A systematic mapping study on teaching and learning Computational Thinking through programming in higher education.

Computational Thinking (CT) through programming in higher education is considered an important skill for students to become problem solvers and thrive in the new digital workplace. Despite the wide interest, a systematic map of CT through programming in higher education is still missing. The aim of this study is twofold. First, we aim to provide a systematic map of the relevant research by identifying the areas and sub-areas of CT through programming teaching and learning in higher education. Second, we aim to investigate these areas based on two dimensions: their evolution over the years and the branches to which CT is applied. For this purpose, we apply a systematic mapping methodology. Main results include the identification of the CT areas of Knowledge Base, Assessment, Learning Strategies, Tools, Factors and Capacity Building. Of these, Knowledge Base, Assessment and Tools have significantly evolved throughout the years, while Capacity Building has only recently emerged. In addition, the introduction of CT to undergraduate students and preservice teachers differs mainly in the tools used and the CT elements that are assessed. The study contributes to the field by providing a structured type of research conducted and identifying gaps and opportunities for future research.
Keywords: Computational Thinking, Programming, Higher education

Introduction

With new technological developments and their areas of application evolving rapidly, "21st-century skills" is one of the most widely used terms in today's educational debate. In the 21st century, individuals need to understand the true potential of computers in order to become effective creators of computational tools, thus participating in the fast-changing digital world (Angeli et al., 2016). In addition, higher education students need to be independent thinkers, problem solvers, and decision makers to thrive in their future professional lives and the new digital workspace (Silva, 2009).

Computational Thinking (CT) is in line with many aspects of 21st century skills (Lye & Koh, 2014) such as thinking creatively, reasoning systematically, and working collaboratively (Resnick et al., 2009). Wing (2006) defines CT as a way of "solving problems, designing systems and understanding human behaviour by drawing on the concepts of computer science", suggesting that CT is a skill for everyone, not just computer scientists.

CT is considered broader than programming and is often promoted through approaches from disciplines other than Computer Science. However, Programming is widely accepted as an ideal medium for CT development (Voogt et al., 2015). Specifically, programming offers the mechanisms for applying CT concepts and practices (Brennan & Resnick, 2012) and at the same time supports the cognitive aspects of CT, such as algorithmic thinking, abstraction, decomposition, and testing (Buitrago Flórez et al., 2017; Shute et al., 2017).

The acquisition of CT is widely discussed in K-12 education in relation to the acquisition of 21st century skills and digital competences (Angeli & Giannakos, 2020; Shuchi Grover & Pea, 2013). In addition, CT is also of interest for higher education research as its introduction to both Computer Science and non-major curricula is considered important. Although methods for teaching CT at K-12 level are being investigated in a large body of research, at higher education level, research on teaching CT as a fundamental skill set is still lagging behind (Czerkawski & Lyman, 2015).

Previous studies (Czerkawski & Lyman, 2015) review CT literature in higher education focusing on efforts to define CT, to implement CT in Computer Science curricula and efforts to integrate CT into other domain of application besides Computer Science. However, a systematic map of CT through programming in higher education is still missing.

A systematic map could offer in research development by providing a structured type of research that has been conducted by categorizing it (Petersen et al., 2008). This study aims to map teaching and learning CT through programming in higher education by identifying its areas and sub-areas. The starting point is considered to be the areas provided by the conceptual model CTPK-12 (Tikva & Tambouris, 2021) namely: Knowledge Base, Learning Strategies, Tools, Assessment, Factors, Capacity Building. In addition, we investigate the areas of CT teaching and learning cycle in higher education based on two dimensions: their evolution over the years and the branches to which CT is applied.

2. Theoretical Foundations

CT was firstly introduced by Papert (1980), who relates programming to procedural thinking skills. The term CT was reintroduced by Wing (2006) who defines CT as a process that "involves solving problems, designing systems, and understanding human behaviour, by drawing on the concepts fundamental to computer science" (Wing 2006, p.33). She points out that CT is a fundamental skill for everyone, not just for computer scientists and argues that "To writing and arithmetic, we should add CT to every child's analytical ability" (Wing 2006, p.33). Wing's definition has subsequently become a reference point for discussion on CT. However, various other definitions have emerged in the literature (Barr & Stephenson, 2011; Brennan & Resnick, 2012; Grover & Pea, 2013).

Definitions often draw upon programming and computing concepts or regard CT as a set of elements related both to computing concepts and problem-solving skills (Tang et al., 2020). International Society for Technology in Education (ISTE) and Computer Science Teacher Association (CSTA) (2011) developed an operational definition that includes, the following elements: (a) formulating problems in a way that enables us to use a computer and other tools to help solve them, (b) logically organizing and analyzing data, (c) representing data through abstractions such as models and simulations, (d) automating solutions through algorithmic thinking (a series of ordered steps), (e) identifying, analyzing, and implementing possible solutions with the goal of achieving the most efficient and effective combination of steps and resources; and (f) generalizing and transferring this problem solving process to a wide variety of problems. In addition to these elements (ISTE) and (CSTA) include the following attitudes to their operational definition: (a) confidence in dealing with complexity, (b) persistence in working with difficult problems, (c) tolerance for ambiguity, (d) the ability to deal with open ended problems; and (e) the ability to communicate and work with others to achieve a common goal or solution.

Selby (2013) defines CT as a though process that involves (a) the ability to think in abstractions, (b) the ability to think in terms of decomposition, (c) the ability to think algorithmically, (d) the ability to think in terms of evaluations; and (e) the ability to think in generalizations.

Shute et al. (2017) developed a competency model that includes the following facets: (a) Decomposition: Dissect a complex problem/system into manageable parts. The divided parts are not random pieces, but functional elements that collectively comprise the whole system/problem; (b) Abstraction: Extract the essence of a (complex) system. Abstraction has three subcategories: (i) Data collection and analysis: Collect the most relevant and important information from multiple sources and understand the relationships among multilayered datasets, (ii) Pattern recognition: Identify patterns/rules underlying the data/information structure, (iii) Modeling: Build models or simulations to represent how a system operates, and/or how a system will function in the future; (c) Algorithms: Design logical and ordered instructions for rendering a solution to a problem. There are four sub-categories: (i) Algorithm design: Create a series of ordered steps to solve a problem, (ii) Parallelism: Carry out a certain number of steps at the same time, (iii) Efficiency: Design the fewest number of steps to solve a problem, removing redundant and unnecessary steps, (iv) Automation: Automate the execution of the procedure when required to solve similar problems; (d) Debugging: Detect and identify errors, and then fix the errors, when a solution does not work as it should; (e) Iteration: Repeat design processes to refine solutions, until the ideal result is achieved; and (f) Generalization:

Transfer CT skills to a wide range of situations/domains to solve problems effectively and efficiently.

Brennan and Resnick's (2012) CT framework draws from programming in Scratch. Although, the context is specific, their framework has been highly adopted in studies outside Scratch. The framework includes the following three dimensions: a) CT concepts including Sequences, Loops, Parallelism, Events, Conditionals, Operators, Data, (b) CT practices including Being incremental and iterative, Testing and debugging, Reusing and remixing, Abstraction and modularity and (c) CT perspectives including Expressing, Connecting, Questioning.

CT as a mental construct that concerns each individual, has attracted research interest in both K-12 and higher education. The accumulation of research plethora has led to efforts to review the literature with an emphasis on K-12, higher education or both.

Czerkawski & Lyman (2015) review CT research in higher education, focusing on issues around definition and scope, CT strategies in computer science and efforts to infuse CT into disciplines outside the field of Computer Science. They report research suggesting that game-based learning and simulations strategies may be effective in teaching CT to students of Computer Science and STEM disciplines. Regarding the application of CT in various other disciplines, they suggest that field-appropriate methods and strategies are required. Furthermore, they discuss issues of digital divide and social equity, highlighting that colleges need to offer innovative programs and support to give students equal access to opportunities. In addition, Czerkawski & Lyman (2015) argue that educational technologists could play an important role in creating professional development material for universities interested in incorporating CT in their practices.

Tikva and Tambouris (2021) develop the CT through Programming in K-12 education (CTPK-12) conceptual model that identifies the concepts involved in the process of CT teaching and learning. The CTPK-12 model consists of the following areas: Knowledge Base Area, Learning Strategies Area, Assessment Area, Tools Area, Factors Area and Capacity Building Area (Table 1).

Table 1. CT Areas adopted from Tikva and Tambouris (2021)

CT Area	Definition
Knowledge Base Area	CT measurable elements and their classification.
Assessment Area	Assessment methods and frameworks for measuring CT through programming in K-12 education.
Learning Strategies Area	Learning strategies leveraged to enhance students' CT learning through programming in K-12 education.
Factors Area	Factors related to CT through programming acquisition in K-12 education.
Tools Area	Tools that are used or specifically developed for teaching and learning CT through programming in K-12 education.
Capacity Building Area	Capacity building needed for teaching CT through programming in K-12 competently.

Although the CTPK-12 model refers to CT through programming in K-12 education, it serves as the basis for this study as it presents all areas of teaching and learning CT in K-12 education. Therefore, in order to identify respectively the areas and sub-areas of teaching and learning CT through in higher education, we use as a starting point the areas of CTPK-12 model, while further identifying whether there are additional areas for teaching and learning CT through programming in higher education. We further analyze these areas based on two dimensions: their evolution over the years and the branches to which CT is applied.

