (0.3<...<0.52)	<ul style="list-style-type: none"> • Identifying key features (constraints) of the problem (Abstraction, task 2) 		

* ungrouped task because of insufficient number of tasks within factor

ficient number of tasks related to this factor (task 4), it is not appropriate to consider it a separate factor in this analysis. The ungrouped task (task 4) highlighted skills of working with branching structures (simple and complex). This is a surprising result, since these skills could be expected to have a strong connection with the first factor, in which the skills and knowledge related to algorithmic thinking prevail. Since this task (task 4) already showed extremely bad discrimination, it is questionable how much the results influenced the odd deployment of the highlighted skill – working with branching structures – given the prominent effects of the factors. The third CT concept considered, decomposition, was not highlighted as an individual factor in this analysis, but we can nevertheless strongly associate it with the concept of algorithmic thinking and the concept of abstraction. One reason could be the fact that decomposition was highlighted by only two of the total ten questions; another could be that a basic premise for successfully solving these two tasks (disaggregating problems to less complex already familiar problems) was that the pupil had to be skilled with certain algorithmic thinking abilities (creating new solutions based on familiar problems, editing and upgrading algorithmic solutions) as well as with skills and knowledge connected to abstraction (understanding of the problem, thinking at different levels of abstraction, working with variables). Given the above results, we can conclude that the concepts of algorithmic thinking and abstraction were clearly highlighted in this assessment, while the concept of decomposition was not clearly recognized as a separate factor.

One of the goals of this research was to investigate and establish the assessment tool's independence of the programming language used by the pupils. Considering that several different programming languages are included in the teaching process in Croatia, it is of utmost importance that the created model be compatible with all these languages. When creating test tasks, it was understood that the way of writing should avoid making task language and images dependent on the specific syntax and structure of some programming language. At the same time, the task had to be intuitive and to not distract pupils' attention from solving the problem. Since research participants were familiar with different programming languages, results which indicated no significant difference in the scores achieved by pupils regardless of the programming languages they had learned were of great importance. These results are consistent with the relevant research (Bubica and Boljat, 2014; Pillay and Jugoo, 2005). The fact that the tasks were written in a language that is very similar to the spoken language of the pupils certainly made it easier to understand the tasks better. Further, since the tool is intended for real teaching practice, it is very important that gender was not highlighted as a factor that can significantly affect this CT assessment. The achieved results are in line with previous relevant research (Bubica and Boljat, 2014; Wilson and Shrock, 2001).

In further quality analysis of the created CT assessment, it is important to consider the question of its reliability (how dependably or consistently a test measures a characteristic). As mentioned earlier this assessment tool showed a satisfactory degree of internal reliability ($\alpha = 0.630$). It is generally accepted that values of reliability coefficient (Cronbach's alpha) between 0.7 and 0.6 are considered acceptable (Cohen, Manion, and Morrison, 2007; Nunnally, 1978; Pallant, 2005; Pallant, 2011) and above 0.6 (Taber, 2017) barely acceptable (satisfactory). The higher the observed coefficient is, the more likely it is that the same results will be repeated when re-applying the assessment tool. The value of the Cronbach's alpha coefficient depends on the number of tasks in the given instrument; a low value, for example 0.5, is not uncommon for instruments up to 10 tasks. In such situations, it is common to analyse the corrected correlations among test tasks; for two tasks in the present study (task 4, $r = 0.104$; task 10, $r = 0.150$), it can be concluded that they do not measure the same thing as the rest of the instrument, as their correlation values deviate from the optimal values among the (other) tasks (Briggs and Cheek, 1986). Further, if these tasks were removed from the instrument, the value of the Cronbach alpha coefficient respectively would not change at all (task 10) or would increase slightly (task 4). As one of these items (task 4) showed extremely poor discrimination value (0.1052), removal or significant improvement of it should be a serious consideration. Also, adding more tasks to the assessment, for example addressing the concept of decomposition, should be considered, since more tasks improve test reliability.

Finally, the possible difficulties or weaknesses of the proposed model should certainly be pointed out. In the modelling phase, design patterns were presented as the foundation for the development of each assessment task. The patterns are the result of personal authors' attitudes / thoughts / experiences as well of analysis of relevant literature. In further work, greater emphasis will be placed on the study of the presented patterns and their possible upgrading and refinement. The most demanding and sensitive part of the model is certainly recognizing and separating targeted and underlying CT concepts and

skills while creating a task due to their big interconnections. This could make the process of creating tasks for teachers quite difficult and challenging.

