IJPCE
International Journal of Physics and Chemistry Education, *10*(4), 71-84

www.ijpce.org
ISSN: 2589-8876

Algorithm Visualization and its Impact on Self-efficacy, Metacognition and Computational Thinking Concepts Using the Computational Pedagogy Model in STEM Content Epistemology

Sarantos Psycharis*
ASPETE, Greece
spsycharis@gmail.com

Dimitris Mastorodimos
Greek Ministry of Education Research and Religion, Greece
mastorodimos@gmail.com

Konstantinos Kalovrektis
University of Thessaly, Greece
kkalovr@gmail.com

Panagiotis Papazoglou
Technological Educational Institute of Central Greece, Greece
papaz@teiste.gr

Lampros Stergioulas
University of Surrey, UK
l.stergioulas@SURREY.ac.uk

Munir Abbasi
University of Surrey, UK
m.abbasi@surrey.ac.uk

Abstract

The objective of this article is twofold. One objective is the development of models of visualized algorithms (VAs) for three fundamental algorithms, the bubble sort algorithm, the selection sort algorithm and the insertion sort algorithm, using the Easy Java simulations software (Ejs) and the Computational Pedagogy model. The second objective is to investigate: a) VAs impact on learners' self-efficacy as a general structure, metacognitive experience, critical thinking and motives and b) VAs impact on learners' self-efficacy relative to Computational Thinking. An intervention in the form of a didactic model was implemented that utilized VAs and the Computational Pedagogy approach. Finally, we argument how VAs can be embedded in the Computational STEM pedagogy approach in teaching and learning sequences through applications related to authentic problems.

State of the literature
- Visualization of algorithms is mainly related to viewing
- Visualization of algorithms is a rather static process
- There is no specific pedagogical model that implements a dynamic transformation, so students will be engaged in the development of visualization

Contribution of this paper to the literature
- Visualization of algorithms is related to Computational Thinking concepts
- Visualization of algorithms is implemented through the computational experiment approach
- Easy Java simulation is used in order to help students to build their visualizations

Keywords

Visualization, Algorithms, self-efficacy, metacognition, Computational Thinking, STEM, Computational Pedagogy

◆Received 10 September 2018◆ Revised 18 November 2018◆ Accepted 22 November 2018

Introduction

Visualisation of Algorithms (VA)

Recently, there is an increased interest in computational thinking (CT) and the ways it can be taught in a teaching and learning sequence and visualization is suggested as one way of supporting learners' learning (Cetin & Andrews-Larson, 2016). Computational thinking includes the abstraction and computer algorithms are also abstract entities, so an effective teaching and learning approach to teach CT in parallel with algorithmic thinking is a challenging task (Katai, 2014).

Computer Science (CS) Educators have used different approaches such as the use of Algorithm Visualization (VA or AV) and Program Visualization (PV) to convey more efficient pedagogical approaches for learners' algorithmic learning. VA has a very long history in CS education, dating from the 1981 video ``Sorting out Sorting'' (Baecker, 1998) and since then many VAs have been presented in education research. The aim of all these VAs was to represent the different steps of the algorithms by animating the transitions between algorithmic steps.

Research also suggests that a useful teaching method for how algorithms work is to visualize their function at specific steps and observe their state at particular time events. According to (Shaffer et al., 2007; Shaffer et al., 2010; Shaffer et al., 2011) while many good algorithm visualizations are available, the need for more and higher quality visualizations continues.

Current visualisation systems and visual analytics systems focus mainly on developing new visualisations based on specific domain knowledge. They often do not provide users with the

* Corresponding author
https://doi.org/10.12973/ijpce/12296

flexibility to replace or to experiment with various computational methods. Such visualisations are generally adapted to only one application or one specific dataset. For example, Andrienko and colleagues (2013) devised a comprehensive visual analytics tool for analysing the collective movement of a group of individuals. In their system, they make use of a group's "centre" and presume that the "centre" can be computed at equal time stamps for each trajectory without allowing variability in time and space. Consequently, a strong need exists to make visual analytics systems more flexible to support a wide range of computational methods.

Algorithm visualization is also considered as a support active learning, especially in the cases where learners are really engaged in the process of the algorithm and they can handle/control it as an active learning experience (Shaffer et al., 2010), and not as a passive one(for example watching the algorithm in a video). Active participation in the structure of algorithms is also discussed in many papers, indicating the importance of learners' intervention (e.g. Naps et al., 2002).

According to (Katai, 2014) the purpose of algorithm visualizations is also to promote students' understanding regarding the procedural behaviour of algorithms.

According to (Staley, 2016) multimedia digital fails to consider the information from all of our senses and we need an active visualization not in a form of a video but in the form that enhances learners 'engagement in the structure of the algorithm.

Succinctly, through the instruction, we need more dynamic processes during the working of an algorithm and such dynamic processes are not well supported by static media such as text and images or even video, and consequently more active methods are needed to be to conveyed in the teaching and learning sequences.

"The first significant attempt to assess the pedagogical effects of algorithm visualization was conducted by Hundhausen et al. (2002). They performed a meta-study of prior small-scale experiments related to AV. They found that 11 of those experiments showed a "statistically significant difference between the performance of a group of students using AV technology and another group of students using an alternative AV technology or no AV technology at all" (Fouh et al.,2012)

In every study about the impact of the AV we should distinguish the impact on conceptual or declarative knowledge (understanding of the concepts of an algorithm) and the impact on procedural knowledge (internalization of the step-by-step behaviour and data manipulation of an algorithm) (Fouh et al., 2012)

According to Hundhausen et al. (2002), AV technology is more effective when it is used in order to engage learners in the algorithmic process actively. This view has had an enormous impact on subsequent AV development and this result had a strong impact on subsequent efforts to visualize algorithms for educational use.

Naps, et al. (2002) extended the conclusions of (Hundhausen, et al.,2002) about active learners' engagement through AVs and they proposed a learners' engagement taxonomy that specifies the level and the type of engagement at which AVs can involve learners (Fouh et al., 2012). They defined six engagement categories.

- "No Viewing" indicates cases where no AV is used.
- "Viewing" involves both watching the AV execution and directing the pace of the AV.
- Responding" involves answering questions about the visualization being viewed.
- "Changing" asks the learner to provide input data to the AV to direct its actions.
- When "constructing", students build their own visualization of the algorithm.
- "Presenting" asks the learners to tell about the AV and gather feedback" (Naps et. al., 2012)".

Research results indicate (Shaffer et al., 2010) that most AVs are still at the Viewing level while this taxonomy can be strongly connected with stages of Bloom's taxonomy and can serve as an indicator for learners' engagement in Bloom's taxonomy.

Another critical issue raised is that beyond the visual and audio channel, it is very essential -during the teaching of algorithms- to consider the dimension of time as well, so learners can be engaged in the study of the structure of the algorithm in real time (Torley, 2014).

Computational Thinking (CT)

Wing (2006;2008) suggested that computational thinking concept involves solving problems, designing systems, and understanding human behaviour, by portraying on the concepts fundamental to computer science (Wing, 2008). The core CT concepts include, abstractions (the mental tools of computing, necessary to solve the problem), layers (problems need to be solved in different levels) and relationships between layers and abstractions.

According to some researchers, the idea of abstraction and students' ability to work and apply abstractions in different phenomena and to implement algorithms related to abstraction is fundamental to CT" (Lu & Fletscher, 2009).

Although CT it emerged in academia over forty years ago, there is much room to invent new mathematical and computational ways to make algorithms relevant and useful in this area. Computational Thinking is simultaneously an ambition inspired by notions of Artificial Intelligence (AI).

Brennan & Resnick (2012) proposed a computational thinking framework consisting of three dimensions: computational concepts, computational practices, and computational perspectives.

They defined computational concepts as the "concepts that designers employ as they program" and include: algorithmic thinking, abstraction, problem decomposition, data, parallelization, and control flow. They also argued that CT practices focus on the process of thinking and learning, moving beyond what you are learning to how you are learning, and include: being incremental and iterative, testing and debugging, and reusing and remixing.

Finally, authors defined "Computational perspectives are the way students understand the effect of computation and technology in the world around them, as well as with their relation to others and themselves" and they suggested that they include: expressing, connecting and questioning.

We easily conclude that CT is related to algorithms, mainly through the CT concepts (think for example the "control flow" concept that directs an algorithm's steps and when an algorithm is completed, allowing the algorithm to repeat steps several times under certain conditions).

