Educational Technologies
Образователни технологии

IMPLEMENTING COMPUTATIONAL THINKING IN IT TRAINING: AN INVARIANT FRAMEWORK FOR IT KNOWLEDGE FEATURES

Silvia Gaftandzhieva, Rositsa Doneva, Marieta Atanasova
University of Plovdiv "Paisii Hilendarski" – Plovdiv (Bulgaria)

Abstract. Contemporary education has been experiencing difficulties because of the fast changing content, especially in the field of Information Technology. In this regard, education institutions have seen an increasing emphasis on the role of computational thinking. Forming Computational Thinking in IT training would help to use fewer efforts and less time to update knowledge and skills in this area after a long period of time – no matter what new technologies, company developed software systems and interface tools have appeared. This article aims to propose an approach to form computational thinking when presenting learning content in order to overcome one of the basic challenges of contemporary IT training – the frequency and high change scales of the studied technologies and tools, which is all a result of high speed of technological advance. The article suggests an invariant framework, which could be used to form computational thinking skills in IT training. The framework allows trainers to formulate skills for pattern recognition, abstraction and focusing on the important information. As a proof for the efficiency of the offered invariant framework for IT knowledge in terms of the discussed problem, part of the learning content related to spreadsheet training has been presented.

Keywords: computational thinking; IT training; invariant knowledge; spreadsheet; framework

Introduction

Contemporary education has been experiencing difficulties not because of the volume of the available information (including the one on the Internet), but its fast changing content. Today, education institutions have seen an increasing emphasis on the role of computational thinking in all disciplines. In March 2006, Jeanette Wing (Wing, 2006) used the term computational thinking to make thinking like a computer scientist a fundamental skill for everyone without a precise definition of the term. Since 2006, many researchers have made many attempts to derive a definition for computational thinking (Barr and Stephenson, 2011; Guzdial, 2008; Denning, 2007; Mohaghegh and McCauley, 2016; Bocconi et al., 2016). Later in 2010, Wing, Cuny and Snyder (Cuny et al., 2010) state that com-

putational thinking for everyone means being able to understand which aspects of a problem are amenable to computation, to evaluate the match between computational tools and techniques and a problem, to understand the limitations and power of computational tools and techniques, to apply or adapt a computational tool or technique to a new use, to recognize an opportunity to use computation in a new way, and to apply computational strategies such as divide and conquer in any domain. Grover & Pea (Grover and Pea, 2013) described the following core concepts that build computational thinking skills: formulating problems, logically organizing and data analysis, data representation, algorithmic thinking, identifying, analysing, and implementing possible efficient and effective solutions and generalizing and transferring problems solving skills into other disciplines. According to other researchers the main characteristics of computational thinking include (Bilbao et al., 2016): analysing and logically organizing data, data modelling, data abstractions and simulations, formulating problems that computers may assist, identifying, testing, and implementing possible solutions, automating solutions via algorithmic thinking and generalizing and applying this process to other problems. According to Liu and Wang (Liu and Wang, 2010) computational thinking is a hybrid of other modes of thinking, like abstract thinking, logical thinking, modelling thinking, and constructive thinking.

Researchers examine the potential advantages of introducing computational thinking in compulsory education. They believe that computational thinking can enable students to think in a different way while solving problems, to analyse everyday issues from a different perspective (Lee et al., 2011), to develop the capacity to discover, create and innovate (Allan et al., 2010), to understand what technology has to offer (Mohaghegh and McCauley, 2016), etc. Curzon et al. (Curzon et al., 2014) develop a framework that helps explain what computational thinking is, describes pedagogic approaches for teaching it and gives ways to assess it. The framework includes four interconnected stages of development to our computational thinking framework: Definition, Concepts, Classroom techniques and Assessment.

Research objective and Methodology

In this section, a research question has been discussed, the aim of the article has been formed and a methodology to create Invariant framework for IT Knowledge has been suggested.

Information Technologies (IT) are generally recognized as very important issues at all education levels and as the base of the world infrastructure (García-Peñalvo, 2018). Many jobs either require computing skills or benefit from computational thinking in today's society (Yeh et al., 2011). These skills can help students develop analytical skills and problem abstractions that help them solve their daily problems on the job.