Furthermore, the assessment model suggests its online application. This could be considered as an advantage of the model because it offers an easy approach and application of assessment. At the same time, an online assessment excludes the possibility of returning to previously solved tasks for reconciliation and resolving. This limitation makes it impossible to easily transfer the online assessment tasks to a paper-pencil form of assessment which could be considered its disadvantage.

6. Future work

In the presented work, the emphasis was on summative assessment. Further work will mainly focus on the analysis of other data that were collected during the research like feedback from the teachers and pupils. The possibility of applying the described assessment model for formative purposes will be explored, primarily by analysing collected evidence of pupil knowledge. This assessment used evidence of pupil's knowledge, which was identified through their responses and associated them with mental models of the observed CT concept. For that reason, it is possible to conduct a deeper analysis of mental models adopted by pupils as well as of difficulties that need more emphasis in further learning. The concept abstraction will be analysed through the way pupils deal with different levels of abstraction (understanding the problem), also in how pupils use variables and logical values; the concept algorithmic thinking will be analysed through pupils' ability to track and analyse algorithmic solutions and well as through their ability to apply different kinds of branching structures. Formative assessment of the CT concepts will be presented in the form of a tree-scale rubric (need much work, partially successful, successful). Such formative feedback seeks to point out pupils' most common mistakes and difficulties – information of great importance for teachers when planning instruction as well as for pupils when provided as timely feedback during the learning process.

Collected qualitative data regarding the clarity of the questions, the presentation and structure of the tool, and other matters that emerge will be explored.

In addition to the already mentioned activities, there will be certainly more work in the future on improving the domain modelling, especially regarding design patterns as they represent important part of the proposed assessment model.

7. Conclusions

The beginning of educational reform in the Republic of Croatia started a process of creating new subject curricula and applying new teaching strategies and evaluations. The new K-12 CS curriculum, taking a learning outcome-based approach, emphasizes the importance of knowing concepts such as programming, algorithms, and data structures and introduces the development of computational thinking with the primary goal

of encouraging pupils' creativity instead of only teaching them how to use information and communication technology. Introducing computational thinking in subject curricula entails the introduction of new teaching methods and materials and presents teachers with the challenge of evaluating the outcomes. There is still not enough research on CT evaluation to provide teachers with enough support in the field. Due to the increasing emphasis on formative evaluation in everyday teaching practice, it is especially important to explore the potential of assessment tools to be applied in different forms of assessment (such as assessment for learning or assessment as learning), for example by identifying valid and invalid mental models of observed concepts.

This study offers a model of CT assessment consistent with the new Croatian K-12 CS curriculum. To answer the research questions, the research tried to find out how suitable the proposed ECD model of CT assessment is for educational practice. Due to the diversity of programming languages used in Croatian schools, it is extremely important that the success that pupils achieve in this assessment not be dependent on the programming language through which they have learned basic CT concepts. Our approach is independent of programming language (already present in school) and equally appropriate for boys and girls. The presented CT assessment model showed an acceptable reliability index; the coincidence of the results of grading by the researcher and the additional independent grader speaks for the reliability of the tool and ease of the use in the classroom.

To investigate the construct validity of the proposed tool, the results for the selected set of pupils were compared with their success on the Bebras challenge tasks. Although the proposed assessment instrument showed better characteristics (coefficient of reliability, consistency with the outcomes of the Croatian CS curriculum, interconnection of tool tasks, etc.) than assessing CT with selected set of Bebras challenge tasks; for more reliable conclusions research with a larger pupil sample should be conducted. Furthermore, such a result may also have been influenced by the choice of Bebras challenge tasks that were applied in the Republic of Croatia in that period. Some other tasks that did not reach the pupils at all might discover some other interrelationship of the two observed types of tasks.

The results of this research showed that although creating an ECD approach assessment is a time consuming and demanding job, it allows us to create assessments that are strongly related to the subject curriculum and offer evidence argument for difficult-to-measure concepts.

Conflict of interest

The authors declare that there is no conflict of interest regarding the publication of this paper.