Algorithmic Thinking

An algorithm can be described as a set of directions, a procedure that may include random elements such as coin flipping, but does not require judgment calls (NCTM, 1989). Futschek (2006) defines an algorithm as a method for problem solving that includes clear instructions, and algorithmic thinking is a set of skills and abilities that are connected to constructing and understanding algorithms in order to analyze and determine specific problems and the ability to find the basic actions that are adequate to the given problem.

An algorithm can be viewed in many ways, for example, control flow in source code or state of data structures. Providing the learner with multiple views can facilitate a better understanding of the algorithm. In particular, it is beneficial to provide a program animation view (where a code is shown and highlighted as the program executes) simultaneously with more abstract algorithm animation views.

According to Torley (2014) one of the strengths of AVs is the involvement of more sensory channels in learning. Kátai (2014) suggested that if a teaching method impacts on different sensory channels, then it can effectively support the teaching and learning of algorithms.

Aho (2012) suggested that CT involves formulating problems so their solutions can be represented as computational steps and algorithms. Denning (2003) also argued CT "has a long history in computer science dating back to 1950s when it was known as algorithmic thinking meaning mental orientation to formulate problems as conversions of some input to an output and looking for algorithms to perform the conversions". In conclusion, "not all definitions of CT are created equal; however, among various definitions abstraction and algorithms are the two main concepts that everyone agrees upon".

Self-efficacy

Self-efficacy can be defined as "an individual's belief that she/he can accomplish a particular task" and is related to the perceived ability rather than the actual ability.Self-efficacy is a construct developed to describe the impact of a person's belief in her/his ability to complete a given task (Bandura, 1997) and he stated that "different people with similar abilities or the same person under different circumstances may perform poorly, adequately, or extraordinarily depending on fluctuations in their beliefs of personal efficacy" (Bandura, 1997).According to Bandura (1997), "Self-efficacy is context-dependent; a person can have a high self-efficacy for a given task in one context (such as a study in a group meeting) and a low self-efficacy for the same task in a different context (like a classroom exam in science)".

Among the factors that are considered that affect the learning process is the self-efficacy perception (Erdogan et al., 2008; Chemes et al., 2001). Self-efficacy has been also suggested as an accurate measure for computational thinking (Bell, 2014; Bean, Weese, Feldhausen, & Bell, 2015) and computer programming (Ramalingam, LaBelle & Wiedenbeck, 2004).There is also an increased interest for the connection of self-efficacy not only with the CT concepts but also with the STEM epistemology. According to (Shaw et al., 2012), " There is a reason to believe that students' self-efficacy beliefs regarding STEM courses are a factor in determining student performance in these courses".

Metacognitive awareness

Metacognition is a concept that is often considered have an impact on the improvement of students' learning outcomes. According to (Thomas & McRobbie, 2001) metacognition can enhance students' thinking and learning processes.

Metacognition is often considered as a fuzzy concept, and has been defined as, for example, knowledge, control and awareness of learning processes (Baird, 1990; Thomas & McRobbie, 2001) and the ability to think about one's thinking (Gilbert, 2005), with Blank (2000) positing additionally that students' consideration of the status of their science ideas also constitutes an aspect of metacognition.

According to (Psycharis, et al., 2014) the Computational Experiment (CE) methodology, described below as a method to apply modelling and CT in the inquiry based teaching and learning approach using computational processes, is a key challenge for the development of metacognitive experiences , as students are stimulated to develop modelling skills, they are engaged in inquiry tasks and also develop strategies to solve particular problems in order to enhance their metacognitive awareness and evaluate their efficacy of strategies.

"Metacognition is associated with planning, monitoring, evaluating and repairing performance, while metacognitive strategies guide students to think before, during, and after a problem solution. It begins by guiding students to plan for selecting the appropriate strategy to accomplish the task, and then continues as they select the most effective strategy and finally students evaluate their learning process and outcomes"(Psycharis et al., 2014).

Computational Pedagogy

According to (NGSS, 2013) "Computational thinking, modeling and programming are now regarded as core epistemic and representational practices in K12 science and engineering". Computational thinking is also intertwined with representational practices such as the development of models of simulations and programming, which leads to the design and development of computational abstractions such as defining patterns (Wing, 2011; Sengupta et al., 2013).

While there continues to be discussion about the definition of computational thinking, there are significant efforts to embed computational thinking in the context of science and mathematics courses. Weintrop et al. (2016) suggested an insightful taxonomy of computational thinking in mathematics and science courses, including a productive elucidation of model building, using programming. The taxonomy divides computational thinking into four types of practices: data, modeling and simulation, computational problem-solving, and systems thinking and we argue that this taxonomy is related to Computational Experiment (CE) approach and the Computational Pedagogy.

In this article we wanted to measure students' metacognitive experiences and self-efficacy for understanding the concepts in three algorithms as well as students' self-efficacy for CT concepts, when VAs are used. For metacognition and self-efficacy(as general structures) we used the SEMLI-S instrument developed for exploring science students' self-perceptions of elements of their metacognition, self-efficacy and science learning processes, modified to STEM learning.

As an effort to further understand the role of self-efficacy in CT concepts, which are related to algorithms as described in the introduction, this study also focused on learners' self-efficacy beliefs about CT concepts when students were engaged in a teaching sequence that used VAs. To investigate this relation, the instrument developed by (Weese, Feldhausen, & Bean, 2016) was used.

Our intervention, for both groups, was based on the Computational Pedagogy model (Psycharis, 2018a). Computational Pedagogy is based on the Computational Science, which is the integration of Mathematics, Computer Science and any other discipline to explore authentic-complex problems and is considered to have its own core knowledge area subjects (e.g. Landau et al., 2008). Juszczak (2015) states that Computational Science, in both natural and social sciences, "is different than the usage of computers to analyze complex systems and data sets. Computational Science is a non-empirical science. Data that is gathered in Computational Science is the result of simulations and virtual experiments". Using Computational Science in Education (CSE), "we accept that we can conduct experiments and iterations that could never be conducted in the real world with results equivalent to the classical experiments" (Psycharis, 2018b).

Computational Science is connected with the STEM epistemology and CT. According to (Psycharis, 2018a), Computational Experiment (CE)-which is grounded on CSE- can be an effective methodology to support learners to solve a STEM problem using computer simulations and this includes diverse tasks, such as: formulating the problem in a way suitable for simulations using models , choosing an efficient computational algorithm , running the simulations and collecting numerical data, analyzing the data obtained , finding patterns in order to generalize the method to other problems ,extracting the solution of the problem in a form that can lead to the creation of artifacts, while all these "components" of CSE are strongly related to CT and with the engineering education epistemology.

According to (Psycharis, 2018b), "we consider modelling as a central issue in the CSE methodology while the CE experiment implements CT in practice in accordance to (Bienkowski et al., 2015), where it is clearly stated that "computational science tends to emphasize data, modeling, and systems thinking". From what mentioned above it is quite evident that CSE can serve as a theoretical framework to implement Engineering Epistemology and CT as a pedagogical framework.

In our approach for the Computational Pedagogy we consider the CSE as the theoretical framework and the CE experiment as the space where computational practices-as they were proposed by (Brennan & Resnick, 2012)- can be applied and students start with the model and follow a process and select an evidence based on data.

Yasar et al. (2016 introduced) the term Computational Pedagogy as an extension of TPACK, and it was named Computational Pedagogical Content Knowledge (CPACK). Psycharis (2018b) extended the model of Yasar et al. (2016) by adding the computational spaces practices related to the engineering design (**Figure 1**).

In our approach, the model is the basic instructional unit, it is used in the inductive process of teaching and is closely connected to abstraction (Yasar, 2013). Psycharis (2016a, 2016b) discussed the spaces of the computational experiment and proposed inquiry based activities at each space. To use the model and simulation in the inductive process of teaching, we need proper environments that support the use of mathematics and algorithms, so the computational experiment will be "equivalent" to the physical experiment.