In most cases, teaching and learning educational content in IT in schools and universities has been based on specific instruments and software tools. The efforts are aimed mainly at converting students into users of computer tools. This has gone from being necessary to being insufficient, because the use of software applications means to manage a digital language that is obsolete in a time that is not proportional, in effort, to the time that has been invested in acquiring these skills (García-Peñalvo, 2018). According to Curzon et al. (Curzon et al., 2014) the IT curriculum which has been demonstrated through 'how' (for example, a software usage skill) or 'what' the students produced has several weaknesses. Firstly, the country's economy depends on technological innovation not just on the use of technology. Secondly, the pace of technology and organizational change is fast in that the IT skills learnt are out of date before a student leaves school. Thirdly, technology invades all aspects of our life and the typically taught office practice is only a small part of technology use today.

As a result of the dynamic development of IT, and more particularly – the shortened appearance cycle of newer and newer versions of the studied software instruments, learners and teachers encounter an essential problem: learning content and acquired knowledge become outdated too quickly. This is the reason why, knowledge should be updated, additional skills to work with newer versions and functional options of the software products should be acquired. All that makes the teaching method of the fast-developing IT area of a great importance. Therefore, the challenge is to prepare students to face the world in which they live, giving them the necessary cognitive tools to succeed in the digital world (García-Peñalvo, 2018).

Based on the problems, which arise as a result of the necessity of acquiring long-term IT knowledge, the following research question has been formed:

Can IT learning content be presented in a way that allows to acquire long-term knowledge, which is sustainable in time?

This article aims to suggest a framework, which could be used to form computational thinking skills in IT training. The latter skills save time and efforts to update the necessary knowledge to obtain the new skills needed to work with a different software application from the studied one, or its newer version. Based on the offered invariant framework for IT knowledge, we have described part of the learning content related to spreadsheet training.

In order to find the answer of the above research question and achieve the aim, an overview of IT learning frames has been made.

Yeh (Yeh et al., 2011) investigates the knowledge gap that non-computing major college students possess about computational thinking in an introductory MS Excel course by measuring their performance using spreadsheet functions in three categories: recall, application, and problem solving. The empirical result shows that students can recall the meaning of those functions but seem to have trouble using them correctly and precisely – they had problems with understanding the data

type, failure in translating problems to productive representations using spreadsheet functions, and inadequate stipulation of the computational representations in precise forms.

Settle and Perkovic (Settle and Perkovic, 2020) describe a framework for implementing computational thinking in a broad variety of general education courses. The framework is designed to be used by a faculty without formal training in IT and it includes examples of computational thinking in a variety of courses, including in the field of computer science. In one of these examples students will learn how to develop Excel workbooks for computing elementary statistics and compute simple statistical inference (confidence intervals, hypothesis testing and linear regression models) using the data analysis toolkit.

A team of researchers focusing on adapting Computational Thinking as defined by Jeanette Wing into a more applied framework in partnership with a broad set of IT professionals (L'Heureux et al., 2012) implement information technology problem-solving constructs and scenarios designed to cultivate computational thinking in information technology education at Bunker Hill Community College via a course entitled "IT Problem Solving".

In 'Invariants in IT training' (Totkov et al., 2010) a possible solution has been shown, which will allow to teach and obtain long-term-knowledge in IT. The solution consists of putting an emphasis on the so called "invariant knowledge" (i.e. the one that is relatively static in time and resistant to possible changes) when presenting/teaching learning content. The same approach has been used in (Somova et al., 2014) for teaching programming languages on the basis of invariants of programming algorithms.

Results

This section suggests a solution to the problem. It presents in details the invariant framework, which could be used to form computational thinking skills in IT training. As a proof for the efficiency of the offered invariant framework for IT knowledge in terms of the discussed problem, part of the learning content related to spreadsheet training is presented.

Forming Computational Thinking in IT training would help to use fewer efforts and less time to update knowledge and skills in this area after a long period of time – no matter what new technologies, company developed software systems and interface tools have appeared. This makes it necessary for the basic learning materials to be presented like "invariant" (independent on the current state of technologies, systems and interface) knowledge and skills, and the additional materials will specify ("build") them in the contemporary context. Thus, we should first have the "invariant knowledge", which is relatively stable (in accordance with origin and version of the software, elapsed period of time, type of technology, etc.). This way, all of the knowledge and skills gained will be long-term and more applicable in all

areas where IT could be useful, and not tightly connected to computer studies and IT in particular.