Acknowledgments

Special thanks to the teachers and experts for their outstanding help and participation in the research. An earlier version of this research was presented as a conference paper, and

the abstract was published in *Proceedings of the International Conference on Computational Thinking Education 2018* (Hong Kong: The Education University of Hong Kong) (Bubica and Boljat, 2018).

References

- Allert, J. (2004). Learning Style and Factors Contributing to Success in an Introductory Computer Science Course,”. *Proceedings of the IEEE International Conference on Advanced Learning Technologies – (ICALT’04, 2004*.
- Almond, R.G., Steinberg, L.S., Mislevy, R.J. (2002, Volume 1, Number 5). Enhancing the Design and Delivery of Assessment Systems: A Four Process Architecture. *The Journal of Technology, Learning, and Assessment*.
- Ambrosio, A.P., Almeida, L. d. (2014). Exploring Core Cognitive Skills of Computational Thinking. *Psychology of Programming Interest Group Annual Conference 2014* (pp. 25–34). Brighton, UK: PPIG, University of Sussex, 2014.
- Anderson, L.W., Krathwohl, D.R., Bloom, B.S. (2001). *A Taxonomy for Learning, Teaching, and Assessing : A Revision of Bloom’s Taxonomy of Educational Objectives / editors, Lorin W. Anderson, David Krathwohl ; contributors, Peter W. Airasian ... [et al.]*. New York : Complete ed. New York: Longman, 2001. Print.
- Araujo, A.L., Santos, J.S., Andrade, W.L., Guerrero, D.D., Dagienė, V. (2017). Exploring computational thinking assessment in introductory programming courses. *2017 IEEE Frontiers in Education Conference (FIE)*, (pp. 1–9). Indianapolis, IN, 2017: pp. 1–9.
- Astrachan, O., Hambruch, S., Peckham, J., Settle, A. (2009). The present and the Future of Computational Thinking. *ACM 978-1-60558-183-5/09/03*, pp. 549–550. Chattanooga, Tennessee, USA.
- Ben-Ari, M. (1998). Constructivism in Computer Science Education. *SIGCSE’98*. Atlanta, GA, USA.
- Bienkowski, M., Snow, E., Rutstein, D., Grover, S. (2015, December). *Assessment Design Patterns for Computational Thinking Practices in Secondary Computer Science: A First Look*. Menlo Park, CA: SRI Education.
- Brennan, K., Resnick, M. (2012). *New Frameworks for Studying and Assessing the Development of Computational Thinking*.
- Briggs, S.R., Cheek, J.M. (1986). The role of factor analysis in the development and evaluation of personality scales. *Journal of Personality*, 54, pp. 106–148.
- Brodanac, P., Bubica, N., Kralj, L., Markučić, Z., Mirković, M., Rubić, M., Sudarević, D. (2016, February). *Computer Science National Curriculum – proposal*. Retrieved from kurikulum.hr: <http://www.kurikulum.hr/wp-content/uploads/2016/03/Informatika.pdf>
- Bubica, N., Boljat, I. (2014). Predictors of Nvices Programmers’ Performance. *ICERI2014 Proceedings*, pp. 1536–1545. Seville, Spain.
- Bubica, N., Boljat, I. (2018). Assessment of Computational Thinking. *CTE2018*. Hong Kong.
- Bundy, A. (2007). Computational thinking is pervasive. *Journal of Scientific and Practical Computing*, 1(2), 67–69.
- Byrne, P., Lyons, G. (2001). “The Effect of Student Attributes on Success in Programming,”. *ITiCSE 2001*. 6101 Canterbury, UK.
- Cohen, L., Manion, L., Morrison, K. (2000). *Research Methods in Education, 5th Edition*. London: Routledge Falmer.
- Cohen, L., Manion, L., Morrison, K. (2007). *Research Methods in Education, sixth edition*. New York, USA: Routledge.
- Cszmadia, A., Curzon, P.P., Dorling, M., Humphreys, S., Ng, T., Selby, D.C., Woollard, D.J. (2015). *Computational Thinking – a Guide for Teachers*. Computing At School.
- CSTA. (2016). *CSTA K–12 CS Standards*. Retrieved from CSTA: <https://www.csteachers.org/page/standards>
- Curran, J. R. (2019). Coding and Computational Thinking – What is the evidence? (report) . *NSW Department of Education*.
- Dagienė, V., Futschek, G. (2019). On the way to constructionist learning of computational thinking in regular school settings. *Constructivist Foundations* 14(3), 231–233.
- Dagienė, V., Stupurienė, G. (2016). Bebras – a Sustainable Community Building Model for the Concept Based Learning of Informatics and Computational Thinking. *Vilnius University* , 25–44.