3 Method

In order to achieve the study goal, we apply a Systematic Mapping Study based on Petersen's et al. (2008) methodology. This includes the following adapted steps.

Step1. Definition of research questions: Definition of research questions based on the study goal (Section 3.1)

Step2. Conduct search for primary studies: Conducting a structured search based on relevant search strings on scientific databases (Section 3.3).

Step3. Screening of Studies: Applying exclusion and inclusion criteria (Section 3.4).

Step4. Classification scheme Identification: Definition of the classification scheme.

Step5. Data Extraction and mapping process: Shorting the studies into the classification scheme and provide visualizations of the results. Fig. 1 presents the study method in terms of steps conducted and relevant outcomes.

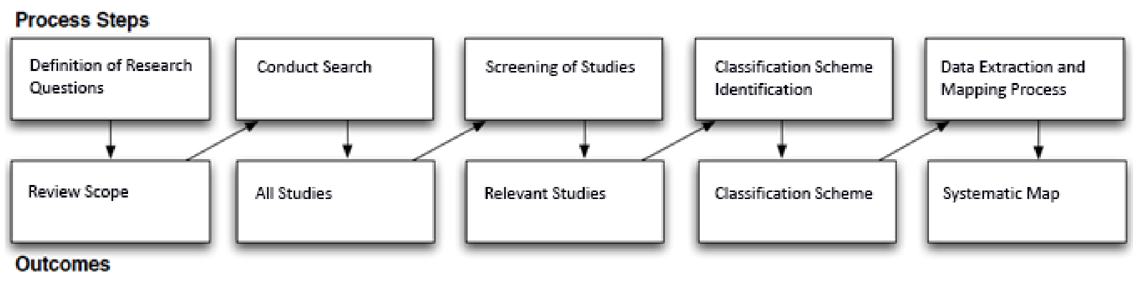


Fig.1. Systematic mapping process, adapted from Petersen et al. (2008)

3.1 Definition of Research Questions

The research questions are the following:

> **RQ1** What are the areas and sub-areas of teaching and learning CT through programming in higher education?
> **RQ2.** How do these areas evolve over the years and how do they apply to various branches?

3.2 Conduct search for primary studies

We structured the search string driven by the research study goal. Specifically, we used the search string *TITLE-ABS-KEY ("computational thinking") AND PUBYEAR > 2005 AND (LIMIT-TO (DOCTYPE , "ar") OR LIMIT-TO (DOCTYPE , "re")) AND (LIMIT-TO (LANGUAGE , "English"))* in Scopus database and *TITLE: ("computational thinking") Refined by: DOCUMENT TYPES: (ARTICLE OR REVIEW) AND LANGUAGES: (ENGLISH) Timespan: 2006-2020. Indexes: SCI-EXPANDED, SSCI, A&HCI, ESCI* in Web of Science database. Searches include articles published between January 2006 and December 2020. Searches resulted in 993 studies, 707 articles in Scopus database and 286 in Web of Science database.

3.3 Screening of studies

During this step we removed 249 duplicates and studies that were not fully availably. Subsequently, we applied inclusion and exclusion criteria to exclude studies that were not relevant to answering the research questions. Table 2 presents the exclusion and inclusion criteria defined. Finally, we included 39 primary studies and 2 additional primary studies that we identified through backward (reviewing citations) and forward searching. Appendix present the total of 41 studies included.

Table 2.

Inclusion and exclusion criteria.

Inclusion Criteria	Exclusion Criteria
Empirical CT studies in which participants are undergraduate students, postgraduate students and academic staff.	Studies which discuss/apply CT through other means than programming.
Empirical CT studies that focus on CT through programming.	

3.4. Classification Scheme Identification

We use as base for the classification scheme the areas of the CTPK-12 model presented in Section 2. Each Area of the model corresponds to one category in the classification scheme. Petersen et al. (2008) propose the extraction of the classification scheme based on keywording of abstracts of the selected studies. For this purpose, we read all the abstracts of the selected articles and wrote down keywords. Each keyword was assigned to one of the classification scheme categories in order to determine if there were any additional categories that could be included in the classification scheme.

3.5 Data extraction and mapping process

In this step we classify the selected primary studies into the classification scheme. According to Petersen et al. (2008) the classification scheme evolves while data extraction is performed. When sorting the selected primary studies into the categories, new sub-categories appear, while others remain unused. We used an Excel table per category to document the different instances of sub-categories in each primary study and the evolution of the classification scheme. When listing a primary study into a particular category and sub-category, we provide a brief rational for why the study should be located in that particular category/sub-category. The final tables show the distribution of primary studies into sub-categories and calculate the relevant frequencies. The analysis of the results focuses on comparing frequencies between different time periods and different targeted groups. This allows us to identify the categories and sub-categories highlighted in CT through programming in higher education research and therefore understand its evolvement and application.

3.6 Study Limitations

We acknowledge that this study has some limitations. First, the study includes only studies written in English. Second, searches were conducted in only two scientific databases, namely Web of Science and Scopus. Third, searches were conducted with a time constraint of 2006 onwards. Thus, the study maps the research conducted since 2006 and not on the initial stages of CT research. Finally, the small number of authors (only two) combined with subjectivity constitutes an additional limitation of the study. Although we applied a systematic mapping method, we had to make subjective choices regarding the evolution of the classification scheme.

4. Results

4.1 Overview

4.1.1 Studies by year

The distribution of studies by year (Fig. 2) reveals an upward trend in the number of studies. This is particularly true from 2017 onwards when the number of studies increases, suggesting that the field is generally beginning to mature. For this reason, we analyze the evolution of the field based on the two time periods 2006-2016 and 2017-2020.

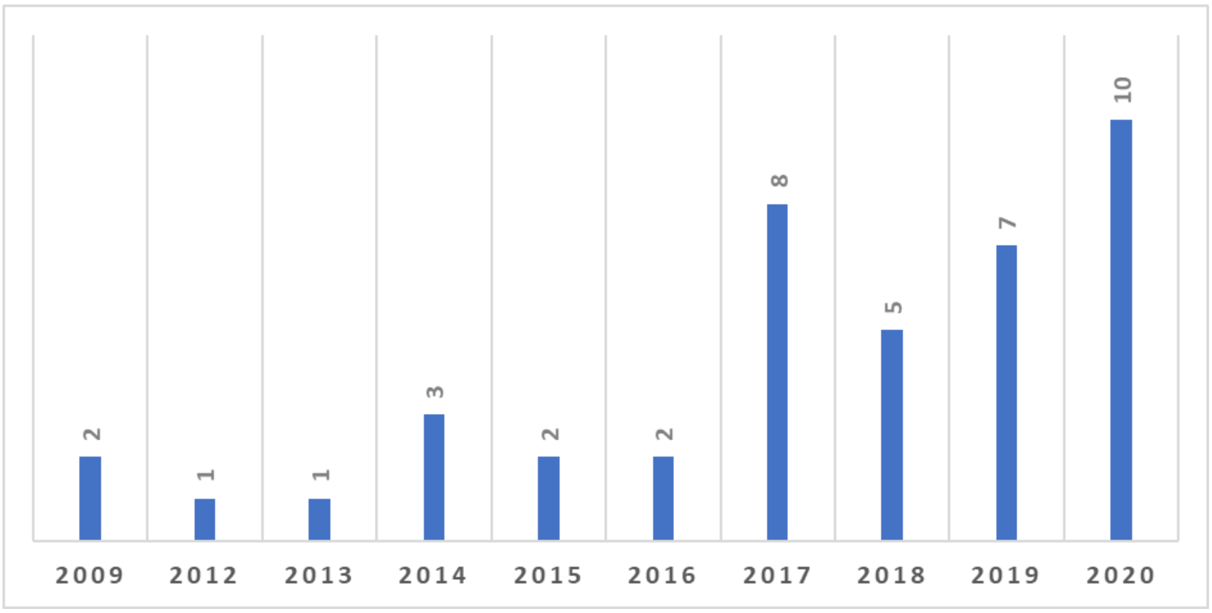


Fig. 2. Studies by year.

4.1.1 Interventions for CT development in higher education.

CT through programming empirical interventions in higher education (Table 3) mainly focus on Education majors, Natural Sciences majors and Computer Science (CS) majors. Table 4 presents the classification of branches based on the selected studies. The intense interest in Education branch led us to classify it as a separate branch in the context of this study. Fig. 3 presents the percentage of studies by branch in periods 2006-2016 and 2017-2020.