In order to obtain long-term knowledge and skills during the time of education, the learning/training approach should be focused on invariant knowledge. A possible solution which helps teachers to follow this approach is to prepare their learning materials according to the previously developed invariant frameworks. The invariant framework for IT knowledge (see Figure 1), proposed here, allows trainers to form some important computational thinking skills (Figueiredo and Alberto, 2017) as for example pattern recognition, focusing on the important information only and abstraction. The framework suggests the following main topics to be used while presenting invariant knowledge in some IT-related subject:

– A common scheme of the suggested resources *and services*;
– *Notions and terms*;
– *Functional specifications of* the software tools which automate the process;
– *Method of use*;
– *Application* in the relevant professional area;
– *Areas of "overlapping" and integration* of other technologies to solve specific problems, which are related to the professional area.

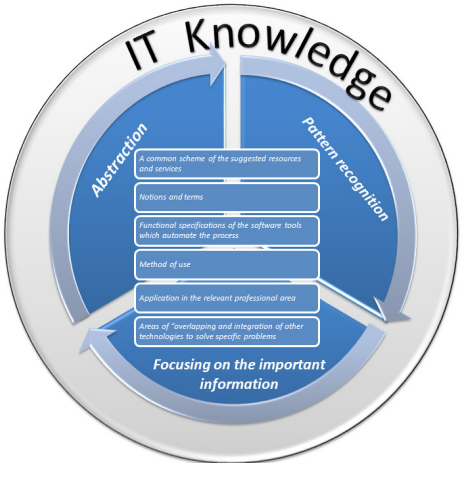

Figure 1. Invariant framework for IT Knowledge

The invariant framework for IT knowledge can be defined as a standard for invariant presentation of learning content, whose subject is a specific technology in the field of IT. In accordance with the invariant framework, the learning content in IT area should include basic topics related to the framework elements.

The learning content of "***A common scheme of the suggested resources and services"*** should get the learners acquainted with the basic terms of the technology itself, the common principles and stages of the relevant technology.

The second topic should define the core concepts involved in the technology. The developed learning content about ***"Notions and terms"*** defines the basic invariant notions and terms related to the logical structure of a document of the relevant type and the notions connected to the document content.

In the third theme ***"Functional specifications of the software tools"***, the learners should be presented with the functional specifications of the software tools necessary to work with a specific technology amongst which are the activities of creating and editing documents of the relevant type, functional possibilities which can make the document processing easy, the options to save the files, reproducing the document in different types of media and printing it on paper.

The learning content in ***"Method of use"*** should give important instructions related to designing and creating documents – to describe the basic stages which the document creation goes through (analysis, design, creation and exploitation) and the characteristic activities, which have to be completed during each one.

In ***"Application in the relevant professional area"***, some specific examples should be given for the wide application area of the technology in different professional areas.

In ***"Areas of overlapping and integration of other technologies to solve specific problems"***, we should give examples for the options to integrate software tools, a subject of the relevant technology, with specialized software products from different areas (for instance, education, business, accounting, architecture, construction, interior design and health care).

The learning content of each topic included in the invariant framework has to be described at an **abstract level** and be independent on a specific software tool and/or its version. This way, the learners will be able to acquire long-term knowledge, which is sustainable in time as well as skills to **recognize patterns** when it is necessary to work with a specific software tool.

When each topic of the suggested framework is being developed, it should include tasks, which will require the students to be able to:

– Describe basic notions and functionalities of the studied technology by not taking into consideration specific software tools (Type 1);

– Recognizing the patterns about a chosen software product by them, which a subject of the studied technology (Type 2).

When learning resources about each topic of the invariant framework are developed, the authors recommend to follow a suitable methodology about the invariant presentation of the learning content. Jabareen (Jabareen, 2009) thinks that building a conceptual framework from existent multidisciplinary literature is a process of theorization, which uses grounded theory methodology rather than a description of the data and the targeted

phenomenon. Phases from the proposed methodology for building conceptual framework (Jabareen, 2009) can be adapted and to be used as an excellent basis to create learning content following each point from the Invariant framework for IT Knowledge:

– Phase 1: Mapping the selected data sources;
– Phase 2: Extensive reading and categorizing of the selected data;
– Phase 3: Identifying and naming concepts;
– Phase 4: Deconstructing and categorizing the concepts;
– Phase 5: Integrating concepts;
– Phase 6: Making it all make sense;
– Phase 7: Validating the learning content;
– Phase 8: Rethinking the learning content.