- Denning, P.J. (2009, June). The Profession of IT – Beyond Computational Thinking. *Communications of the ACM*, 52(6), 28–30.
- Denning, P.J. (2010). The Great Principles of Computing. *American Scientist*, 98, 369–372.
- Dorling, M., Walker, J. M. (2014). *Computing Progression Pathways*. Retrieved from <http://community.computingatschool.org.uk/resources/1692>
- Dr Scratch. (2021, 4 17). Retrieved from Dr. Scratch – Analyze your Scratch programs here: <http://www.drscratch.org/>
- Fincher, S. (1999). What are We Doing When We Teach Programming? *29th ASEE/IEEE Frontiers in Education Conference*. San Juan, Puerto Rico.
- Grover, S. (2020). Designing an Assessment for Introductory Programming Concepts in Middle School Computer Science. *SIGCSE '20*. Portland, OR, USA: ACM 978-1-4503-6793-6/20/03.
- Guzdial, M. (2008, August). Paving the way for the Computational Thinking. *Communications of the ACM*, 51(8), 25–27.
- Hair, J.J., Black, W.C., Babin, B.J., Anderson, R.E. (2014). *Multivariate Data Analysis, seventh edition*. England; Harlow: Pearson Education Limited.
- Hendrickson, A., Ewing, M., Kaliski, P., Huff, K. (2013). Evidence – Centered Design: Recommendations for Implementation and Practice. *Journal of Applied Testing Technology, JATT*, volume 14, Association of Test Publishers.
- ISTE, CSTA. (2011). *CSTeachers*. Retrieved from Computational Thinking resources: <https://c.ymcdn.com/sites/www.csteachers.org/resource/resmgr/CompThinkingFlyer.pdf>
- Kane, M. (2013). The Argument-Based Approach to Validation. *School Psychology Review*, pp. 448–457.
- Lee, M.J., Ko, A.J. (2011). Personifying Programming Tool Feedback Improves Novice Programmers' Learning. *ICER'11*. ProvidenceRI, USA.
- Leron, U., Hazzan, O. (1998). Computers and applied constructivism. In: J. D.-T. Tinsley D., *Information and Communications Technologies in School Mathematics* (pp. 195–203). Boston, MA: IFIP – The International Federation for Information Processing. Springer, https://doi.org/10.1007/978-0-387-35287-9_23.
- Lye, S., Koh, J. (2014). Review on teaching and learning of computational thinking through programming: what is next for K-12? *Computers in Human Behaviour*, 41, 51–61.
- Microsoft Research (2009). *Kodu Game Lab – 3D game programming for kids*. Retrieved from Kodu Game Lab: <http://www.kodugamelab.com/>
- Ministry of Education. (2018). *Informatics subject curriculum*. Retrieved from Nacionalni kurikulum: <https://mzo.hr/hr/rubrike/predmetni-kurikulumi>
- Ministry of Education. (2006). *Curriculum program for elementary school*. Retrieved from Ministry of Education – elementary education: https://mzo.hr/sites/default/files/dokumenti/2017/06/nas-tavni-plan-i-program-za-os_2006.pdf
- Ministry of Education. (2018, 2 17). *Computational thinking and Programming*. Retrieved from Ministry of Education: <https://loomen.carnet.hr/mod/feedback/view.php?id=302617>
- Mislevy, R.J., Haertel, G. (2006). Implications for evidence-centered design for educational assessment. *Educational Measurement: Issues and Practice*, 25, 6–20.
- Mislevy, R.J., Almond, R.G., Lukas, J.F. (July, 2003). *A Brief Introduction to Evidence-centered Design*. NJ 08541: Research & Development Division Princeton.
- Mislevy, R.J., Riconscente, M.M. (2005). *Evidence-Centered Assessment Design: Layers, Structures, and Terminology*. SRI International.
- Moon, J., Do, J., Lee, D., Choi, G.W. (2020, February 13). A conceptual framework for teaching computational thinking in personalized OERs. *Smart Learning Environments, SpringerOpen*.
- Moreno-León, J., Robles, G., Román-González, M. (2015, Sep 15). Dr. Scratch: Automatic Analysis of Scratch Projects to Assess and Foster Computational Thinking. *RED-Revista de Educación a Distancia. Número 46*, pp. 1–23.
- Nunnally, J.C. (1978). *Psychometric Theory (2nd ed.)*. New York: McGraw-Hill.
- Pallant, J. (2005). *SPSS Survival Guide: A Step by Step Guide to Data Analysis Using SPSS for Windows 3rd Edition*. New York: Open University Press.
- Pallant, J. (2011). *SPSS Survival Manual: A Step by Step Guide to Data Analysis Using the SPSS Program. 4th Edition*. Berkshire: Allen & Unwin.
- Papert, S. (1980). *Mindstorms: Children, Computers, and Powerful Ideas*. New Year: ACM. 0-465-04627-4.
- Pellegrino, J. W. (2020). Important Considerations for Assessment to Function in the Service of Education. *Educational Measurement: Issues and Practice, Vol. 00, No. 0*, pp. 1–5.
- Pillay, N., Jugoo, V.R. (2005). An Investigation into Student Characteristics Affecting Novice Programming