Table 3

Study	Content	Branch	Participants
(Adler & Kim, 2018)	Science methods course	Education	19 graduate and 13 undergraduate preservice teachers
(Bui et al., 2018)	Mindmaps and Scrath programming	Mathematics Education	50 preservice teachers
(Cachero et al., 2020)	Programming training	Health Information Systems, Psychology	104 undergraduate students

(Chao, 2016)	Principles and methods of C++ language programming	Information Communication	158 undergraduate students
(Choi, 2019)	Java programming class	Undefined	28 undergraduate students
(Cutumisu & Guo, 2019)	Educational Technology course	Education	139 preservice teachers
(Cetin, 2016)	Programming language course	Education	56 pre-service teachers
(Dolgopolovas & Jevsikova, 2015)	Structured programming (SP) course	Software Engineering	65 undergraduate students
(Fang et al., 2017)	Database Principles course	Computer Science and Technology	24 undergraduate students
(Fernandez et al., 2018)	Workshop	Education	21 inservice and pre-service teachers
(Fernandez et al., 2018)	Start to Programming course	Physics, Mathematics and Natural Sciences	22 undergraduate students
(Gabriele et al., 2019)	Programming course	Primary Education	141 preservice teachers
(Hambrusch et al., 2009)	Introduction to CT	Physics and Chemistry	13 undergraduate students
(Hou et al., 2020)	Programming course	Beauty Science	40 sophomore students
(Jaipal-Jamani & Angeli, 2017)	Science education methods course	Elementary Teacher Education	21 preservice teachers
(Jeon & Kim, 2017)	CT-based programming course applicable to liberal arts	Education	110 preservice teachers
(Kang & Lee, 2020)	Project-based learning course	Non-engineering majors	Undergraduate students
(Kazimoglu et al., 2012)	Introductory computer programming	Computer Science	25 undergraduate students
(Katai, 2020)	Sorting algorithms	Humanities, Science	48 undergraduate students
(Kwon & Kim, 2018)	CT and Software Coding & Problem Solving and Algorithm courses	Humanities, Social sciences and Arts	250 undergraduate students
(Lee & Cho, 2020)	Computer programming	Undefined	151 undergraduate students
(Lin & Chen, 2020)	Program Logic Thinking Education	Arts, Music, Chinese, Public Administration	97 undergraduate students
(Mouza et al., 2017)	Integrating Technology in	Education	21 preservice teachers

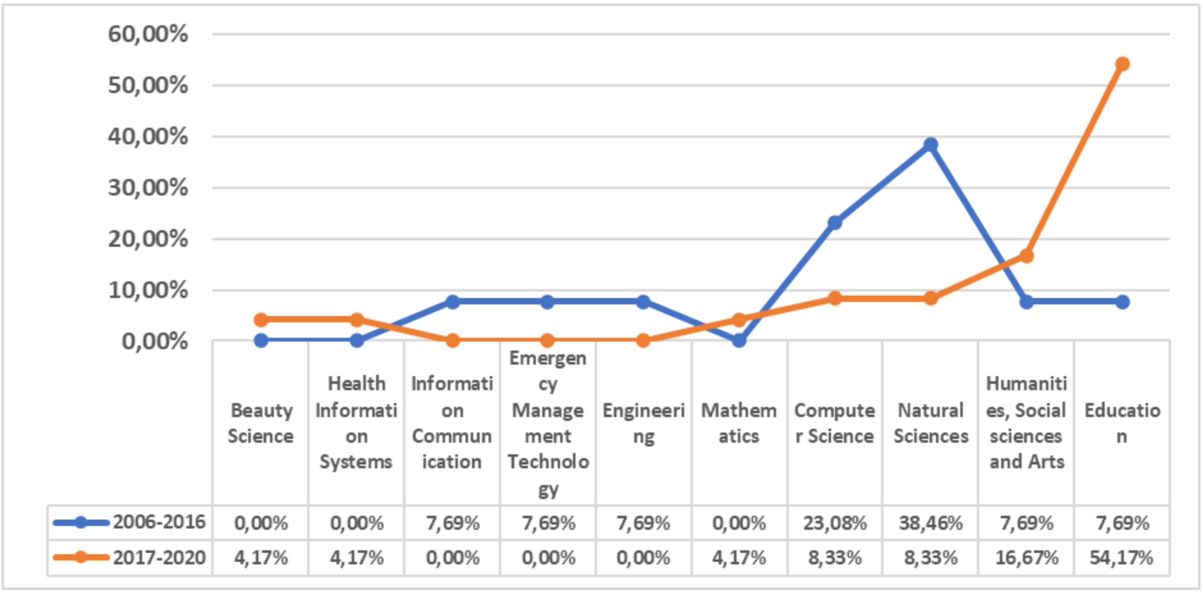
	Education program		
(Page & Gamboa, 2013)	How Computers Work: Logic in Action	Science, Engineering, History, Letters, Philosophy, Linguistics, Economics, Drama, Business, Psychology, Business, Computer Science, Computer Engineering	36 undergraduate students
(Pala & Mıhçı Türker, 2019)	Programming-I	Education	33 preservice teachers
(Qin, 2009)	Introduction to Bioinformatics	Biology	Undefined
(Rodríguez-García et al., 2020)	AI, ML and its societal implications workshop	Computer Science	14 students
(Romero et al., 2017)	StorytoCode creative challenge	Elementary School Education	120 preservice teachers
(Rubinstein & Chor, 2014)	Computational Approaches for Life Scientists	Biology	25 graduate and undergraduate students
(Shih et al., 2015)	Computer Applications in Emergency Management	Emergency Management Technology	18 undergraduate students
(Wu et al., 2019)	Introduction to C++ programming	Education	47 preservice teachers
(Yuen & Robbins, 2014)	Introductory computer science course (data-driven)	Biology	5 undergraduate students
(Zha et al., 2020a)	Educational Technology course	Education	59 preservice teachers
(Zha et al., 2020b)	Educational Technology course	Education	15 preservice teachers

Table 4

Classification of branches

Branch sub-category	**Description**
Majors (CS)	Computer Science including Computer Science and Technology, Computer Science, Computer Engineering, Software Engineering
Education majors	Education including Mathematics Education, Primary Education, Elementary School Education, Secondary Education
Non-majors in CS	Natural Sciences including Chemistry, Biology, Physics
	Humanities, Social sciences and Arts including History, Letters, Philosophy,

	Linguistics, Economics, Drama, Business, Psychology, Business, Arts, Music, Chinese, Public Administration.
	Engineering
	Mathematics
	Health Information Systems
	Information Communication Beauty Science



	Beauty Science	Health Information Systems	Information Communication	Emergency Management Technology	Engineering	Mathematics	Computer Science	Natural Sciences	Humanities, Social sciences and Arts	Education
2006-2016	0,00%	0,00%	7,69%	7,69%	7,69%	0,00%	23,08%	38,46%	7,69%	7,69%
2017-2020	4,17%	4,17%	0,00%	0,00%	0,00%	4,17%	8,33%	8,33%	16,67%	54,17%

Fig. 3. Percentage of studies by branch in periods 2006-2016 and 2017-2020.

During period 2006-2016 a percentage of 69,23% focuses on Computer Science, Engineering and Natural Sciences while only 7.69% focuses on Education majors. During the next period 2017-2020 the focus shifts from the aforementioned branches to Education. A 54,17% of interventions for CT through programming focus mainly on preservice teachers' preparation. Thus, we can conclude that there is an upward research trend for interventions aimed at Teacher Education.

4.2 CT teaching and learning areas in higher education

The classification scheme identification phase revealed that there are no additional areas in the selected studies other than those indicated by the CTPK-12 model (Table 1). Therefore, the areas of CT through programming in higher education which are analyzed and synthesized in the following sections are the following: Knowledge Base, Learning Strategies, Tools, Assessment, Factors, Capacity Building.

14.2.1 Knowledge Base

15 studies discuss elements of CT including domain specific elements, programming elements and higher-order skills. Table 5 presents the classification of CT elements in the selected studies. Fig. 4 presents the distribution of CT Knowledge Base sub-categories by periods 2006-2016 and 2017-2020. Table 6 presents the distribution of CT Knowledge Base sub-categories by classified branch.

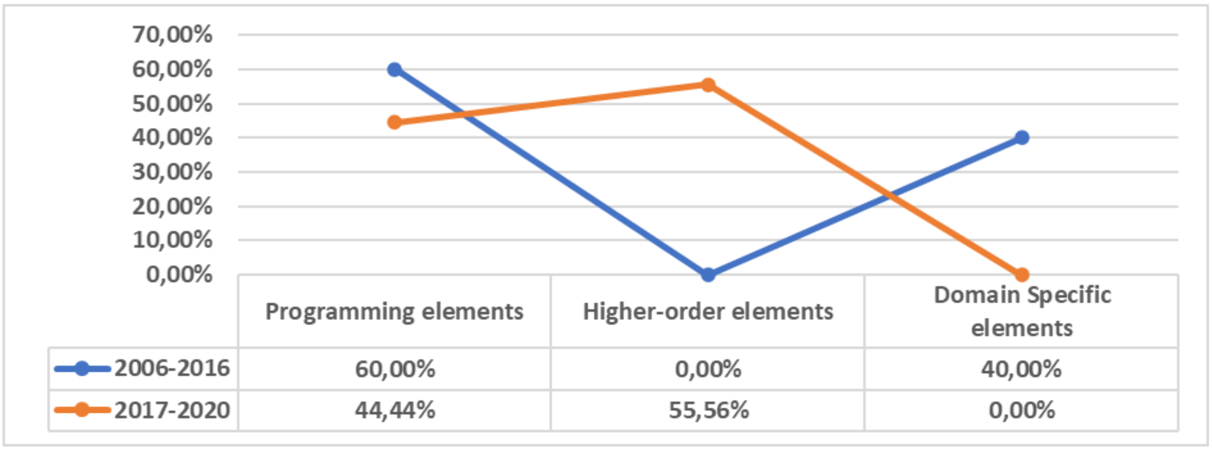
Chao (2016) investigates Computational practice (Sequence, Selection, Simple iteration, Nested iteration, Testing), Computational design (Problem decomposition, Abutment composition, Nesting composition) and Computational problem-solving performance (Goal attainment, Program size). Wu et al. (2019) adapts Brennan & Resnick's framework (2012), proposing Concepts (Sequence, Loops, Conditions, Operators, Data), Practices (Incremental and Iterative, Testing and Debugging, Reusing and Remixing, Abstracting and Modularizing) and Identities (Expressing, Questioning). In the same line, Cutumisu & Guo (2019) adopts Brennan & Resnick's framework (2012) for assessing CT concepts, practices and perspectives. Cetin (2016) investigates variables, conditional and selection statements, loops, arrays, and functions as CT elements. Yuen & Robbins (2014) investigates students' CT based on a coding scheme that includes Organization (Coding style, Data organization), Construction (Following procedures, Visualizing data) and Analysis (Interpretation and Conclusions). Jaipal-Jamani & Angeli (2017) investigate correct sequence, decisions on the flow of control and debugging.