In the first phase, the learning content authors should collect suitable literature sources about the current technology (about which learning content based on the suggested framework will be developed) and conduct interviews with specialists from the practice if possible. During the second phase of learning content development, the authors should make a review of the collected literature sources and classify the materials in accordance with the Invariant Framework for IT Knowledge. In the third phase, the authors identify and give names to the basic concepts related to the presented technology. Then, they should classify them according to the framework requirements and add their description (Phase 4). The aim of Phase 5 is to integrate and group together conceptions, which have common features in a new conception so that the number of notions will be decreased. During Phase 6, it is expected to develop a completed learning content about each topic from the Invariant Framework for IT Knowledge by following the invariance when presenting it. In Phase 7, the learning content has to be validated and evaluated (including by researchers from this area), which allows to double the long-term knowledge, independent on a specific technology. Based on the collected feedback from the evaluation process in the last phase, the learning content has to be updated.

Example content to learn Electronic tables

This section presents a fragment of learning content (Theme 3. Functional specifications of Spreadsheets software tools) in the field of spreadsheet training which is structured on the base of the "invariant framework" mentioned above and two tasks, as an evidence of the versatility and usefulness of the chosen approach to teach and learn IT related subjects.

Learning content: Functional specifications of Spreadsheets software tools

The program packages which help to work with a spreadsheet are called *table processors* and consist of three basic parts – editor (helps to introduce and edit content), calculator (makes calculations) and interface (visualizes the cell content in the wanted customer view). The editor and calculator provide the stage of creating and editing (including creating and modifying) from the computer processing of the tables while the interface provides their reproduction.

Spreadsheets provide the typical, when ***editing*** a computer document, meta-operations, operations to insert, delete, copy, move, find and replace the cells content as well as the characteristic table operations for deleting rows and columns, inserting new rows and columns, etc.

When the content of the documents is being processed, a spreadsheet includes creating (integration) and editing the data in the table.

The creation of spreadsheet is related to making a new workbook (a new document). The new cluster contains a specific number of worksheets from the spreadsheet type with standard names (for example, in Excel they are Sheet1, Sheet2, etc.). The appropriate tools help create new sheets, delete, rename, copy and move sheets within the same cluster or from one to another.

The next step is to integrate the content in the table. The separate cells content from the table can be inserted by different input devices (keyboard, mouse) or extracted from outer data sources. In terms of the cells content, the table processors distinguish some types of data aimed at facilitating automation of the work with them, for example: text, number, date or time, currency. Table processors provide opportunities for automated integration of ***serial data*** – sequent numbers, days, months, years or sequences defined by the user.

In order to make table calculations automated, it is necessary to use formulae in the cells.

The cells content must be in accordance with one of the basic characteristics of the cell – its content type. So when in-built functions are integrated, the content has to be in accordance with the arguments type, which the relevant function can accept. In order to avoid this type of mistakes, many of the table processors provide options to check the correctness of data integration.

After the cell content is integrated in the table, the chosen cells (or the whole electronic table) can be locked so that future unexpected value changes can be prevented. This is usually applied for the cells which contain formulae, but it could be used for the ones that contain constants as well.

The mistakes made during the content integration in the table can be corrected by deleting the necessary elements.

When *content is being processed* in a spreadsheet, the differentiation of the elements into static and dynamic has an important role. The content of the static elements (i.e. the ones which don't contain a formula) can be changed by the user. The content of dynamic elements is automatically generated by the table processor – for example, the calculator checks the cells for available formulae and if there are any, it makes the necessary calculations of the cell values, which are visualized after that on the visualizing screen (the interface of the table processor). The option of using relative, absolute or mixed references of a spreadsheet is interesting – the formula elements defining the value of the dynamic elements, which brings to a different type of calculation.