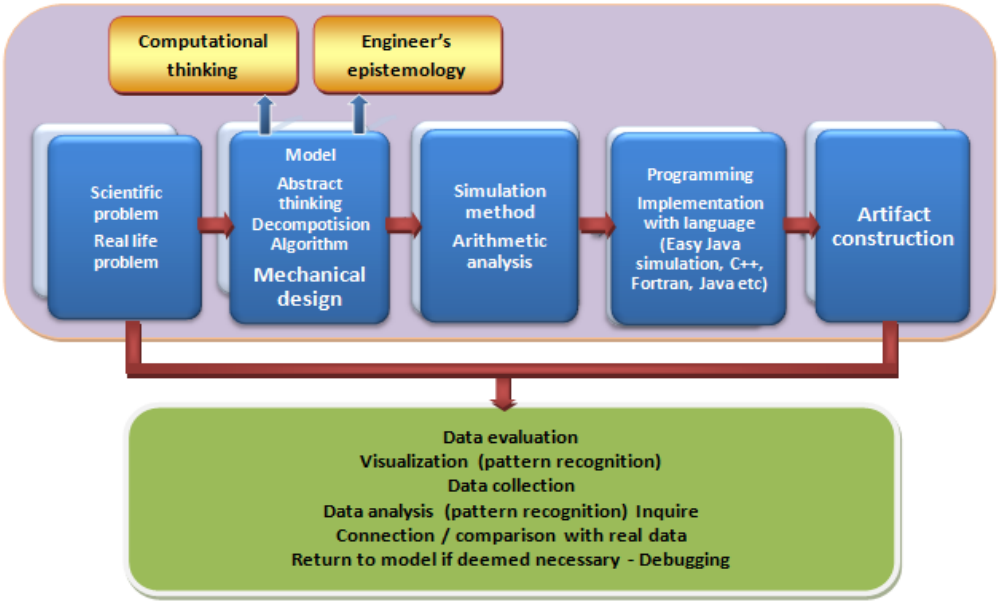

Figure 1. The Computational Science Experiment (CE experiment) with engineering design and CT-Computational Pedagogy Model

This study aims at investigating the role of the Computational Pedagogy model integrated with a) text based programming (for the control group) and b) with VAs (for the experimental group) on self-efficacy and metacognition and the self-efficacy for CT. Therefore, the research questions are as follows:

1. Is there any difference of the impact of the Computational Pedagogy model integrated with text based programming and the Computational Pedagogy model integrated with VAs on students' self-efficacy and metacognition?
2. Is there any difference of the impact of the Computational Pedagogy model integrated with text based programming and the Computational Pedagogy model integrated with VAs on students' self-efficacy for CT?

Methodology

Participants

The study was conducted with sixty (N=52) pre-service engineering education school teachers from a Greek Higher Education Institute who participated voluntarily in the research during the course 'Pedagogical applications of Computers". Students were enrolled in their second year of their undergraduate studies and they had already attended a course in C programming.

Students study for five (5) years in order to take their degree as engineering educators. The course includes the development of algorithms and models of simulation in alignment with contemporary pedagogical theories and students have to develop their own pedagogical scenario using the inquiry teaching and learning approach. After graduation, students are expected to teach engineering courses and computing-related courses at secondary school education.

Materials

All students participated in the thirteen periods (three hours each) of the course. Initially, students were exposed –for six hours-to models in Physics and Mathematics from the repository (www.opensourcephysics.org). Models of simulations, included in this repository, are developed using the easy java simulator tool (Ejs) (http://www.um.es/fem/EjsWiki/Main/Download), and students discussed with the Instructor(one of the authors) the technical details of the software and the possibility to develop their own models using the "Model" and "View" components of the software. The topics were selected from the thematic areas of Mechanics and Electromagnetism.

Easy Java Simulator, also known as Ejs (http://www.um.es/fem/EjsWiki/), is a free authoring tool written in Java that helps non-programmers to create interactive simulations in Java, mainly for teaching or learning purposes.

Easy Java Simulations is a software tool (java code generator) designed for the creation of discrete computer simulations where the learner can write down an algorithm in a specific part of the interface in a way that resembles the way he writes using text based language.

For the first three weeks, students were engaged in the Ejs environment using algorithms related to certain phenomena, i.e. nuclear decay, RC circuit, etc. , in order to familiarize themselves with the process to write down an algorithm relevant to STEM disciplines. Most of the algorithms used were contained in the repository (https://www.compadre.org/osp/?).

In order to include the JavaScript, Ejswas extended to EjsS, which was used in the present study, as it provides the option for the creation of Web Pages and the development of models for mobile devices, using the EjsSreader (**Figure 2**).

EjsS is considered as a very powerful tool for development of models of simulations, even for users without good experience in programming (Chartier & Kreutzer, 2010), since its interface is easy to be learned and contains a lot of tools to build algorithms and write a source code (see **Figure 3**).

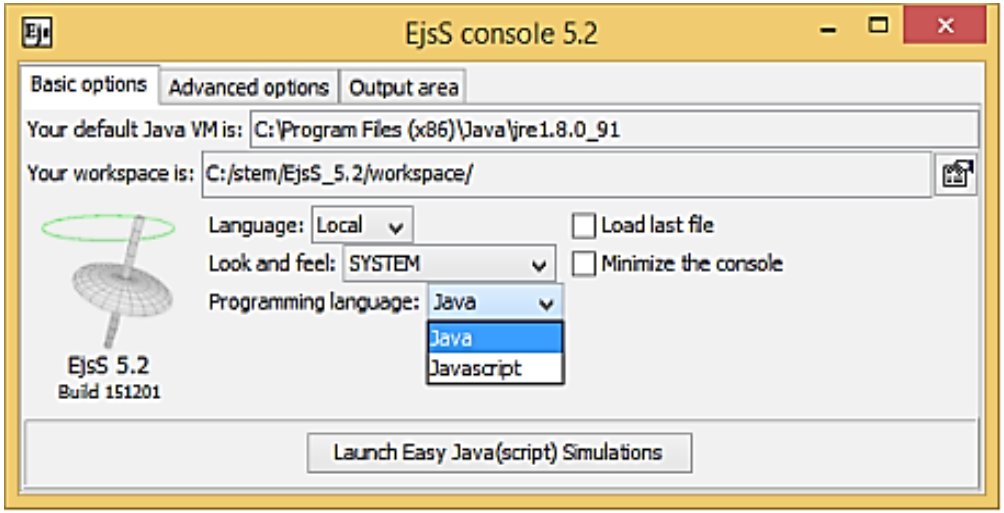

Figure 2. The installation of EjsS

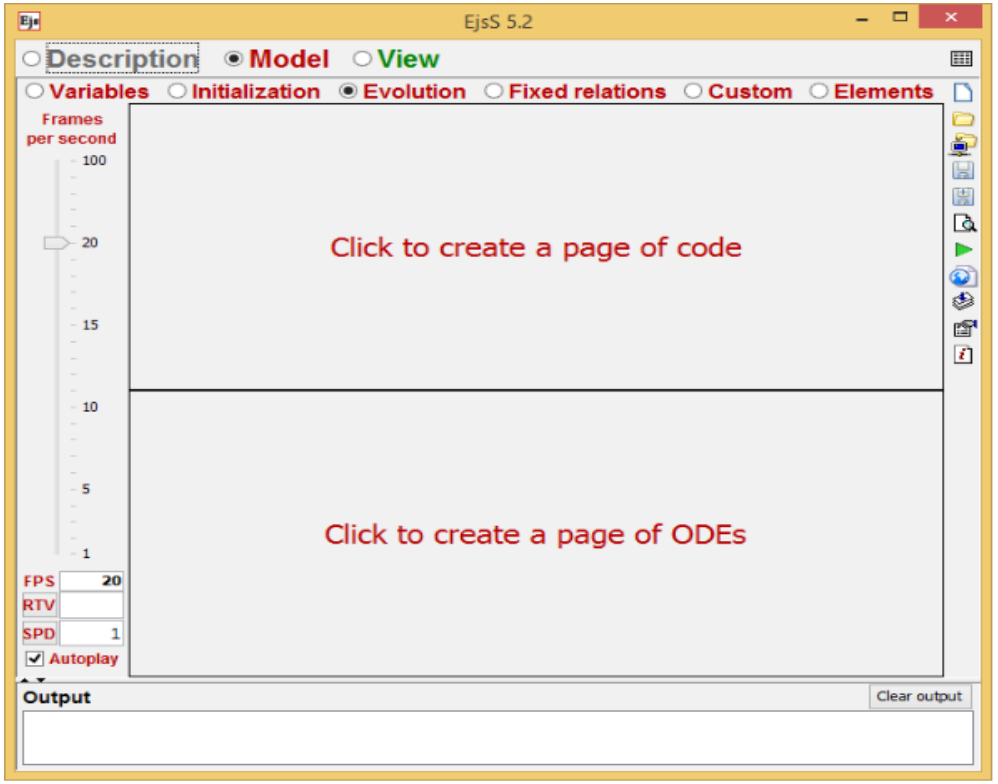

Figure 3. The working environment of EjsS for creation of code

At the model component, the part "Elements" provides the possibility of communication of the model developed in EjsS with third party libraries and platforms, like accelerometer, Arduino, Labview etc. (**Figure 4**).