Qin (2009) propose Multilevel abstraction and conceptualization, Iteration, recursion and backtracking, Modularization, Assessment and error corrections, Optimization and Simulation among other CT skill sets that are domain specific, derived from mapping CT skills to specific bioinformatics topics. In the same line, Rubinstein & Chor (2014) propose Abstraction, Generalization, Modular design and decomposition, Data structures and Computational models among other domain specific computational concepts and processes.

Other studies propose skills such as Abstraction, Decomposition, Recognition of Patterns and Algorithms (Fernández et al., 2018; Hou et al., 2020), Creativity, Algorithmic Thinking, Cooperativity, Critical Thinking and Problem Solving (Korkmaz et al., 2017; Lin & Chen, 2020; Pala & Mıhçı Türker, 2019). Sondakh et al. (2020) propose a holistic CT framework that includes the skills of Abstraction, Algorithmic Thinking, Decomposition, Debugging, Evaluation, Generalization and the attitudes of Problem solving, Teamwork and communication.

Table 5
CT Knowledge Base sub-categories

Knowledge Base sub category	**Description**	**Studies**
Domain Specific elements	CT concepts, skills and processes mapped to specific domains.	PS31, PS34
Programming elements	Programming related concepts, practices, identities and designs.	PS4, PS5, PS7, PS11, PS15, PS38, PS39
Higher-order elements	Higher-order thinking skills and competencies.	PS10, PS13, PS21, PS24, PS29, PS36



	Programming elements	Higher-order elements	Domain Specific elements
2006-2016	60,00%	0,00%	40,00%
2017-2020	44,44%	55,56%	0,00%

Fig. 4. Distribution of CT Knowledge Base elements sub-categories by time period.

Domain-specific elements are discussed in studies during period 2006-2016 while in period 2017-2020 these elements are absent. Higher-order elements are introduced during period 2017-2020 with a percentage of 60% in the selected studies of this period. Programming elements are discussed throughout the years.

Table 6. Percentage of studies' CT Knowledge Base elements sub-categories by classified branch.

Knowledge Base sub-category	Non-majors in CS	Education majors
Programming elements	33,33%	66,67%
Higher-order elements	33,33%	33,33%
Domain Specific elements	33,33%	0,00%
	100,00%	100,00%

Domain Specific elements are discussed only in studies targeted non-majors in CS. Programming elements have the strongest presence in the selected studies and particularly in Education majors.

4.2.2 Learning Strategies

Researchers in 24 studies discuss, propose or apply teaching and learning strategies for CT through programming in higher education. Out of these studies, seven apply more than one learning strategy or practice. We classify learning strategies in nine sub-categories, namely, **Game Based Strategies, Modeling & Simulations Based Strategies, Problem Solving Strategies, Project Based Strategies, Scaffolding Practices, Collaborative Strategies, Flipped Classroom, Hands-on strategies and Lectures**. Table 7 presents studies by each sub-category. Fig. 5 presents the distribution of learning strategies sub-categories by time periods 2016-2016 and 2017-2020. Table 8 presents the distribution of learning strategies sub-categories by classified branch.

Six studies discuss **Problem Solving Strategies** (Cetin, 2016; Hambrusch et al., 2009; Jeon & Kim, 2017; Kang and Lee, 2020; Lee & Cho, 2010; Yuen & Robbins, 2014). For example, Yuen & Robbins (2014) examine how undergraduate students develop CT skills during a data-driven programming course that encompasses problem-solving iterative processes. Lee & Cho (2020) exploit problem-solving methods to improve students' CT skills and logical thinking ability. Hambrusch et al. (2009) developed a course aimed at introducing students to CT based on a problem-driven format.

Four studies discuss **Collaborative Strategies**: Pair programming (Choi, 2019), Think-Pair-Share practice (Choi, 2019), Collaborative programming (Wu et al., 2019), teamwork (Jaipal-Jamani & Angeli, 2017; Zha et al., 2020b). Collaborative programming is proposed as an effective learning strategy to enhance students' CT in higher education (Wu et al., 2019). For example, Choi (2019) develops an instructional model that exploits Think-Pair-Share Strategy and pair programming. The results of this study show that collaborative strategies could help students learn CT and programming.

Three studies discuss **Project Based Strategies** (Ma et al., 2017; Wu et al., 2019). Wu et al. (2019) support that project-based learning contexts can help novice students develop different learning pathways to learn CT. In the same line, Ma et al. (2017) propose using project-driven learning strategies to enable students to acquire CT.

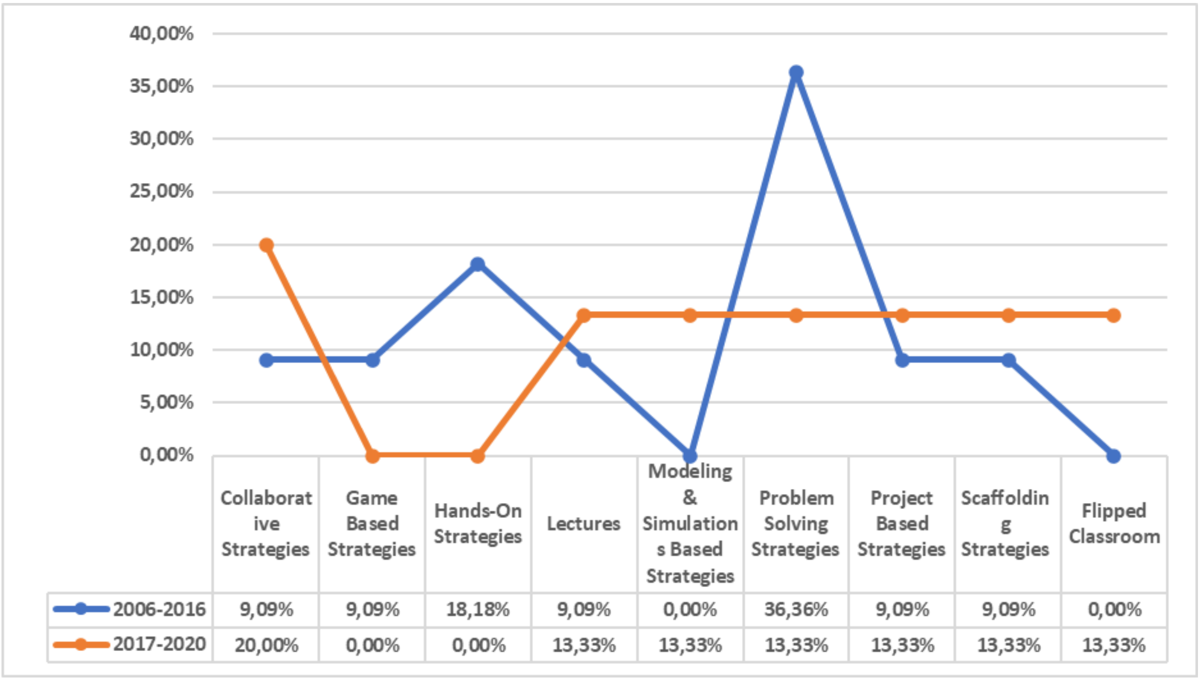
Three studies discuss **Scaffolding strategies** (Chao, 2016; Jaipal-Jamani & Angeli, 2017; Yuen & Robbins, 2014) usually combined with other strategies. Yuen & Robbins (2014) propose scaffolding as an effective learning strategy in order to enable students to focus on higher-order computational concepts without struggling with coding process in a text programming language such as MATLAB. In the same line, Chao (2016) argues that scaffolding may facilitate students to develop programming strategies and skills. Jaipal-Jamani & Angeli (2017) also found that the scaffolded programming instructional strategy they applied in their study, helped students to acquire CT.

Two studies discuss **Modeling & Simulations Based Strategies** (Adler & Kim, 2018; Magana & Silva Coutinho, 2017), two studies **Flipped classroom (**Zha et al.,2020a, Zha et al. 2020b) and one study **Game Based Strategies** (Kazimoglu et al., 2012**).** Specifically, Kazimoglu et al. (2012) propose a serious game where students develop their game strategies through programming based on an educational game framework for CT.