Spreadsheet software systems usually offer the option of ***sorting data***. Data arrangement is called sorting, and feature used for the arrangement is called sorting criteria. Data sorting can be done in ascending and descending order. When there is an ascending sorting of tables, which contain columns (or row) with different type of cells or just cells with alphanumeric text, the table processors use specific rules to sort data (for example, capital letters are placed in front of the other letters, etc.). In order to rearrange a spreadsheet area, the user have to select it first (mark it) and choose the criteria by which the arrangement will happen.

Searching and sorting data is not enough for quick finding of specific values in the tables. Tables very often contain big quantity of data and in a certain moment only a part of the data has to be processed. Data ***filtering*** helps to find a subset of data and working with it. After filtering, only the table rows, which fulfil certain requirements called ***filtering criteria***, are visible**.** In order to filter data, the user has first to select the column whose data will be defined when the filtering process is conducted, and then to choose the requirements, which the visible rows should meet (filtering criteria). Table processors allow data filtering by choosing simple criteria – only one requirement, or more complicated criteria – a logical function of more than one requirements.

Unlike sorting, filtering does not rearrange the rows. It just hides the rows, which do not meet the filtering criteria. Besides, after sorting filtered data, only the visible rows are rearranged.

A spreadsheet provides specific characteristics for each of the invariant elements of the workbooks, which can be changed aiming at more efficient assimilation of the created table by the person as an end user. For example, amongst the essential characteristics of the cell content format are the font, style, signs size; of the cell – size, content type, borders, background, etc.; of the worksheet – name, location in the workbook, editing options or automatic calculations, etc.; of the workbook – name, saving format, creation date and change of it, title, comments, author, file size, etc.

When creating formulae, the table processor provides a special ***language for formulae writing***, which allows users to write equations which have to be calculated. The language for ***formulae writing*** is generally easy, which is similar to the one used in Math (Table 1).

Table 1. Examples for formulae writing

Sign	Operation	Examples
+	Addition	2.3-3, A5+C3
-	Extraction	16.3-8, D1-C6
*	Multiplication	2*A3*D6
/	Division	B1/A1
^	Grading	A2^2

As operands in the equations, clearly stated constants can be used, cell values of the spreadsheet or cell areas, integrated functions in the table processor, or even complete expressions in brackets. Spreadsheet cells are written in the formulae through their references (names). In most of the table processors, the cells can be named and the users refer them by their names. This way of referring to variables (cells) in the formula allows an easy check of the formula in terms of arithmetic, which is expected to be realized. The reference (the name) means that as an operand of the equation, the value of the specific cell should be used. There is an option to use absolute or relative reference of the operands, where different syntax is used when writing the elements references of the spreadsheet (for example, A3 or A3). If the operand is set with an absolute reference, this means that it is defined by the value of the specific cell and cannot be changed. Whereas, if it is set with a relative reference, it can be changed depending on the cell location in the worksheet (for example, when copying, the relative references are modified by the editor as being automatically fitted into the new location on the worksheet). The formulae in most spreadsheets can connect cells in different tables (worksheets). In these cases, when they are modified, more complicated names are used.

After the formula is introduced, the ***calculator*** is activated, which does all the necessary calculations. The order in which the calculations are to be done follows the operations priority typical for Math (equations in brackets, functions, grading, multiplication, division and finally, addition and subtraction).

After each data update in the spreadsheet, the calculator is also activated, which does automatic ***recalculation***. When the automatic recalculations are completed, the visualizing part is activated, which shows the cells values on the display. That is, in fact, the general advantage of the spreadsheet – once created, the tables can be used a number of times to calculate different data.

The integrated functions in the table processors, which we can be used when writing formulae, help to perform additional operations. ***Function*** is a preliminary integrated named formula to conduct common calculations. The functions usually need arguments, do the calculation and respond with a value or values as a result. In the common case, ***the syntax of the functions*** is

```
<name> (<a list of arguments>)
```

, where <name> is the name of the function, and < a list of arguments > is a sequence of arguments divided by a specific divider (depending on the setting of the table processor). In table processors, there are a number of integrated functions: financial calculations, statistical analysis, date and time, math calculations, logical operations, etc. There is an option for functions to be defined by the user.