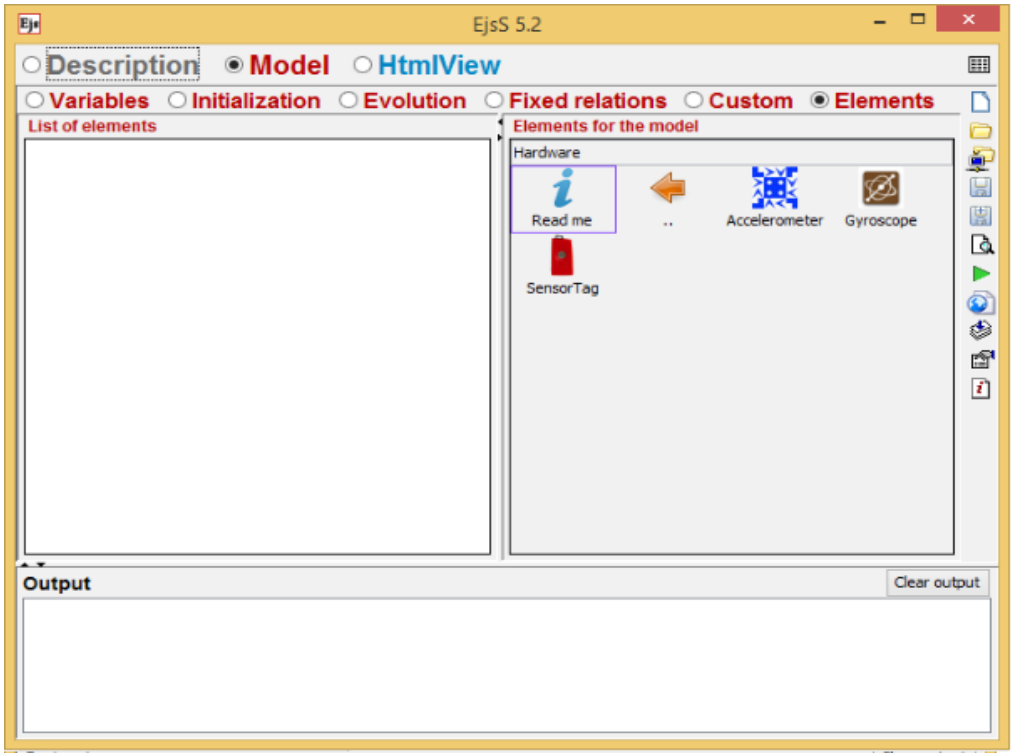

Figure 4. Connection of EjsS with other platforms and sensors

The View component of EjsS provides the user with the option to introduce two dimensional and three dimensional objects in the model of simulation. In addition, the user can add motion and sound to the objects of simulation, so she/he can have a kind of visualization of the model and the algorithm.

Procedure design and measures

The main purpose of this research study was to examine the impact of teaching algorithms using the Computational Pedagogy model integrated with VAs on learners' self-efficacy and metacognition and their self-efficacy for CT. In order to investigate this relationship, the type of the research design adopted was the quasi-experimental design and in particular, the non-equivalent control group pre-test/post-test design. In non-equivalent group designs, different groups receive different treatments and the effectiveness of a treatment is evaluated by comparing the performances of the two groups (Millsap & Maydeu-Olivares, 2009). This design requires a pre-test to have an indication of how similar the control and the experimental group were before the intervention, as well as a post-test (Fife-Schaw, 2000).

The instructor informed students about the study and students were divided between the control group (N=25) and the experimental group (N=27).Students of the control group were exposed to the Computational Pedagogy by following the steps described in **Figure 1**, but without the use of VAs. For example, students constructed the algorithm using the pencil and paper, starting from the abstraction process, they decompose the problem in smaller ones, and they created the code using text based programming. In case they needed help, the Instructor guided them in specific repositories in order to find parts of the code.

On the contrary, the experimental group followed the Computational Pedagogy model but instead of text based programming, the model interface of EjsSwas used for visualization of algorithms. During the instruction, students developed the code working with the Instructor.

Before the instruction of the algorithms, all students filled the two questionnaires, one for the self-efficacy and metacognition- SEMLI-S(Thomas et al. ,2008) and one for self-efficacy for CT concepts (Weese et al.,2016). SEMLI-S questionnaire contains 30 questions and self-efficacy questionnaire contains 23 questions relevant to self-efficacy in CT concepts (Weese et al., 2016).From SEMLI-S questionnaire we did not used the questions for Learning Risks Awareness and for Control of Concentration, because we considered that they might also be considered to relate to monitoring and evaluation of learning, as was also suggested by (Thomas et al., 2008).

The questions of SEMLI-S are distributed as: 7 questions for Constructivist Connectivity (CC)- items that explore students' perceptions of whether they construct connections between information and knowledge across various science learning locations, 9 questions for Monitoring, Evaluation & Planning (MEP)- items that might be seen as traditionally related to metacognition and they reflect important strategies for the learning of STEM-, and 6 questions for STEM Learning Self-efficacy (SE)- items that explore students' perceptions of their orientation to organize and execute actions that are needed to attain STEM learning goals.

The resulting 22-item initial instrument utilized a five-point Likert scale (1 = Never or almost never; 2 = Sometimes; 3 = About half the time; 4 = Frequently; 5 = Always or almost always).

The questionnaire for self-efficacy in CT concepts (Weese et al. , 2016) contains 23 questions and each of these questions measured self-efficacy on a five-value Likert scale: strongly agree, somewhat disagree, not sure, somewhat agree and strongly agree.

Control group students were taught with the traditional method and viewed pre-made visualizations. Instructor used the traditional method to explain the different stages of the three algorithms, the nested loops, the variables and their changes as well as the various steps of execution.

The during-class period for the experimental group consisted of two sections, explanation and applications using the models developed in EjsS. During the explanation, students were informed about the nature of the three algorithms. Next, students were engaged in the development of the three algorithms alongside with the Instructor. Students in the experimental group were given instruction that required in order to construct visualizations related to algorithms in the environment of EjsS.

The intervention for both groups lasted for 6 hours. Learners' scores were compared before and after the programming course. In particular, the pre-test scores learners' self-efficacy and metacognition and self-efficacy for CT were compared with the corresponding post-test scores. A paired t- test could have been employed to compare the means of the same group in the pre-test and post-test and the independent measures t-test could have been used to assess the significance of the difference between the means of the two groups. Since the data were not normally distributed (value of significance using the Shapiro-Wilk was found 0.036<0.05), the Wilcoxon signed ranked test (for the same group) and the Mann–Whitney U test (for independent groups) were used instead. The Wilcoxon signed ranked test is a non-parametric test and was used to test the difference between the median calculated more than once for the same group while the Mann–Whitney U test was used to test if the data collected from two independent groups differ significantly.

The applications

Below, the algorithms are presented in the environment of EjsS. The bubble sort algorithm was developed using the button "Model" as shown in **Figure 5**. The reader can find the whole model at the address http://portal.opendiscoveryspace.eu/en/edu-object/algorithmos-taxinomisis-fysalidas-me-ti-hrisi-ejss-848384

During the execution of the simulation (**Figure 6**), learners can intervene in order to change the number of items, to stop the simulation, to change the steps, and to execute the simulation step by step, so that they can have more direct communication with the algorithm. They were involved in the Computational experiment methodology, through the hypothesis space (selection of variables and relationship between them), the experiment space (development of simulation) and the prediction space (debugging end extension to similar situations raising their metacognitive awareness).