- Performance. *inroads – The SIGCSE Bulletin*, 37(4), 107–110.
- Roman-Gonzales, M. (2014). Aprender a programar ‘apps’ como enriquecimiento curricular en alumnado de alta capacidad. *Bordon, Revista de Pedagogia*, 66(4), 135–155.
- Román-González, M. (2015). Computational Thinking Test: Design Guidelines and Content Validation. *Proceedings of the 7th Annual International Conference on Education and New Learning Technologies (EDULEARN 2015)*, pp. 2436–2444. Barcelona, Spain.
- Taber, K.S. (2017). The Use of Cronbach’s Alpha When Developing and Reporting Research Instruments in Science Education. *Research in Science Education*.
- Werner, L., Denner, J., Campe, S. (2012). The Fairy Performance Assessment: Measuring Computational Thinking in Middle School. *SIGCSE’12*. Raleigh, North Carolina, USA: Copyright 2012 ACM.
- Wiebe, E., London, J., Aksit, O., Mott, B.W., Boyer, K.E., Lester, J.C. (2019). Development of a Lean Computational Thinking Abilities Assessment for Middle Grades Students. *SIGCSE ‘19*. Minneapolis, MN, USA: ACM ISBN 978-1-4503-5890-3/19/0.
- Wilson, B.C., Shrock, S. (2001). Contributing to Success in an Introductory Computer Science Course: A Study of Twelve Factors. *SIGCSE*. Charlotte, NC, USA.
- Wing, J.M. (2006, March). Computational thinking. *Communication of the ACM*, 49(3), 33–35.
- Wing, J.M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 366 (1881).
- Wing, J.M. (2010, November 17). Retrieved from [www.cs.cmu.edu: https://www.cs.cmu.edu/~CompThink/resources/TheLinkWing.pdf](https://www.cs.cmu.edu/~CompThink/resources/TheLinkWing.pdf)
- Writers, S. (2021, May 5). *Women in Computer Science: Getting Involved in STEM*. Retrieved from Computer Science org: <https://www.computerscience.org/resources/women-in-computer-science/>
- Yadav, A., Gretter, S., Jon Good, McLean, T. (2017). Computational Thinking in Teacher Education. *Emerging Research, Practice, and Policy on Computational Thinking, Educational Communications and Technology: Issues and Innovations*, pp. 205–219.
- Yadav, A., Stephenson, C., Hong, H. (April 2017). Computational Thinking for Teacher Education. *Communications of the ACM* 55, 60(4), 55–62.

N. Bubica (Mokosica Middle School, Dubrovnik, Croatia) received her doctorate in educational sciences from the Faculty of Science in Split, Croatia in 2022 – CS and Math Teacher graduated at University of Split, Faculty of Science. Engaged by the Ministry of Science and Education to work with the Expert working groups for the creation and later introduction of new CS curricula in the K12 education. Student at the postgraduate doctoral study at the University of Split, Faculty of Science: Education Research in Natural and Technical Science. Author and co-author of several scientific papers and textbooks related to the topic of research in education and teaching of informatics in K12 education. Member of the CS research working group.