For quick data analysis in a table or visualizing a bulk of data, graphics in the form of ***diagrams*** are used. The basic element of the data about which a diagram is to be created is called ***a data point*** (for example the amount of production in

January), which has x-coordinate (***category***) and y-coordinate (***size***). ***A series of data*** (for example the amount of production in all months from January to April) is called a number of data points, which are connected by one category (the example with months). Important elements for each diagram are the ***axes***: the x axis is the one from the categories, and the y axis – from the sizes. Axes restrict the diagram area and are divided– the X axis by the different categories, and the У axis by number divisions, which show the size of the visualized data. The diagrams are usually provided with ***a legend***, which contains the names of the shown series and used signs (colours, ways of shading, etc.). The diagrams could have set names and axes definitions.

In order to construct a diagram, the user has to choose a rectangular zone from spreadsheet cells, which include all the data in the studied series. The separate rows (columns) correspond with different series, which should be illustrated. It is practical to add to the data an initial row (column), which contains characteristics of the data nature of the relevant column (row). Similarly, it is a good practice to add to the data an initial column (row), which contains the series names. The values connected with one of the series can be graphically presented in different ways. This formatting is common as ***type of the diagram.*** The most frequently used diagrams to present series of data are the linear, columnar, circular and functional diagrams. The type of the diagram is chosen depending on the type of the presented data and what tendency do we want to analyse or show.

The stage of ***reproduction*** of a spreadsheet again is based on the common scheme, which is typical for every computer document. Some of the following options are provided: *saving information on external memory* in the specific or standardized file format, which helps data transfer to another computer or computer application.

Similarly, *reproduction of the documents in different types of media* is provided. The spreadsheet can be visualized on the screen of a computer system (View) in a couple of visualization categories (for example, Normal, Page Layout, Full Screen, Print Preview), be printed on paper (Print) or be transferred electronically (Send).

Tasks

Task from Type 1: Make a research of the options to make easier access to the elements from the logical structure of the spreadsheet, which the software tools offer. For this purpose, discuss at least 5 different tools. Describe their basic functional access options to elements from the logical structure of the spreadsheet by not taking into account specific software tools.

Task from Type 2: By having in mind the presented functional options to sort data in the learning content, describe which are the specific steps to sort data in 3 software instruments for spreadsheet work, chosen by you. You can use the following spreadsheet:

Table 2. Example of a task

Step	Software tool 1	Software tool 2	Software tool 3
How do you choose a sorting range?			
How do you activate the sorting command?			
How do you choose sorting criteria?			

Conclusion

The current work is related to one of the basic challenges of contemporary IT training – the frequency and high change scales of the studied technologies and tools, which are all results of high speed of technological advance. In order to overcome this challenge, the authors sequentially follow the approach to form computational thinking when presenting learning content. In their opinion, this approach assures forming long-term fundamental knowledge, based on basic terms and IT principles, which allows future self-training and easy update of knowledge and skills for a short time.

As a result of the research an Invariant Framework for IT Knowledge has been suggested, which could be used as a base when developing learning content in IT, which as well allows to acquire long-term knowledge independent on specific software instruments. As a proof for the efficiency of the offered invariant framework for IT knowledge in terms of the discussed problem, part of the learning content related to spreadsheet training has been presented. The suggested Invariant Framework and presented learning content fragment provide a clear answer to the asked research question – IT learning content can be presented in a way that allows obtaining long-term knowledge, which is sustainable in time.

Currently, the offered invariant framework for IT knowledge is being experimented with students in engineering specialties. For experimental purposes, an e-learning course "Spreadsheets" has been created for IT training (http://pdu.uni-plovdiv.bg/). The e-learning content of the course, as well as the other e-learning elements, follow the framework, which is based on "invariance" when learning content has been presented.

The suggested invariant framework for IT knowledge can be developed in other areas of IT – web-development and design, database, hardware components, etc.

Acknowledgements

The paper is supported within the National Scientific Program "Young scientists and Post-doctoral students" in accordance with Appendix No. 11 of Council of Ministers Decision No. 577 of 17 August 2018.

REFERENCES

Wing, J. (2006). Computational thinking. *Commun. ACM*, 49, 33 – 35

Barr, V., & Stephenson, C. (2011). Bringing computational thinking to K-12: what is Involved and what is the role of the computer science education community?. *ACM Inroads*, 2, 48 – 54.

Guzdial, M. (2008) Education: Paving the way for computational thinking. *Commun. ACM*, 51, 25 – 27.