Figure 5. The model of the bubble sort algorithm in EjsS

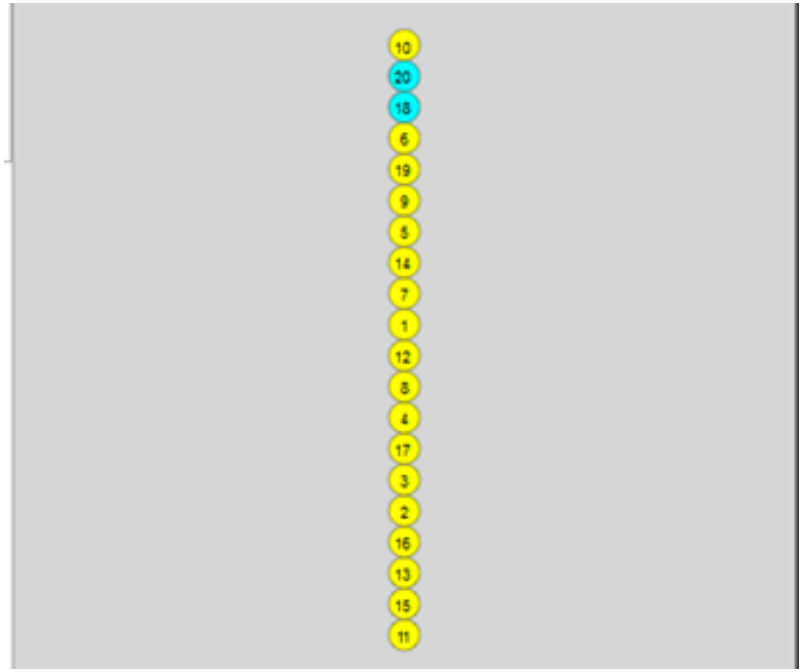

Figure 6. The execution of the bubble sort algorithm

Below we present the models for the other two algorithms. We present the model for the selection sort algorithm in **Figure 7**.

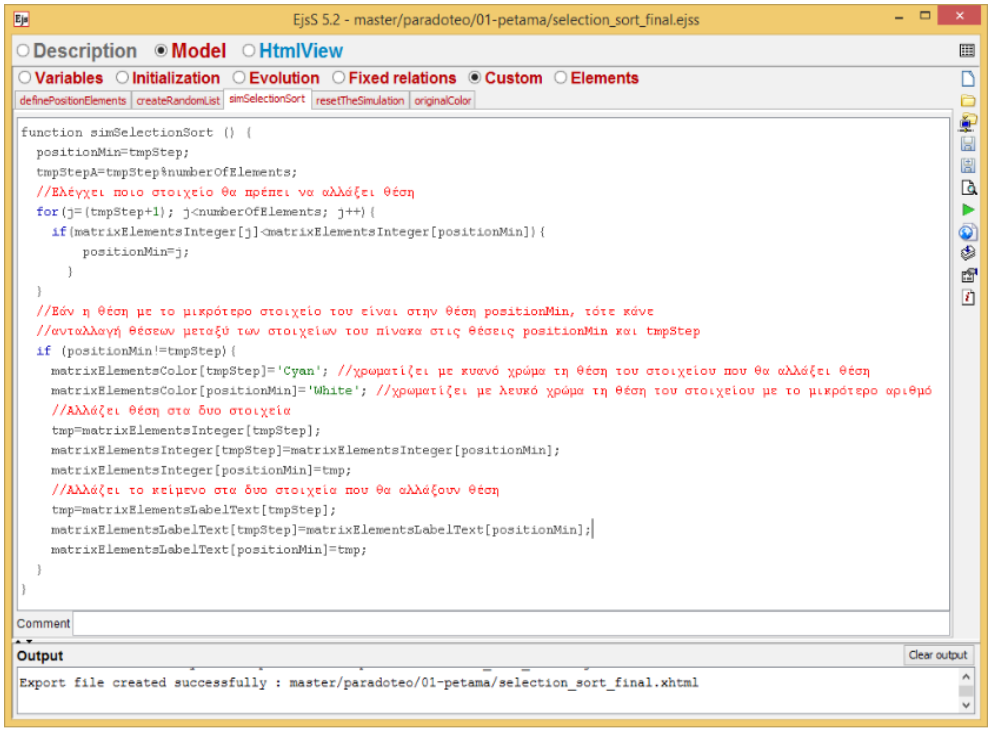

Figure 7. The model of the selection sort algorithm in EjsS

Figure 8 shows the model for the insertion sort algorithm.

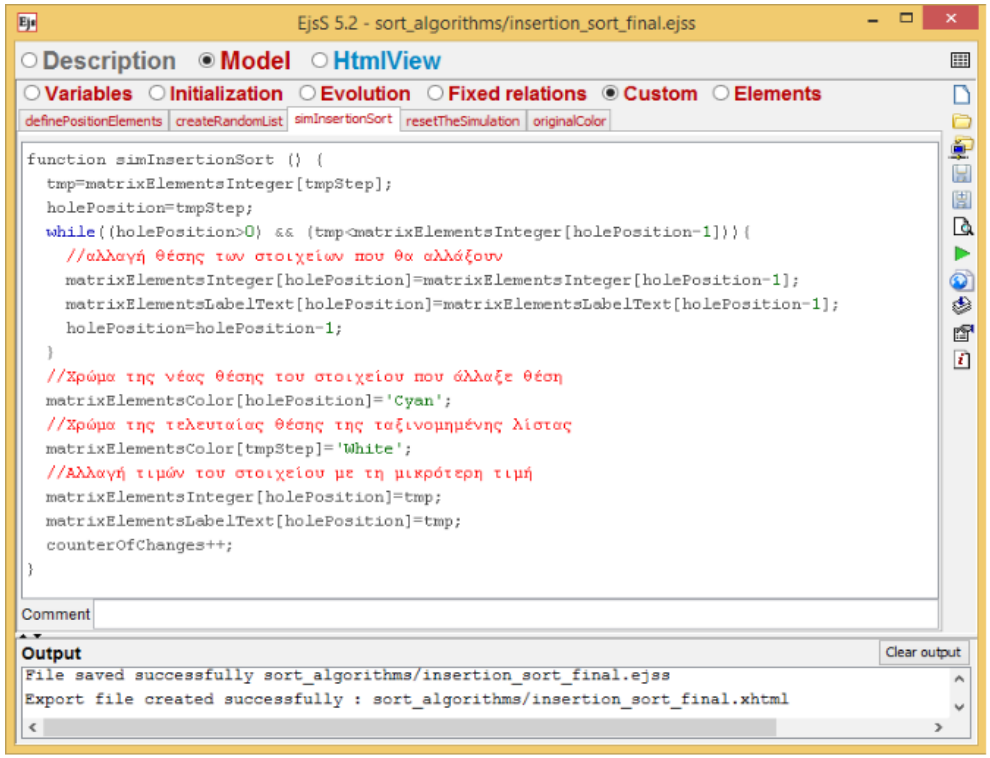

Figure 8. The model of the insertion sort algorithm in EjsS

Below (**Figure 9**) we present the variables used for one of the algorithms, the bubble sort

EjsS 5.2 - master/paradoteo/01-petama/selection_sort_final.ejs

numberOfElements	2	int	
tableElementsPosition	0.0	double	[24][2]
tableElementsLabelText		String	[24]
tableElementsLabelVis...		boolean	[24]
tableElementsInteger		int	[24]
tableElementsColor		String	[24]
i	0	int	
j	0	int	
tmp	0	int	
exist	false	boolean	
tmpStep	0	int	
counterOfCompares	0	int	

Figure 9. The variables of the bubble sort algorithm

Application of Visualization in STEM disciplines

In order to establish a connection of the algorithms with authentic processes and include the dimensions of CT, we decided to develop learning sequences based on applications from real problems. One such application was about the sorting of data from a computational experiment were the software Labview (http://www.ni.com/en-us/shop/labview.html) and the microcontroller Arduino (https://www.arduino.cc/) were used. Students had to design an artifact in order to measure the period of a phenomenon and during this they had to create an algorithm visualized in Labview (**Figure 10**). During the experiment, students realized that the data received in real time had to be classified using an algorithm they were taught during the class and be analyzed in order to find patterns. Students followed the phases as they are described in **Figure 1**. They were also engaged in the engineering design by creating

Results

In this section, we turn to our research questions by examining students' scores on the self-efficacy and metacognition and the self-efficacy for CT.

Effect on learners' self-efficacy and metacognition

Throughout the work the level of statistical significance is set to p=0.05. To ensure that learners' pretest results had not any significant difference between the two groups for the self-efficacy and metacognition using SEMLI-S, the Mann–Whitney U test was applied. The results showed no statistically significant difference between the control and the experimental group pre-test scores (p=0.7).