Two researchers choose to give **hands-on** activities (Qin, 2009; Rubinstein & Chor, 2014) and three use **lectures** (Cetin, 2016; Gabriele et al., 2019; Jaipal-Jamani & Angeli, 2017). Other strategies involve reflective learning (Choi, 2019), storytelling (Romero et al., 2017) and network autonomous learning (Li & Hou, 2014). Additionally, learning strategies are implemented in traditional classroom settings or in blended environments (Fernández et al., 2018; Mouza et al., 2017; Zha et al., 2020b).

Table 7. Learning strategies sub-categories

Learning Strategies sub-category	**Description**	**Studies**
Game Based Strategies	Game Based Related Strategies involve game design and digital/video game development, programming games and any strategy that exploits games and programming.	PS18
Modeling & Simulations Based Strategies	Modeling & Simulations Based Related Strategies involve designing of scientific models and simulations.	PS1, PS26
Problem Solving Strategies	Problem Solving Related Strategies involve Problem Based Learning and problem-solving learning strategies in general.	PS4, PS12, PS16, PS23, PS26, PS39
Project Based Strategies	Project Based Related Strategies involve the engagement with authentic projects set around real challenges and problems.	PS26, PS38
Scaffolding Strategies	Scaffolding Related Strategies involve practices that offer support to students as they learn.	PS6, PS15, PS39
Collaborative Strategies	Collaborative Related Practices involve practices where students actively interact during the learning process including Pair programming, Think-Pair-Share practice and any practice based on student's collaboration and cooperation.	PS5, PS15, PS38, PS40
Flipped Classroom Strategies	Flipped classroom Strategies involve strategies that reverse the traditional model of classroom instruction.	PS40, PS41
Hands-On Strategies	Hands-on activities	PS31, PS34
Lectures	Theoretical lectures	PS4, PS11, PS15



	Collaborative Strategies	Game Based Strategies	Hands-On Strategies	Lectures	Modeling & Simulations Based Strategies	Problem Solving Strategies	Project Based Strategies	Scaffolding Strategies	Flipped Classroom
2006-2016	9,09%	9,09%	18,18%	9,09%	0,00%	36,36%	9,09%	9,09%	0,00%
2017-2020	20,00%	0,00%	0,00%	13,33%	13,33%	13,33%	13,33%	13,33%	13,33%

Fig. 5. Distribution of learning strategies sub-categories by time period.

During period 2006-2016 problem solving Strategies have the strongest presence (36.36%), while during period 2017-2020 almost all learning strategies sub-categories occupy the same percentage (13.33%) with the exception of Game Based Strategies which has no presence at all and Collaborative Strategies which have a slightly stronger presence than the rest (20%).

Table 8. Percentage of learning strategies sub-categories by classified branch.

Learning strategies sub-category	CS majors	Education majors	Non-majors
Collaborative Related Strategies	0,00%	23,08%	12,50%
Game Based Related Strategies	33,33%	0,00%	0,00%
Hands-On Strategies	0,00%	0,00%	25,00%
Lectures	0,00%	23,08%	12,50%
Modeling & Simulations Based Related Strategies	0,00%	7,69%	0,00%
Problem Solving Related Strategies	33,33%	15,38%	25,00%
Project Based Related Strategies	33,33%	7,69%	12,50%
Scaffolding Related Strategies	0,00%	7,69%	12,50%
Flipped classroom	0,00%	15,38%	0,00%
	100,00%	100,00%	100,00%

No strategy seems to be dominant in any of the classified branches. In addition, as shown in Table 8, in studies aimed at preservice teachers and non-majors, a greater variety of studies is applied than in studies aimed CS majors.

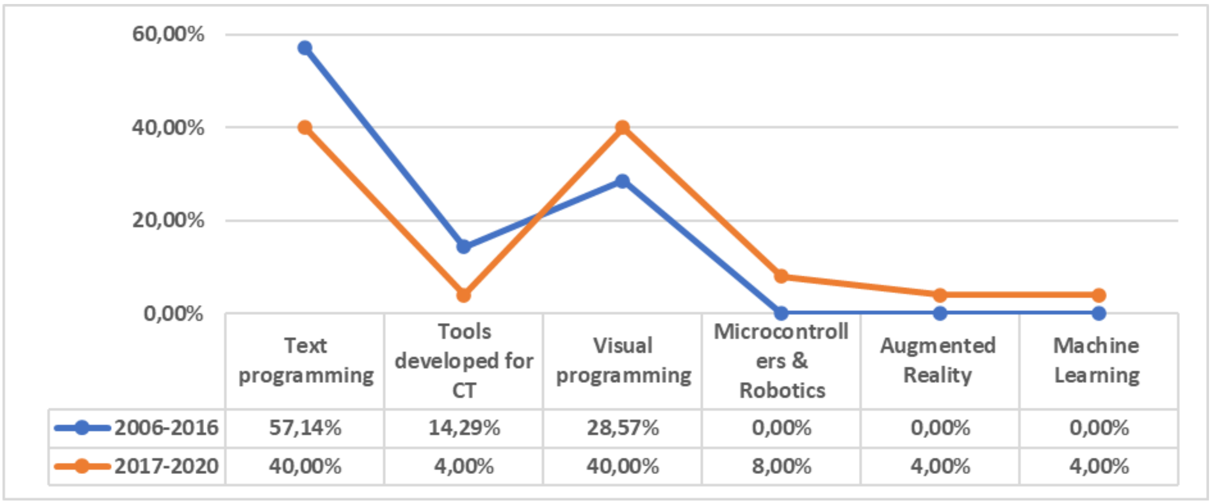
4.2.3 Tools

Researchers in 37 studies discuss, propose or exploit tools for CT teaching and learning in higher education. We classify tools in five sub-categories, namely, **Programming tools, Robotics & Microcontrollers, Augmented Reality Systems, Machine Learning tools and tools specifically developed for CT**. Table 9 presents tools sub-categories leveraged in the selected studies. Fig. 6 presents the distribution of tools sub-categories in periods 2006-2016 and 2017-2020. Table 10 presents the distribution of tools sub-categories by classified branch.

Eight studies exploit Scratch (Adler & Kim, 2018; Bui et al., 2018; Cetin, 2016; Gabriele et al., 2019; Hou et al., 2020, Mouza et al., 2017; Romero et al., 2017, Zha et al., 2020a), two studies Hour of Code (Adler & Kim, 2018; Mouza et al., 2017), one study Code.org (Cutumisu & Guo, 2019), one study App Inventor (Shih et al., 2015), one study ARDUINO IDE (Pala & Mıhçı Türker, 2019), one study LEGO® WeDo robotics (Jaipal-Jamani & Angeli, 2017), one study Java (Choi, 2019), one study Hopscotch (Zha et al., 2020b), one study HTML5 and CSS3 (Jeon & Kim, 2017) nine studies Python (Cachero et al., 2020; Dolgopolovas & Jevsikova, 2015; Hambrusch et al., 2009; Kang & Lee, 2020; Kwon & Kim, 2018; Lee & Cho, 2020; Magana & Silva Coutinho, 2017; Pala & Mıhçı Türker, 2019; Rubinstein & Chor, 2014), one study ACL programming language (Page & Gamboa, 2013), two studies C++ (Pala & Mıhçı Türker, 2019; Wu et al., 2019), three studies SQL (Huang & Leng, 2019; Qin, 2009; Fang et al., 2017), two studies MATLAB (Magana & Silva Coutinho, 2017; Yuen & Robbins, 2014), and four (Chao, 2016; Katai, 2020; Kazimoglu et al., 2012; Lin & Chen, 2020) studies develop a tool. For example, Chao (2016) develops a problem-solving programming environment and Lin & Chen (2020) develop a deep learning recommendation based augmented reality system.

Table 9. Tools sub-categories

Tools sub-category		**Studies**
Programming tools	Visual programming &	PS1, PS2, PS4, PS5, PS7, PS11, PS13, PS18, PS19, PS28, PS33, PS35, PS39, PS40
	Text programming tools.	PS3, PS6, PS8, PS9, PS12, PS14, PS16, PS17, PS22, PS23, PS27, PS29, PS30, PS31, PS34, PS35, PS38, PS39
Robotics & Microcontrollers		PS15, PS30
Augmented Reality systems		PS25
Machine Learning tools		PS32
Tools specifically developed to support a CT strategy		PS5, PS18, PS25, PS19



	Text programming	Tools developed for CT	Visual programming	Microcontrollers & Robotics	Augmented Reality	Machine Learning
2006-2016	57,14%	14,29%	28,57%	0,00%	0,00%	0,00%
2017-2020	40,00%	4,00%	40,00%	8,00%	4,00%	4,00%

Fig. 6. Distribution of tools sub-categories by period.

During period 2006-2016 text programming tools have the strongest presence (57.14%) while 28.57% of studies investigates visual programming. Subsequently during period 2017-2020 a 40% of studies investigating visual programming. Thus, an upward trend in visual programming is revealed. In addition, new tools such as Microcontrollers, Robotics, Machine Learning tools and Augmented Reality systems are introduced.

Table 10. Percentage of tools sub-categories by classified branch.