I. Boljat (Faculty of Science, University of Split, Croatia) received his doctorate in pedagogy – didactics from the Faculty of Pedagogy in Rijeka, Croatia in 1996. Author of numerous scientific papers, Member of ACM SIGCSE. As one of the initiators of the study of Informatics, he participated in the development of curricula for several informatics courses. As a member of the working group for drafting the postgraduate doctoral study program in Education in the natural and technical sciences, he defined the several course curricula. He is the leader of the informatics working group in the project HR.3.1.15-0032.

APPENDIX I

Created CT assessment tool

Task 1

Bee Maya was assigned to find her favorite yellow flower in the presented labyrinth. Maya knows where the flower is situated, but she doesn't know where all obstacles, the gray-colored squares, as well as the edges of the labyrinth were situated.

Help Maya in finding her favorite flower. To get to the flower, Maya will follow all steps listed in the Action Labyrinth. Action Labyrinth also use Action Walk.

Maya will start moving always from the first column and the first row, a shown in the picture below. The direction of view of the Maya Bee does not matter!

Maya always begins her movement through the labyrinth with the Action Labyrinth with the Maze Action, which further uses Action Walk.

Action Labyrinth:
While *not* (flower_up or flower_right) do:
Walk

Action Walk:
If obstacle_up:
Do 1 step right →

else:
Do 1 step up ↑

At what position (row, column) will Maya be after taking 5 steps if she uses Action Labyrinth (and Action Walk within) for movement through the labyrinth.

Task 2

Look at the picture of the labyrinth and notice that it is possible for Maya not to be able to reach her flower. This could happen because of the obstacles in the labyrinth or because of the allowed movement actions. Obstacles in the labyrinth are gray fields or boundaries of the labyrinth. The direction of view of the Maya Bee isn't relevant for her movements!

Maya always begins her movement through the labyrinth with the Action Labyrinth with the Maze Action, which further uses Action Walk.

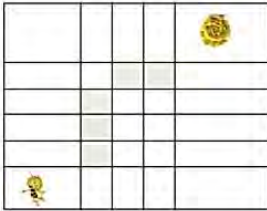
Aktion Labyrinth:
While *not* (flower_up or flower_right) do:
Walk

Action Walk:
If *not* obstacle_up:
Do 1 step right →

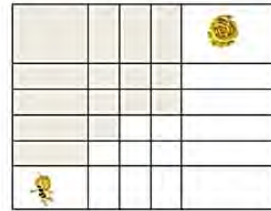
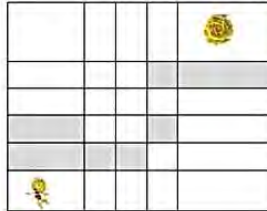
else
Do 1 step up ↑

In which of the following labyrinth examples will Mayan not be able to reach her flower with acceptable movement actions?

a)

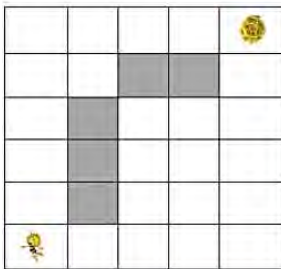


b)



Task 3

The picture below shows one labyrinth and actions which Maya could use while moving through the labyrinth. Maya should always go first up, and if that isn't possible, she should try to go right. However, the instructions given in the Action Walk hide an error since they are not in accordance with the agreed rules. Find a mistake and write the correct actions for moving through the labyrinth.



Aktion Labyrinth:
 While not (flower_up or flower_right) do:
 Walk

Action Walk:
 If not obstacle_up:
 Do 1 step right →
 else
 Do 1 step up ↑



Find and correct the mistake in the action Walk!

Action Walk:
 If not obstacle_up:
 do 1 step right
 else
 do 1 step up

Task 4

We noticed that in some previous examples of labyrinth Maya was stuck before reaching her flower. To avoid the possibility, we should allow her how to move through the labyrinth.

For that reason, let's insert a new command in movement actions in the way that Maya can go left only when she is sure that up and right movements are no longer possible. Let' insert new command

(Do 1 step left ←).