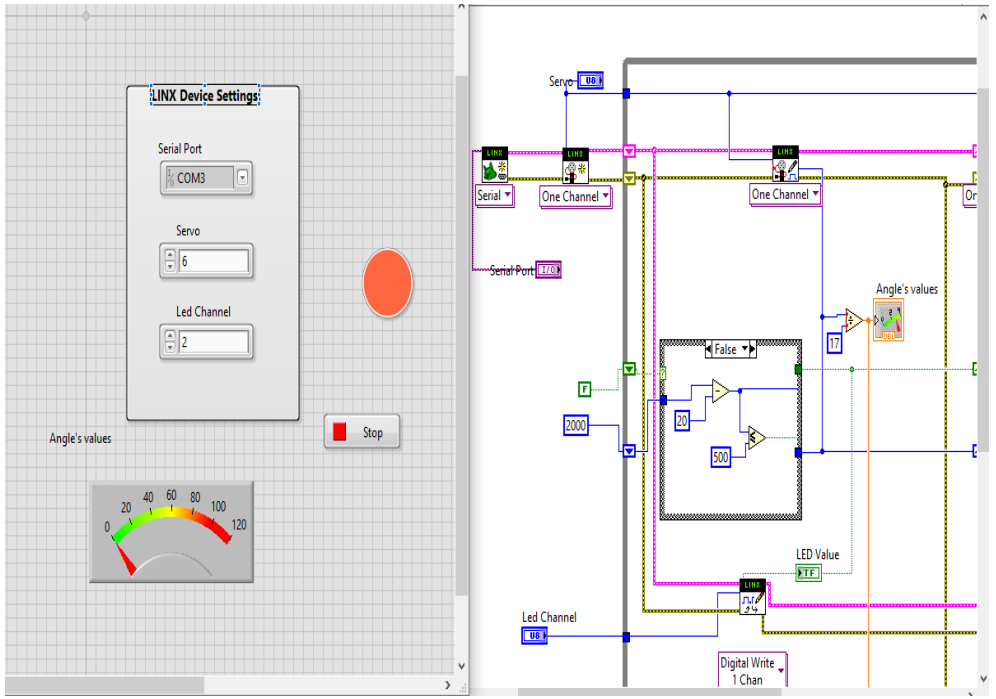

Figure 10. The use of the Computational Pedagogy model and sorting of data with Labview and Arduino

To examine whether the difference between the pre-test and the post-test in the same group was significant, we used the Wilcoxon signed rank test. The null hypothesis that was tested is: "There is no significant difference between the pre-test score and the post-test score for the control group". Since the p value (p=0.073) is greater than 0.05, the null hypothesis cannot be rejected. Thus, the difference between the mean score in the pre-test and in the post-test for the control group is not significant.

Results show that the Computational Pedagogy model integrated with text based programming had not any impact on learners' self-efficacy and metacognition. For the experimental group, the null hypothesis that was tested is: "There is no significant difference between the pre-test score and the posttest score for the experimental group". The p value is less than 0.05 (p=0.014) and thus the null hypothesis is rejected which means that the difference between the pre-test score and the post-test score is significant. Specifically, the Wilcoxon signed-rank test showed that the instruction of algorithms in parallel with visualization using the Computational Pedagogy model, did elicit a statistically significant change in the self-efficacy of the students.

To test whether the two groups differ significantly in the post-test, the independent t test could not be employed since the data of the groups were not normally distributed. Shapiro-Wilk test for normality was used and the value of significance was 0.04 (< 0.05), meaning that the data is not coming from a normal distribution.

For this reason, the non-parametric Mann–Whitney U test was used. Specifically, the null hypothesis that was tested is: "There is no significant difference between the scores of the experimental group and the control group in the post-test". Considering that the p value is equal to 0.023, the null hypothesis is rejected. Specifically, the Mann–Whitney U test showed that the instruction of VAs in parallel with the Computational Pedagogy model did elicit a statistically significant difference in the self-efficacy and metacognition of the students in the experimental group from students in the control group.

Effect on learners' self-efficacy relative to CT

To ensure that learners' pretest results had not any significant difference between the two groups for the self-efficacy for CT, the Mann–Whitney U test was applied. The results did not show statistically significant difference between the control and the experimental group pre-test scores (p=0.9). To examine whether the difference between the pre-test and the post-test in the same group was significant, we used the Wilcoxon signed rank test.

The tested null hypothesis was: H_0: "There is no significant difference between the pre-test score and the post-test score for the control group". Since the *p* value (0.01) is less than the value (0.05), the null hypothesis is rejected. Thus, the difference between the mean score in the pre-test and in the post-test for the control group is significant. For the experimental group, the null hypothesis that was tested is: "There is no significant difference between the pre-test score and the post-test score for the experimental group". The *p* value equals zero (0.000) and thus the null hypothesis is rejected which means that the difference between the pre-test score and the post-test score is significant.

To test whether the two groups differ significantly in the post-test, the independent t-test could not be employed since the data of the groups were not normally distributed. For this reason, the non-parametric Mann–Whitney U test was used. Specifically, the tested null hypothesis was: H_0: "There is no significant difference between the scores of the experimental group and the control group in the post-test". Considering the *p* value (0.04), the null hypothesis is not accepted. Specifically, the Mann–Whitney U test showed that the instruction of VAs in parallel with the Computational Pedagogy model did elicit a statistically significant difference in the self-efficacy related to CT of the students in the experimental group from students in the control group, while the experimental group had higher scores.

Discussion

The results showed that the intervention based on visualization construction of algorithms, combined with the Computational Pedagogy model, resulted in significantly better results for the SEMLI-S questionnaire items for the experimental group (Cronbach's Alpha reliability coefficient was found to be 0.966).Instead, implementing the Computational Pedagogy model with text based programming seems to have no impact on learners' self-efficacy.

However, it is worthwhile to notice that for the six (6) questions of SEMLI-S that are related directly to the STEM efficacy, there is no statistical significant difference between the control and the experimental group, which indicates that Computational Pedagogy model (which applies to both groups) has a substantial impact for the STEM efficacy for both groups. For these questions, both groups had better results after the intervention, and these differences can be attributed to the intervention using the Computational Pedagogy model.

For the self-efficacy in CT, results for both groups showed a statistically significant difference for the pre and posttest, with students of the experimental group to achieve higher scores.

Results provide support for the relationship among Computational Pedagogy model and STEM self-efficacy as well as for the self-efficacy in CT, while VAs seem to have a strong impact only on the "classical" metacognitive and self-efficacy skills.

Conclusions

The driving goal of this research was to create and test teaching and learning sequence based on the Computational pedagogy model, differentiated only in the form of programming (visual or text-based).

The difficulties presented in computer programming can be reduced by using optical programming tools like Scratch, AppInventor, Alice (see e.g. Armoni, 2011; Gokcearslan & Alper, 2015) and their effectiveness has been proved (see e.g. Lye & Koh, 2014; Kalelioğlu, 2015). There is strong research support for the positive impact of visualization of algorithms on students' learning (e.g., Boticki et al., 2013; Brusilovsky & Su, 2002; Cetin & Andrews-Larson, 2016; Konecki & Mrkela, 2014; Reif & Orehovacki, 2012).

However, our intention was not to investigate this issue but to investigate the impact of VAs on the general/classical form of self-efficacy-metacognition and the self-efficacy related to CT, when the computational pedagogy model is applied in the teaching process. Our intention was also to apply these methodologies to tertiary level students. In our research process, both groups were involved in programming using the Computational Pedagogy model. The difference between the two groups concerned the type of programming that was engaged.

It is well known that evaluation of programming as a teaching method is based on various psychological variables which include higher order cognitive and affective processes (Erdogan et al., 2008; Korkmaz & Altun, 2014) and self-efficacy is considered as one of them and examining the self-efficacy of students is considered as an essential factor in terms of how successful they are in programming (Aşkar & Davenport, 2009; Anastasiadou & Karakos, 2011).

Computational thinking is also related to programming. According to some research papers (e.g., Thomas & Velthouse, 1990; Wolber, Abelson, Spertus, & Looney, 2015; Resnick et al., 2009) programming empowerment is considered as a person's perceived self-sufficiency and competence to use computational thinking while programming is considered as a proper process to cultivate CT .

Lye & Koh (2014) consider that Programming (which is differentiated from coding), can help students to be engaged in computational thinking in the form of problem-solving, abstraction and decomposition. Among the findings of (Lye & Koh, 2014) are that variables and loops are computational thinking concepts that students learn via programming and Visualizing programming code output helps students to understand computational thinking

Grover, Pea, and Cooper (2015), developed and tested a computer science course for middle school students, called 'Foundations of Advancing Computational Thinking' (FACT) and reported that using this it provides strong indications for gains in dimensions of CT, like algorithmic thinking.