Tools sub-category	**CS majors**	**Education majors**	**Non-majors in CS**
Tools developed for CT	20,00%	0,00%	12,50%
Microcontrollers & Robotics	0,00%	13,33%	0,00%
Visual programming	20,00%	66,67%	25,00%
Text programming	40,00%	20,00%	56,25%
Augmented Reality	0,00%	0,00%	6,25%
Machine learning	20,00%	0,00%	0,00%
Total	100,00%	100,00%	100,00%

Visual programming is investigated mainly in studies that focus on preservice-teachers while it is not prevalent in studies that target Non-majors and CS majors. Text-programing is investigated in all branches while it is prevalent in studies that target Non-majors in CS (56,25%) and CS majors (40%).

4.2.4 Assessment

29 studies discuss CT through programming assessment methods. Assessment methods are classified in four sub-categories, namely, **Self-report methods, Tests, Artifact analysis and Observations**. Table 11 presents assessment methods applied in the selected studies. Fig. 7 presents the distribution of assessment sub-categories in periods 2006-2016 and 2017-2020. Table 12 presents the distribution of assessment sub-categories by classified branch.

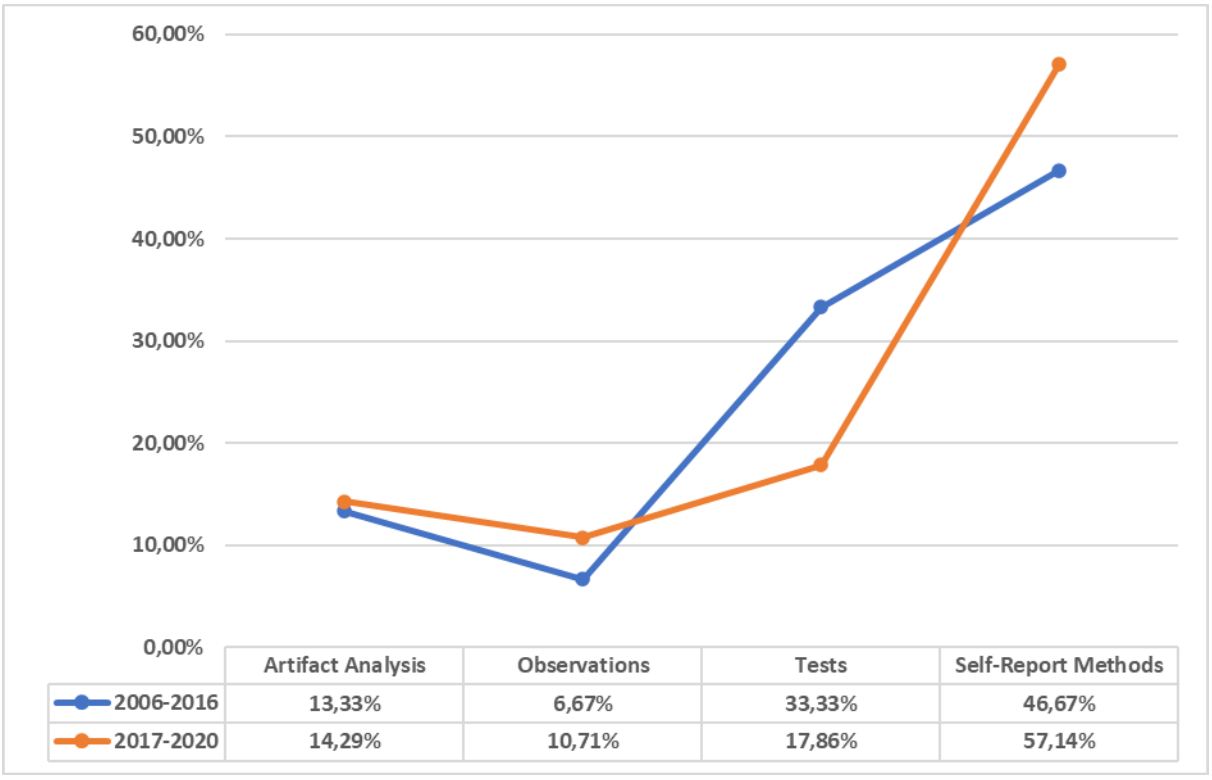
Four of the selected studies involve **observations**. Wu et al. (2019) record students' actions and conversations (screen and video recording) to examine how novice programmers develop CT by interacting with each other during collaborative programming and problem solving. More specifically, they investigate students' trajectories and their different CT development pathways. Screen recording is used to capture the programming process while video recording is used to capture student's conversations. Yuen & Robbins (2014) collect field notes during participants interviews.

Six of the selected studies involve **artifact analysis**. Chao (2016) collects log data about the participants' practice, strategies, and performance of computational problem-solving activities. Choi (2019) evaluates students’ programming artifacts. Yuen & Robbins (2014) collect source code from students’ in-class activities. Romero et al. (2017) analyze students’ projects through Dr. Scratch (Moreno-Leon et al., 2015) and manual inspection based on entities, events, code blocks and errors. Gabriele et al. (2019) analyzed students’ Scratch files through manual inspection for programming concepts, code organization and designing for usability adapted by Denner et al. (2012) and automatic inspection through Dr. Scratch.

23 studies exploit **self-report assessment methods.** Five studies exploit scales, three surveys, seven interviews, eight questionnaires and one study students’ reflections. Yuen & Robbins (2014) use interviews as their primary method for data collection. Shih et al. (2015) survey students’ perceptions about programming and their experiences with the applied CT intervention. Mouza et al. (2017) assess students’ CT knowledge based on a pre/post scale. Cutumisu & Guo (2019) used topic modeling techniques to extract participants CT understanding through their reflections. Researchers also develop and validate self-report scales in their studies. For example, Korkmaz et al. (2017) developed the CTS scale in order to assess students’ CT skills. The scale includes the items of Creativity, Algorithmic Thinking, Critical Thinking, Problem Solving and Cooperativity. Sondakh et al. (2020) propose a scale for CT assessment validated through Fuzzy Delphi Method that includes the items of Abstraction, Algorithmic Thinking, Decomposition, Debugging, Evaluation, Generalization, Problem solving, Teamwork, Communication and spiritual intelligence. In the same line, Kılıç et al. (2020) developed and validated a scale that includes the items of Conceptual Knowledge, Algorithmic Thinking and Evaluation. Finally, ten studies assess students’ CT through **tests and assignments**. For example, Jaipal-Jamani & Angeli (2017) used programming worksheets with completed, semi-completed and new programming tasks. Lin & Chen (2020) used multiple-choice and fill-in-the-blank questions to assess students’ programming understanding.

Table 11. Assessment sub-categories

Assessment sub-category	**Description**	**Studies**
Self-Report Methods	scales, questionnaires, surveys, interviews, reports, reflections	PS1, PS2, PS3, PS4, PS5, PS7, PS10, PS11, PS15, PS16, PS17, PS20, PS21, PS22, PS28, PS29, S30, PS31, PS34, PS35, PS36, PS39, PS40
Tests	multiple choice and open-ended tests, quizzes, tasks, assignments	PS3, PS4, PS15, PS18, PS25, PS31, PS32, PS34, PS39, PS40
Artifact analysis	automatic analysis, manually inspection of artifacts, log data	PS5, PS11, PS19, PS33, PS38, PS39
Observations	observations of students’ actions, screen recordings, camera recordings, field notes	PS2, PS37, PS39, PS40



	Artifact Analysis	Observations	Tests	Self-Report Methods
2006-2016	13,33%	6,67%	33,33%	46,67%
2017-2020	14,29%	10,71%	17,86%	57,14%

Fig. 7. Distribution of assessment sub-categories by period.

During period 2017-2020 an upward trend in the use of observations (+4,04%) and self-report methods (+10.47%) and a downward trend in the use of tests (-15.47%) is revealed in the assessment of CT. Artifact analysis shows a very small increase of 1.48%.

Table 12. Percentage of assessment sub-categories by classified branch.

Assessment sub-category	CS majors	Education majors	Non-majors in CS
Artifact Analysis	0,00%	15,79%	14,29%
Observations	0,00%	15,79%	7,14%
Tests	50,00%	10,53%	28,57%
Self-Report Methods	50,00%	57,89%	50,00%
Total	100,00%	100,00%	100,00%

Self-report methods have the strongest presence compared to other methods in studies targeted Non-majors in CS (50%) and education majors (57.89%).

4.2.5 Factors

Nine studies discuss factors that affect CT. Table 13 presents factors discussed in the selected studies. The effects that CT could have on interest in Computing and attitudes toward programming (Cetin, 2016; Hambrusch et al., 2009; Shih et al., 2015), self-efficacy (Jaipal-Jamani & Angeli, 2017; Kwon & Kim, 2018), creativity (Romero et al., 2017), interest in CT (Zha et al., 2020a), motivational impact (Katai, 2020), enrollment in CS courses (Hambrusch et al., 2009) and occupational change (Kwon & Kim, 2018) are discussed in the selected studies. CT-related factors are discussed through the years, 33.33% of the studies are published during 2006-2016 and another 66.67% during 2017-

2020. Furthermore, studies that investigate CT-related factors focus on both Education Majors (57.14%) and Non-majors in CS (71.43%).