Action Labyrinth:
While *not (flower_up or flower_right)* do:
Walk

Action Walk:
If *not obstacle_up:*
Do 1 step right →

else:
Do 1 step up ↑

				🌻
🐛				

up

↑

← right →

Which of the following actions needs to be changed in order for Maya to move left along the labyrinth, of course, if possible, given the obstacles? Again, the obstacles are the gray squares and the edges of the labyrinth.

- a) Action Labyrinth
- b) Action Walk
- c) There is no need for any changes in the Maya movements.
- d) New action, containing only one command for going 1 step left should we written. There will be no cahnges in the action Labyrinth and Action Walk.

Task 5

To facilitate Maya movements through the labyrinth we tried to make few changes to the Action Walk for the labyrinth on the picture below. With these changes Maya will always be able to take a step in one of the following directions: up, right or left.

Action Labyrinth:
While *not (flower_up ili flower_right)* do:
Walk

Action Walk:
????

				🌻
🐛				

up

↑

← right →

Which of the following versions of the Action Walk, presented below, will enable Maya to successfully navigate through the labyrinht in the way that Maya shouldn't stop because of an obstacle)? A, B or C?

a)

If *obstacle_up:*
Do 1 step right →

else:
Do 1 step up ↑

b)

If *obstacle_up:*
Do 1 step right →

else:
If *obstacle_left:*
Do 1 step right →

else:
Do 1 step up ↑

c)

If *obstacle_up:*
If *obstacle_right:*
Do 1 step left ←

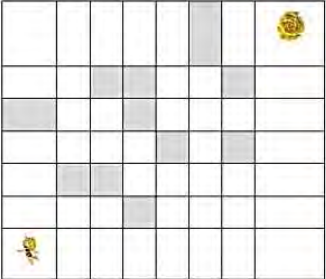
else:
Do 1 step right →

else:
Do 1 step up ↑

Task 6

We accelerated Maya's movement around the labyrinth by modifying the Action Walk so that Maya moves up until she encounters an obstacle and only then turns right or left.

Observe the next labyrinth to see if Maya could reach her flower according the new moving instructions.




Action Labyrinth:
While not (flower_up or flower_right) do:
Walk

Action Walk:
While not obstacle_up:
Do 1 step up ↑

If not obstacle_right:
Idi 1 karak desno →

else:
If not obstacle_left:
Do 1 step left ←



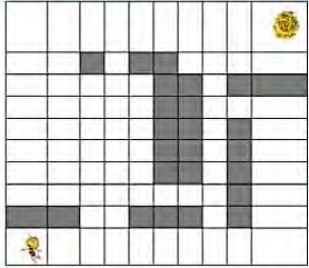
Which of the following statements are true? (There may be multiple correct answers, also each incorrect answer is scored negatively with - 0.5points).

- a) With this algorithm Maya will make at least three steps after she makes her first turn.
- b) With this algorithm, Maya Bee will not be able to reach her flower.
- c) With this algorithm, Maya will turn left at least once.
- d) With this algorithm, Maya will take at least three steps before making left or right turn.

Task 7

Maya decided to keep track of the number of steps she makes on her journey through the labyrinth. She recorded the number of steps in the value named steps_number (variable is positioned above the labyrinth in the picture). At the beginning of each movement through the labyrinth, the value of the steps_number is set to zero (0).

steps_number




Action Labyrinth:
Set steps_number to 0
While not (flowerup or flower_right) do:
Walk

Action Walk:
While not obstacle_up::
Do 1 step up ↑

If not obstacle_right:
Do 1 step right →

else:
If not obstacle_left:
Do 1 step left ←



Upgrade Maya's movement directions in the way that the steps_number value always presents the total number of steps taken.

In the appropriate place in the Action Walk, you need to write your version of the commands which will increase the value of steps_number by 1 each time a step is made.

Action Labyrinth:

Set steps_number to 0

While not flower_up do:

Walk

Action Walk :

While not obstacle_up do:

Do 1 step up

If not obstacle_right:

Do 1 step right

else:

if not obstacle_left:

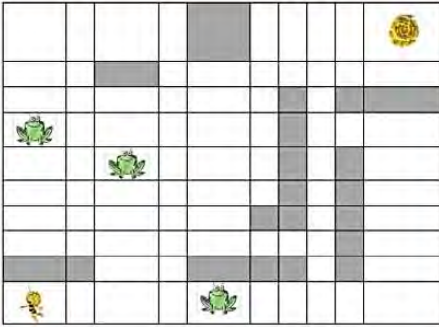
do 1 step left

Task 8

Hazardous frogs have appeared in certain places in the maze seeking to stop Maya Bee on her way towards yellow flower. May **could skip the frog only while moving up**, so we have to make some changes in up movement.