According to (Weinberg, 2013), programming teaching via visual programming can be an effective method for teaching the dimensions of computational thinking to students. Block-based visual programming languages are considered as proper programming for young learners (Kölling, 2015). Brennan & Resnick's (2012) developed a framework for studying and assessing the development of computational thinking by introducing concepts, practices and perspectives using block-based programming

Seiter and Foreman (2013) proposed a model, the Progression of Early Computational Thinking (PECT), for evaluating progression in computational thinking, and they found that using Scratch, students increase their computational thinking skills. From the research we can conclude that Computational thinking and its related concepts should also be implemented in learning and teaching methodologies, such as visual programming. In some countries, e.g. in England, learners are required, as they move to secondary school in computing lessons, to learn text-based programming languages instead of block based programming (DfE, 2013). In our work, EjsS was used as a visual programming language, which is not block based but offers the tools to create visual objects whose actions are determined by a graphical interface.

Work on the relation between self-efficacy and programming has been reported in many research papers (e.g. Korkmaz&Altun,2014).Authors used the "Computer Programming Self-Efficacy Perception Scale" (CPSEPS) instrument designed by Ramalingam and Wiedenbeck (1998) and they found that that the level of students' self-efficacy perception for programming in C++ in engineering faculties is not adequate and acceptable.

According to the literature, self-efficacy has been also suggested as a measure for computational thinking (Bell, 2014; Bean, Weese, Feldhausen, & Bell, 2015) and our results also support these findings. The responses on the questionnaire for the relation of self-efficacy with CT for both groups support this view and we can justify this result by taking into account that both groups used the Computational Pedagogy model which inherently is strongly related to the concepts of CT. Computational Pedagogy model can support Computational thinking concepts since it provides the practices in a form of a didactic model for the development of models of simulations and programming, which can lead to the design and development of computational abstractions such as developing artifacts that provide solutions to real life problems. To dive deeper in understanding self-efficacy of CT, future work will also include studying the application of EjsS to other real problems selected form students' curriculum in tertiary education. Our experimental results seem to support the hypothesis that making models of simulation in the environment of EjsS, improved students' self-efficacy for CT, as students considered that this environment helped them as it is like writing a kind of pseudo-code.

Further research is necessary for investigating the reasons that Computational Pedagogy integrated with text based programming (like C+) seems to have no impact on students' classical self-efficacy and metacognition. While this research has raised many questions, it also has provided key insights for the relationship between VAs and self-efficacy for CT. Findings from the self-efficacy for CT concepts (Weese, et. al., 2016) showed the importance the Computational Pedagogy model. Students of both groups gained higher scores, which can be attributed to this model.

Integrated STEM education continues to raise questions about the type of integration and how this integration can help students' self-efficacy for CT. Further research is necessary for including issues like development of visualized algorithms that impact CT dimensions like concepts and practices as well as dimensions like algorithmic thinking, pattern recognition etc.

In our approach the Computational Pedagogy Model was applied in the form that is described in Fig.1 and he phases of the Computational Pedagogy model are related to the dimensions of CT. Students of both groups followed this model but experimental group gained higher scores. This can be justified by the fact that visualization offers –maybe- more insight in the development of the model and its abstraction, but further research is necessary to generalize our results.

References

Aho, A. V. (2012). Computation and computational thinking. *Computer Journal, 55*(7), 832 – 835.

Anastasiadou, S.D., & Karakos, A.S. (2011). The beliefs of electrical and computer engineering students regarding computer programming. *The International Journal of Technology, Knowledge and Society, 7*(1), 37-51.

Andrienko, N., Andrienko, G., Barrett, L., Dostie, M., Henzi, S.P. (2013). Space Transformation for Understanding Group Movement. *IEEE Trans Vis Comput Graph., 19*(12), 2169–78.

Armoni, M. (2011). The nature of CS in K-12 curricula: the roots of confusion. *ACM Inroads, 2*(4), 19-20. doi:10.1145/2038876.2038883

Aşkar, P., & Davenport, D. (2009). An investigation of factors related to self-efficacy for java Programming among engineering students. *The Turkish Online Journal of Educational Technology TOJET, 8*(1): 26-32.

Baecker, R. (1998). Sorting Out Sorting: A Case Study of Software Visualization for Teaching Computer Science. In J. Stasko, J. Domingue, M. H. Brown, and B. A. Price (Eds.) *Software Visualization,* MIT Press, pp. 369--381.

Baird, J.R. (1990). Metacognition, purposeful enquiry and conceptual change. In E. Hegarty-Hazel (Ed.), *The student laboratory and the science curriculum (pp. 183–200). London: Routledge.*

Bandura, A. (1997). *Self-efficacy: The exercise of control.* New York, NY: W.H. Freeman & Company.

Bean, N., Weese, J., Feldhausen, R., & Bell, R. S. (2015). Starting from scratch: Developing a pre-service teacher training program in computational thinking. In *Frontiers in Education Conference (FIE),* 2015 IEEE (pp. 1-8). IEEE.

Bell, R. S. (2014). *Low Overhead Methods for Improving Capacity and Outcomes in Computer Science.* Manhattan, KS: Kansas State University.

Bienkowski, M., Snow, E., Rutstein, D. W., & Grover, S. (2015). *Assessment design patterns for computational thinking practices in secondary computer science: A first look.* SRI International

Blank, L.M. (2000). A metacognitive learning cycle: A better warranty for student understanding? *Science Education, 84*(4), 486–506.

Boticki, I., Barisic, A., Martin, S., & Drljevic, N. (2013). Teaching and learning computer science sorting algorithms with mobile devices: A case study. *Computer Applications in Engineering Education, 21*(S1), E41-E50.

Brennan, K., & Resnick, M. (2012). Using artifact-based interviews to study the development of computational thinking in interactive media design. *In Annual American Educational Research Association meeting,* Vancouver, BC, Canada.

Brusilovsky, P., & Su, H. D. (2002, June). Adaptive visualization component of a distributed web-based educational system. In *6th International Conference on Intelligent Tutoring Systems* (pp. 229-238). Springer Berlin Heidelberg.

Cetin, I., & Andrews-Larson, C. (2016). Learning sorting algorithms through visualization construction. *Computer Science Education, 26*(1), 27-43.

Chartier, T., & Kreutzer, E. (2010). How easy is 'easy java simulations' programming? Retrieved on December 28, 2016 from: http://www.maa.org/press/periodicals/loci/developers/how-easy-is-easy-java-simulationsprogramming

Chemers, M. M., Hu, L. T., & Garcia, B. F. (2001). Academic self-efficacy and first year college student performance and adjustment. *Journal of Educational psychology, 93*(1), 55.

Denning, P. (2003). Great Principles of Computing. *Communications of the ACM, 46*(11). 15-20.

DfE Computing programmes of study Key Stages 1 and 2 National Curriculum in England (2013). Department of Education. Retrieved from https://www.gov.uk/government/publications/national-curriculum-in-england-computing-programmes-of-study

Erdogan, Y., Aydin, E., & Kabaca, T. (2008). Exploring the Psychological Predictors of Programming Achievement. *Journal of Instructional Psychology, 35*(3).

Fife-Schaw, C. (2000). *Quasi-experimental designs.* In G. M. Breakwell, J. A. Smith, & D. B. Wright (Eds.), *Research methods in psychology (pp. 74–87).* California: SAGE Publications Ltd.

Fouh, E., Akbar, M., & Shaffer, C. A. (2012). The role of visualization in computer science education. *Computers in the Schools, 29*(1-2), 95-117.

Futschek, G. (2006, November). *Algorithmic thinking: the key for understanding computer science. In International conference on informatics in secondary schools-evolution and perspectives (pp. 159-168).* Springer, Berlin, Heidelberg.

Gilbert, J.K. (2005). Visualization: A metacognitive skill in science and science education. In J.K. Gilbert (Ed.), *Visualization in Science Education (pp. 9–27).* Dordrecht, The Netherlands, Springer.

Grover, Pea, & Cooper. (2015). Designing for deeper learning in a blended computer science course for middle school students. *Computer Science Education, 25(2), 199–237.*

Gokcearslan, Ş., & Alper, A. (2015). The effect of locus of control on learners' sense of community and academic success in the context of online learning communities. *The Internet and Higher Education, 27,* 64-73. Doi: 10.1016/j.iheduc.2015.06.003

Hundhausen, C. D., Douglas, S. A., & Stasko, J. T. (2002). A meta-study of algorithm visualization effectiveness. *Journal of Visual Languages & Computing, 13*(3), 259-290.