Hambrusch's et al. (2009) study reveals that the problem-driven approach focused on computational principles and scientific discovery they applied, increased students' interest in CS. In the same line, Shih et al. (2015) found a positively change in students' perceptions about computing after they attended a course aimed to encourage students to apply CT and problem-solving skills to authentic problems. On the contrary, Cetin (2016) found no significant difference between control and experimental group students in terms of their attitudes towards programing. However, he suggests that this this is probably due to the short duration of the intervention and the difficulty of changing students' already high attitudes. Kwon & Kim (2018) conclude that a software education curriculum based on CT can stimulate students' intrinsic motivation and improve students' self-efficacy. In the same line, Jaipal-Jamani & Angeli (2017) found that after participated in a CT robotics program students' self-efficacy related to robotics and interest in learning robotics significantly increased. Kwon & Kim's (2018) study reveals that integrating CT could affect students' perspectives about their future occupation.

Table 13. Factors investigated in the selected studies.

Factors	Description	Studies
Non-Cognitive factors	Personal traits, attitudes and motivations such as attitudes toward programming, self-efficacy, creativity, interest in CS, perspective about future occupation.	PS4, PS12, PS15, PS19, PS22, PS27, PS33, PS35, PS41

4.2.6 Capacity Building

Only three of the selected studies discuss academic faculty training and professional development and they are all published in period 2017-2019. Table 14 presents methods regarding capacity building discussed in the selected studies.

Magana & Silva Coutinho (2017) survey industry and academia experts to identify the challenges facing academic staff in integrating CT at undergraduate level. Ma et al. (2017) suggest ways to improve university student's CT skills, including faculty professional training based on two principles: the mobility of academic staff and the organization of training programs. Taylor et al (2018) emphasize the role of collaboration between institutions as a means of motivating academic staff to redesign courses to integrate new concepts such as CT and coding.

Table 14. Capacity Building methods

Capacity Building	Description	Studies
Professional development	Variety of tools such as training programs, mobility of academic staff, collaboration between institutions.	PS26, PS27, PS37

5. Discussion and conclusions

The analysis of the selected studies revealed that the areas of Knowledge Base, Learning Strategies, Assessment, Tools, Factors and Capacity Building proposed by the CTPK-12 model also cover teaching and learning CT through programming in higher education. However, different sub-areas emerge in CT areas, while some of the model's sub-areas do not exist in the selected higher education studies. Fig. 8 presents the revised CTPK-12 model that corresponds to CT through programming in higher education. The CTPK-12 model also depicts the relationships between the areas of teaching and learning CT through programming as links between the areas shown in Fig. 8. The revised model could be use in order to develop research questions between the areas of teaching and learning CT through programming in higher education. For example, which learning strategies could be more appropriate for teaching CT domain specific elements, which for CT programming elements and which for CT higher-order skills.

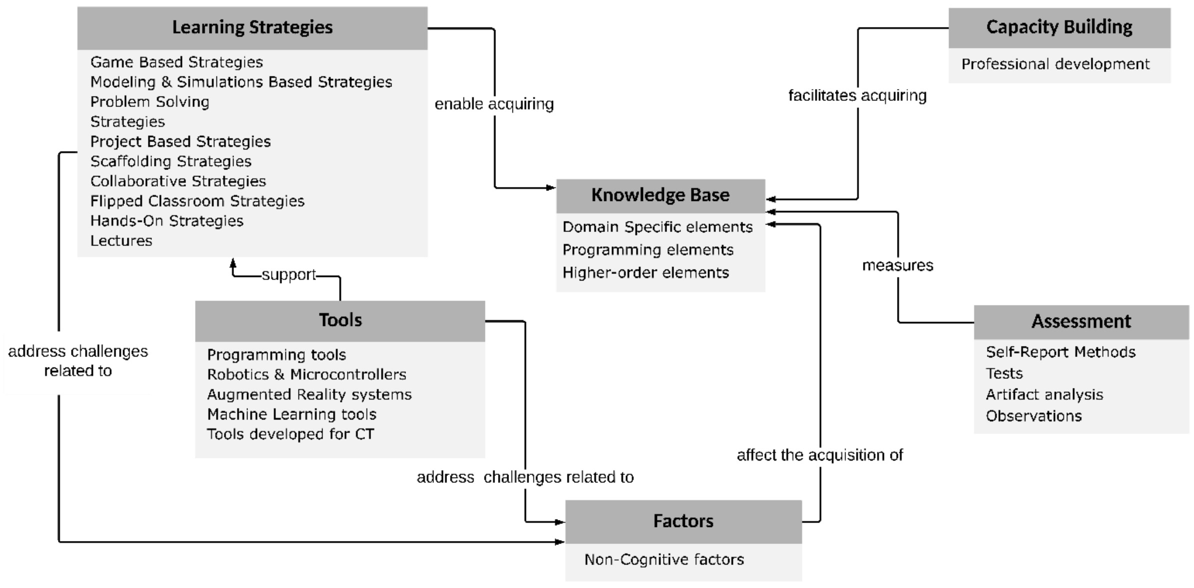


Fig. 8. The revised conceptual model for CT through programming in higher education (CTPHE).

Furthermore, as CT applications become more mature these areas evolve. Early attempts often link CT to domain-related elements, drawing on topics and activities related to specific courses and disciplines. However, in the coming years, CT is considered as a construct that is more associated with high-level skills such as abstraction and decomposition. Elements related to programming are most prevalent and evident throughout the years. This is plausible as CT draws from CS concepts according to Wing's (2006) definition.

CT through programming in higher education is traditionally implemented through text programming environments. However, the analysis of the selected studies revealed an upward trend in visual programming. This could be explained as visual programming is often applied to teacher education courses that have been at the forefront of CT higher education in recent years. In addition, tools such as Microcontrollers, Robotics and Augmented reality systems have recently emerged.

CT assessment is generally considered difficult to achieve by several authors (Brennan & Resnick, 2012; Denning, 2017; Fronza et al., 2017; Werner et al., 2012; Zhong et al. 2016). While self-report methods are the most common, the analysis of the selected studies also revealed a shift from tests to artifact analysis and observations in recent years. These methods are incorporated in order to provide a more complete picture of the CT acquisition.

Learning strategies and factors related to CT development such as personal traits, attitudes and motivations are discussed throughout the years, while academic faculty training and professional development gained attention only recently.

Teaching and learning CT through programming in higher education could be also organized in two areas: CT development for Non-majors and CS majors; and Teacher Education. The first concerns interventions and studies that propose the integration of programming aiming to help Non-majors and CS majors to acquire CT. The second concerns efforts to educate and support pre-service teachers with ultimate goal the integration of CT in K-12 education. The two areas present differentiation mainly in the tools used and the CT elements that are assessed with the second one to draw upon research on CT contacted in K-12 settings. Implementation of CT through programming for pre-service teachers is designed mainly on the basis of programming elements and includes mainly visual programming.

The analysis of the selected studies reveals that the focus of CT research in higher education is mainly on re-designing courses to align disciplinary knowledge with CT core concepts and to provide instructional models. The development of frameworks for learning strategies, tools and assessment methods is not extensively discussed in the selected studies.

Herein we also identify gaps that we discuss in the following paragraphs in an attempt to draw connections and implications from K-12 education where extensive efforts are being made worldwide to integrate CT.

In terms of learning strategies, although previous research has revealed that game design is often selected to introduce software engineering to students Souza et al. (2018), this is not the case for CT in higher education. There is no study in the selected studies that focuses on the development of CT through programming that applies game design learning strategy. In contrary, in K-12 education, game design is one of the most common strategies applied in several studies such as (Garneli & Chorianopoulos, 2018; Repenning et al., 2015; Weintrop et al., 2016). This is probably due to the capabilities of the tools offered to different age groups. In K-12 education, various tools such as Scratch (Resnick et al., 2009), and Agentsheets (Repenning et al., 2015) are utilized for game design and media computation, supporting the implementation of learning strategies that include game design learning. Although these tools are widely used in K-12 education and in higher education to prepare future teachers (Adler & Kim, 2018; Angeli et al., 2016; Gabriele et al., 2019), they are rarely used in interventions targeted other CS major or Non-major students. Text programming languages that are mainly used in higher education pose challenges to students such as dealing with complex syntaxes and abstract concepts (Buitrago Flórez et al., 2017) and require deep programming learning and experience to enable students to develop a game.

The importance of learning strategies in CT development is emphasized in both K-12 and higher education studies. Denner et al. (2012) study reveals that introducing CT to young students without applying a learning strategy, causes difficulties in developing students' CT skills. In the same line, Dolgopolovas & Jevsikova, (2015) argue that appropriate learning strategies should be exploited in order to facilitate CT skills development. They suggest that programming didactical approaches in higher education should focus on problem solving skills rather than language programming syntax.

Only few studies (Lee & Cho, 2020; Li & Hou, 2014; Ma et al., 2017) focus on creating frameworks by aligning learning strategies with CT. The bulk of research in higher education focuses on the implementation of learning strategies within specific courses and the development of instructional models.

Although there are studies that underline the role of communities in CT development (Xing, 2019) and the need to shift from tools to Communities (Clark & Sengupta, 2019; Kafai, 2016), as CT and programming are social practices, the exploitation of programming Communities in higher education is still lacking behind. Content-specific tools and mainly text programming languages are those applied in the higher education context. This in line with Magana & Silva Coutinho's (2017) study, showing that tools for teaching and learning CT in higher education are chosen on the basis of subjects rather than on their ability to support the acquisition of these skills. Exception are studies that focus on pre-service teachers that investigate mainly visual programming.