Denning, P. (2007). Computing is a natural science. *Commun. ACM*, 50, 13 – 18.

Mohaghegh, M., & McCauley, M. (2016). Computational Thinking: The Skill Set of the 21st Century. *International Journal of Computer Science and Information Technologies (IJCSIT)*, 7(3), 1524 – 1530.

Bocconi, S., Chioccariello, A., Dettori, G., Ferrari, A., & Engelhardt, K. (2016). Developing computational thinking in compulsory education – Implications for policy and practice. *EUR 28295 EN*

Cuny, J., Snyder, L., & Wing, JM. (2010). Demystifying Computational Thinking for Non-Computer Scientists

Grover, S., & Pea, R. (2013). Using a discourse-intensive pedagogy and android's app inventor for introducing computational concepts to middle school students. *In Proceeding of the 44th ACM technical symposium on Computer science education*, 723 – 728.

Bilbao, J., García, O., Rebollar, C., Bravo, E., & Varela, C. (2016). Beyond the Digital Competence: Computational Thonking, *Proceedings of INTED2016 Conference 7th-9th March 2016*, 8333 – 8338.

Liu, J., & Wang, L. (2010). Computational Thinking in Discrete Mathematics, *IEEE 2nd International Workshop on Education Technology and Computer Science*, 413 – 416.

Lee, I., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J., & Werner, L. (2011). Computational thinking for youth in practice. *ACM Inroads*, 2(1), 32 – 37.

Allan, W., Coulter, B., Denner, J., Erickson, J., Lee, I., Malyn-Smith, J., & Martin, F. (2010). Computational Thinking for Youth. *ITEST Small Working Group on Computational Thinking*.

Curzon, P., Dorling, M., Thomas, N., Selby, C., & Woollard, J. (2014). Developing computational thinking in the classroom: a framework, 15.

García-Peñalvo, FJ. (2018). Computational thinking. *IEEE Revista Iberoamericana de Tecnologías del Aprendizaje (IEEE RITA)*, 13(1), 17 – 19.

Yeh, KC, Xie, Y,& Ke, F. (2011). Teaching Computational Thinking to Noncomputing Majors Using Spreadsheet Function. *41st ASEE/IEEE Frontiers in Education Conference*.

Settle, A., & Perkovic, L. (2020). Computational Thinking across the Curriculum: A Conceptual Framework. *Technical Reports*. https://via.library.depaul.edu/tr/13. Accessed January 27, 2020.

L'Heureux, J., Boisvert, D., Cohen, R., & Sanghera, K. (2012). IT problem solving: an implementation of computational thinking in information technology. In *Proceedings of the 13th annual conference on Information technology education* (SIGITE '12), 183 – 188.

Totkov, G., Doneva, R., Besaleva, L., & Chakarova, I. (2010). Invariants in IT training, *in Proceedings of the National Conference Education in the Information Society*, 22 – 29.

Somova, E., Enev, J., & Totkov, G. (2014). Invariants in Programming training, *International scientific on-line journal Natural & Mathematical science*. 4(3), 25-30.

Figueiredo, Q., & Alberto, J. (2017). How to Improve Computational Thinking: a Case Study. *Education in the Knowledge Society*, 18(4), 35 – 51.

Jabareen, Y. (2009). Building a conceptual framework: philosophy, definitions, and procedure. *International journal of qualitative methods*. 8(4), 49-62.

✉ **Dr. Silvia Gaftandzhieva, Assoc. Prof.**
Department of Computer Science
Faculty of Mathematics and Informatics
University of Plovdiv "Paisii Hilendarski"
Plovdiv, Bulgaria
E-mail: sissiy88@uni-plovdiv.bg

✉ **Prof. Dr. Rositsa Doneva**
ECIT Department
Faculty of Physics and Engineering Technology
University of Plovdiv "Paisii Hilendarski"
Plovdiv, Bulgaria
E-mail: rosi@uni-plovdiv.bg

✉ **Dr. Marieta Atanasova, Assist. Prof.**
Department of Educational technologies
Faculty of Physics and Engineering Technology
University of Plovdiv "Paisii Hilendarski"
Plovdiv, Bulgaria
E-mail: marieta.atanasova@gmail.com