To help Maya we introduce a new rule: **If Maya encounters a frog while moving in an upward direction, she can skip it by taking two steps at a time.**

Steps_number



up
↑

left ← right →

Action Labyrinth:
Set steps_number to 0
While not (flowerup or flower_right) do:
Walk

Action Walk:
While not obstacle_up::
Do 1 step up ↑
Increment value steps_number by 1
If not obstacle_right:
Do 1 step right →
Increment value steps_number by 1
else:
If not obstacle_left:
Do 1 step left ←
Increment value steps_number by 1

Select the image which highlights the set of directions to be modified according to the new labyrinth rules.

a)

Action Walk:

While not obstacle_up::
Do 1 step up ↑
Increment value steps_number by 1
If not obstacle_right:
Do 1 step right →
Increment value steps_number by 1
else:
If not obstacle_left:
Do 1 step left ←
Increment value steps_number by 1

b)

Action Walk:

While not obstacle_up::
Do 1 step up ↑
Increment value steps_number by 1
If not obstacle_right:
Do 1 step right →
Increment value steps_number by 1
else:
If not obstacle_left:
Do 1 step left ←
Increment value steps_number by 1

c)

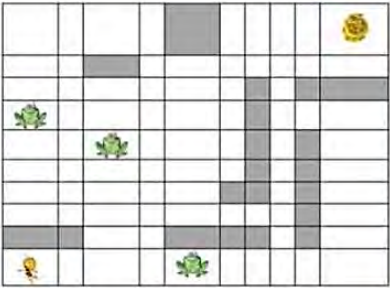
Action Walk:

While not obstacle_up::
Do 1 step up ↑
Increment value steps_number by 1
If not obstacle_right:
Do 1 step right →
Increment value steps_number by 1
else:
If not obstacle_left:
Do 1 step left ←
Increment value steps_number by 1





Task 9

In the previous assignment, we identified the instructions that needed to be changed in order for Maya to successfully navigate through the labyrinth and avoid dangerous frogs as much as possible. And again, Maya can only jump over a frog when going in an upward direction. How does Maya jump over a frog? **If Maya encounters a frog when moving upwards, she can skip it by taking two steps at a time.**

Steps_number:



Action Labyrinth:
 Set `steps_number` to 0
 While *not* (`flower_up` or `flower_right`) do:
 Walk

Action Walk:
 While *not* `obstacle_up`:
 Go_up_and_skip_frog  
 If *not* `obstacle_right`:
 Do 1 step right 
 Increment value `steps_number` by 1
 else:
 If *not* `obstacle_left`:
 Do 1 step left 
 Increment value `steps_number` by 1


Action Go_up_and_skip_frog:
 ?????

Upgrade Maya's movement through the labyrinth so that she can avoid frogs when moving upwards. Write Maya's movement directions in the form of a new action named **Go_up_and_skip_frog**. New action should allow Maya to go 2 steps up if she encounters the frog on her way up, or just one step up if there is





Task 10

Observe the following labyrinth and its movement actions.

Steps_number:



Action Labyrinth:
 Set `steps_number` to 0
 While *not* (`flower_up` or `flower_right`) do:
 Walk

Action Walk:
 While *not* `obstacle_up`:
 Go_up_and_skip_frog  
 If *not* `obstacle_right`:
 Do 1 step right 
 Increment value `steps_number` by 1
 else:
 If *not* `obstacle_left`:
 Do 1 step left 
 Increment value `steps_number` by 1

```
Action Go_up_and_skip_frog:  
  If frog_up::  
    Do 2 steps up ↑  
    Increment value steps_number by 2/  
  else:  
    Do 2 steps up ↑  
    Increment value steps_number by 1 ○
```

How many steps did Maya take before she had to stop (she reached the flower, or she encountered an obstacle she didn't know how to bypass)? Write your answer in the form of a number. {1: SHORTANSWER = 15}