Juszczak, M. D. (2015). From Towards a Computational Pedagogy – Analysis of ABM Deployment in Pedagogical Instances. *International Journal of Pedagogy Innovation and New Technologies,* 2(1), 2-13. Doi: 10.5604/23920092.1159113

Kalelioğlu, F. (2015). A new way of teaching programming skills to K-12 students: Code.org. *Computers in Human Behavior, 52,* 200-210. doi:10.1016/j.chb.2015.05.047

Katai, Z. (2014). Intercultural computer science education. In Proceedings of the 2014 conference on Innovation & technology in computer science education (pp. 183-188). *ACM. 19*(1) (2014): 183-188. doi: 10.1145/2591708.2591744

Kölling, M. (2015). Lessons from the Design of Three Educational Programming Environments: Blue, BlueJ and Greenfoot. *International Journal of People-Oriented Programming (IJPOP), 4*(1), 5–32.

Konecki, M. & Mrkela, V. (2014). Students' acceptance of animated interactive presentation of sorting algorithms. *In Proceedings of the 17th International Multiconference Information Society - Human-Computer Interaction in Information Society* (pp. 18-21), Ljubljana, Slovenia.

Korkmaz, Ö., & Altun, H. (2014). Adapting Computer Programming Self-Efficacy Scale and Engineering Students' Self-Efficacy Perceptions. *Participatory Educational Research (PER), 1*(1), 20-31.

Landau, R. H., Páez, J. & Bordeianu, C. (2008). *A Survey of Computational Physics: Introductory Computational Science.* Princeton and Oxford: Princeton University Press.

Lu, J. J., & Fletcher, G. H. (2009). Thinking about computational thinking. *ACM SIGCSE Bulletin, 41*(1), 260-264.Proceedings of the 40th ACM technical symposium on Computer science education, March 04-07, 2009, Chattanooga, *TN, USA.* Doi:10.1145/1508865.1508959

Lye, S. Y., & Koh, J. H. L. (2014). Review on teaching and learning of computational thinking through programming: What is next for K-12? *Computers in Human Behavior, 41,* 51–61.doi:10.1016/j.chb.2014.09.012

Millsap, R. E., & Maydeu-Olivares, A. (2009). *The SAGE handbook of quantitative methods in psychology.* Thousand Oaks: SAGE.

Naps, T. L., Rößling, G., Almstrum, V., Dann, W., Fleischer, R., Hundhausen, C., & Velázquez-Iturbide, J. Á. (2002, June). Exploring the role of visualization and engagement in computer science education. In *ACM SIGCSE Bulletin, 35*(2), 131-152.

National Research Council. (2013). Next generation science standards: For states, by states.

Psycharis, S., Botsari, E., Mantas, P., & Loukeris, D. (2014). The impact of the Computational Inquiry Based Experiment on Metacognitive Experiences, Modelling Indicators and Learning Performance. *Computers & Education, CAE2501, PII: S0360-1315*(13)00278-9, DOI: 10.1016/j.compedu.2013.10.001

Psycharis, S. (2016a). 'The Impact of Computational Experiment and Formative Assessment in Inquiry Based Teaching and Learning Approach in STEM Education ; Journal of Science Education, and Technology 25(2),316-326 (JOST) DOI 10.1007/s10956-015-9595-z

Psycharis, S., (2016b). Inquiry Based- Computational Experiment, Acquisition of Threshold Concepts and Argumentation in Science and Mathematics Education Journal. *Educational Technology & Society, 19*(3).

Psycharis, S (2018a) STEAM in Education: A Literature review on the role of Computational Thinking, Engineering Epistemology and Computational Science. Computational STEAM Pedagogy (CSP). *Scientific Culture, 4*(2), 51-72. https://sci-cult.com

Psycharis, S. (2018b). Computational Thinking, Engineering Epistemology and STEM Epistemology: A primary approach to Computational Pedagogy. International Conference on Interactive Collaborative Learning, ICL 2018: *The Challenges of the Digital Transformation in Education,* 689-698. https://link.springer.com/chapter/10.1007/978-3-030-11935-5_65

Ramalingam, V., & Wiedenbeck, S. (1998). Development and validation of scores on a computer programming self-efficacy scale and group analyses of novice programmer self-efficacy. *Journal of Educational Computing Research, 19*(4), 367-381.

Ramalingam, V., LaBelle, D., & Wiedenbeck, S. (2004). Self-efficacy and Mental Models in Learning to Program. *ACM SIGCSE Bulletin, 36*(3), 171-175.

Reif, I., & Orehovacki, T. (2012, May). *ViSA: Visualization of sorting algorithms. In Proceedings of the 35th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO) (pp. 1146-1151).* Opatija, Croatia: IEEE.

Resnick, M., Maloney, J., Monroy-Hernandez, A., Rusk, N., Eastmond, E., Brennan, K., et al. (2009). Scratch: Programming for all. *Communications of the ACM, 52*(11), 60-67.

Seiter, L., & Foreman, B. (2013). Modeling the learning progressions of computational thinking of primary grade students. *Proceedings of the ninth annual international ACM conference on International computing education research (pp. 59–66). ACM.*

Sengupta, P., Kinnebrew, J. S., Basu, S., Biswas, G., & Clark, D. (2013). Integrating computational thinking with K-12 science education using agent-based computation: A theoretical framework. *Education and Information Technologies, 18*(2), 351-380.

Shaffer, C. A., Cooper, M., & Edwards, S. H. (2007). Algorithm visualization: a report on the state of the field. *ACM SIGCSE Bulletin, 39*(1), 150-154.

Shaffer, C. A., Cooper, M. L., Alon, A. J. D., Akbar, M., Stewart, M., Ponce, S., & Edwards, S. H. (2010). Algorithm visualization: The state of the field. *ACM Transactions on Computing Education (TOCE), 10*(3), 9.

Shaffer, C. A., Akbar, M., Alon, A. J. D., Stewart, M., & Edwards, S. H. (2011, March). Getting algorithm visualizations into the classroom. In *Proceedings of the 42nd ACM technical symposium on Computer science education* (pp. 129-134). ACM.

Shaw, K., Gurkas, P., & Webster, Z. (2012). An Analysis of Factors Expected to Impact Student End-of-Course Grades in Introductory College Science Classes. *Perspectives in Learning, 13*(1), 5.

Staley, J. D. (2006). Imagining the multisensory classroom. *Campus Technology, 6*(6).

Thomas, K., & Velthouse, B. (1990). Cognitive elements of empowerment: An interpretive model intrinsic task motivation. *Academy of Management Review, 15,* 666–681.

Thomas, G.P., & McRobbie, C.J. (2001). Using a metaphor for learning to improve students' metacognition in the chemistry classroom. *Journal of Research in Science Teaching, 38*(2), 222–259.

Thomas, G., Anderson, D., & Nashon, S. (2008). Development of an instrument designed to investigate elements of science students' metacognition, self-efficacy and learning processes: The SEMLI-S. *International Journal of Science Education, 30*(13), 1701-1724.

Törley, G. (2014). Algorithm visualization in teaching practice. *Acta Didactica Napocensia, 7*(1), 1.

Weese, J. L., Feldhausen, R., & Bean, N. H. (2016). The Impact of STEM Experiences on Student Self-Efficacy in Computational Thinking. *Proceedings of the 123rd American Society for Engineering Education Annual Conference and Exposition* (ASEE 2016). New Orleans, LA, USA.

Weinberg, A. E. (2013). *Computational thinking: An investigation of the existing scholarship and research. (Unpublished Doctoral Thesis),* Colorado State University, School of Education, Colorado.

Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L., & Wilensky, U. (2016). Defining computational thinking for mathematics and science classrooms. *Journal of Science Education and Technology, 25*(1), 127-147.

Wing, J. M. (2006). Computational thinking. *Communications of the ACM, 49*(3), 33-35.

Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical transactions of the royal society 366*(1881), 3717-3725.

Wing, J. (2011). Research notebook: Computational thinking-What and why? The Link Magazine. Retrieved from http://www.cs.cmu.edu/link/research-notebook-computational-thinking-what-and-why

Yasar, O., Veronesi, P., Maliekal, J., Little, L. J., Vattana, S. E., Yeter, I. H. (2016). *Presented at: ASEE Annual Conference and Exposition.* Presented: June 2016. Project: SCOLLARCIT

Wolber, D., Abelson, H., Spertus, E., & Looney, L. (2015). *App Inventor 2: Create your own Android apps.* Sebastopol, CA: O'Reilly

Yaşar, O. (2013). Teaching Science through Computation. International Journal of Science, *Technology and Society, 1*(1), 9-18. doi: 10.11648/j.ijsts.20130101.12