CT assessment in higher education applies the same assessment methods (Artifact Analysis, Observations, Tests and Self-report) as in K-12 education. However, the assessment is mainly carried out in the context of course evaluation. There are some efforts to develop universally accepted assessment methods but all of them are self-report methods. This is consistent with Lyon and Magana (2020) review that highlights the strong presence of self-report assessment methods in higher education CT studies. In addition, studies do not always attempt to validate the methods used and often do not yield quantitative results. Other challenges involve the small sample size and the lack of CT specific elements in the studies' results.

Moreover, often while studies present in the background various definitions of CT, they do not ultimately provide information on which elements of CT they focus on based on these definitions. Many times, they do not mention the CT context on which they are based, or display CT elements that are not based on a clear definition, are poorly documented and often vague.

Females and minority groups are often underrepresented in computing, as well as in technology labor (Jenson & Droumeva, 2016). Cooper et al. (2014) suggest that research in computing education should focus on gender and other minority groups. In addition, Shute et al. (2017) review the literature highlighting that researchers consider utilizing CT to motivate learners, especially females and minorities. However, there are limited studies (e.g., Zha, 2020a) in higher education that discuss the use of CT through programming to address issues related to female or underrepresented students. In addition, although gender as a factor affecting CT acquisition is particularly discussed in K-12 education (Atmatzidou & Demetriadis, 2016; Durak & Saritepeci, 2018), this is not the case for higher education. Studies in higher education do not focus on examining the relationship between gender and other social factors with CT.

Although teachers' knowledge and needs and their preparation to support students' understanding of CT are highly discussed in K-12 literature (e.g., Alfayez & Lambert, 2019; Angeli et al., 2016; Bower et al., 2017; Giannakos et al., 2015; Israel et al., 2015; Mouza et al., 2017; Yadav et al., 2017), research in higher education rarely focuses on faculty preparation. Only two of the selected studies involve higher education faculty (Magana & Silva Coutinho, 2017) or discuss opportunities for professional development (Ma et al., 2017).

To conclude, the results of this review indicate that several efforts have been emerged in CT through programming in higher education research recently, although challenges remain in the six areas identified in this review: Knowledge Base, Learning Strategies, Tools, Assessment, Factors and Capacity Building. Future studies should address remaining challenges by basing their design on clear definitions of CT as categorized and described in 1.4.2.1 section. The assessment should be based on the recording of CT elements as previously defined in the context of the studies. In addition, it is proposed to integrate direct assessment methods in combination with self-report methods in order to provide a more objective picture of the development of students' CT. The alignment of CT elements and assessment methods could provide a more comprehensive understanding of students' CT development. Future research should also explore how different learning strategies could support CT development. In addition, future research could focus on the development of tools suitable for higher education, which would enable the exploitation of game design strategies. Finally, studies should also focus on the investigation of how factors such as gender, creativity, self-efficacy, motivation may affect CT and how professional development of academic stuff could enhance the CT integration in higher education context.

Declaration of competing interest

None.

Appendix. List of the selected Primary Studies

PS1 Adler, R. F., & Kim, H. (2018). Enhancing future K-8 teachers' computational thinking skills through modeling and simulations. *Education and Information Technologies*, Vol. 23, pp. 1501–1514. https://doi.org/10.1007/s10639-017-9675-1

PS2 Bui, L. D., Kim, Y. G., Ho, W., Thi, H., Ho, T., & Pham, N. K. (2018). Developing WebQuest 2.0 model for promoting computational thinking skill. In *International Journal of Engineering & Technology* (Vol. 7). Retrieved from http://bit.ly/2jH9KT2.

PS3 Cachero C., Barra P., Melia S., Lopez O.,"Impact of Programming Exposure on the Development of Computational Thinking Capabilities: An Empirical Study",2020,"IEEE Access","10.1109/ACCESS.2020.2987254","https://www.scopus.com/inward/record.uri?eid=2-s2.0-85084334725&doi=10.1109%2fACCESS.2020.2987254&partnerID=40&md5=1333d67d45be18c0a20cbc955aa580d6"

PS4 Cetin, I. (2016). Preservice Teachers’ Introduction to Computing: Exploring Utilization of Scratch. *Journal of Educational Computing Research*, Vol. 54, pp. 997–1021. https://doi.org/10.1177/0735633116642774

PS5 Chao, P. Y. (2016). Exploring students’ computational practice, design and performance of problem-solving through a visual programming environment. *Computers and Education*, *95*, 202–215. https://doi.org/10.1016/j.compedu.2016.01.010

PS6 Choi, S.-Y. (2019). Development of an instructional model based on constructivism for fostering computational thinking. *International Journal of Innovative Technology and Exploring Engineering*, Vol. 8, pp. 381–385. Retrieved from https://www.scopus.com/inward/record.uri?eid=2-s2.0-85064633190&partnerID=40&md5=9c6da0548c789dfbb24134edc2cbfdc7

PS7 Cutumisu, M., & Guo, Q. (2019). Using Topic Modeling to Extract Pre-Service Teachers’ Understandings of Computational Thinking From Their Coding Reflections. *IEEE Transactions on Education*. https://doi.org/10.1109/TE.2019.2925253

PS8 Dolgopolovas, V., & Jevsikova, T. (2015). On Evaluation of computational thinking of software engineering novice students. *Proceedings of The*, *4*(2), 105–112. https://doi.org/10.13140/RG.2.1.2855.9206

PS9 Fang, A.-D., Chen, G.-L., Cai, Z.-R., Cui, L., & Harn, L. (2017). Research on blending learning flipped class model in colleges and universities based on computational thinking - “Database principles” for example. *Eurasia Journal of Mathematics, Science and Technology Education*, Vol. 13, pp. 5747–5755. https://doi.org/10.12973/eurasia.2017.01024a

PS10 Fernández, J. M., Zúñiga, M. E., Rosas, M. V., & Guerrero, R. A. (2018). Experiences in Learning Problem-Solving through Computational Thinking. *Journal of Computer Science and Technology*, *18*(02), e15. https://doi.org/10.24215/16666038.18.e15

PS11 Gabriele, L., Bertacchini, F., Tavernise, A., Vaca-Cárdenas, L., Pantano, P., & Bilotta, E. (2019). Lesson planning by computational thinking skills in Italian pre-service teachers. *Informatics in Education*, Vol. 18, pp. 69–104. https://doi.org/10.15388/infedu.2019.04

PS12 Hambrusch, S., Hoffmann, C., Korb, J. T., Haugan, M., & Hosking, A. L. (2009). A multidisciplinary approach towards computational thinking for science majors. *SIGCSE Bulletin Inroads*, Vol. 41, pp. 183–187. https://doi.org/10.1145/1539024.1508931

PS13 Hou H.-Y., Agrawal S., Lee C.-F.,"Computational thinking training with technology for non-information undergraduates",2020,"Thinking Skills and Creativity","10.1016/j.tsc.2020.100720","https://www.scopus.com/inward/record.uri?eid=2-s2.0-85090228702&doi=10.1016%2fj.tsc.2020.100720&partnerID=40&md5=a6c007fa4e1130f69a15118167520743"

PS14 Huang, X.-P., & Leng, J. (2019). Design of database teaching model based on computational thinking training. *International Journal of Emerging Technologies in Learning*, Vol. 14, pp. 52–69. https://doi.org/10.3991/ijet.v14i08.10495

PS15 Jaipal-Jamani, K., & Angeli, C. (2017). Effect of Robotics on Elementary Preservice Teachers’ Self-Efficacy, Science Learning, and Computational Thinking. *Journal of Science Education and Technology*, Vol. 26, pp. 175–192. https://doi.org/10.1007/s10956-016-9663-z

PS16 Jeon, Y., & Kim, T. (2017). The effects of the computational thinking-based programming class on the computer learning attitude of non-major students in the teacher training college. *Journal of Theoretical and Applied Information Technology*, *95*(17), 4330–4339.

PS17 Kang Y., Lee K.,"Designing technology entrepreneurship education using computational thinking",2020,"Education and Information Technologies","10.1007/s10639-020-10231-2","https://www.scopus.com/inward/record.uri?eid=2-s2.0-85085371936&doi=10.1007%2fs10639-020-10231-2&partnerID=40&md5=72af666d82419c9dc4344f0144fb7e18"

PS18 Kazimoglu, C., Kiernan, M., Bacon, L., & Mackinnon, L. (2012). A Serious Game for Developing Computational Thinking and Learning Introductory Computer Programming. *Procedia - Social and Behavioral Sciences*, *47*, 1991–1999. https://doi.org/10.1016/j.sbspro.2012.06.938

PS19 Katai Z.,"Promoting computational thinking of both sciences- and humanities-oriented students: an instructional and motivational design perspective",2020,"Educational Technology Research and Development","10.1007/s11423-020-09766-5","https://www.scopus.com/inward/record.uri?eid=2-s2.0-85083359163&doi=10.1007%2fs11423-020-09766-5&partnerID=40&md5=d27baf0027852e2a276dfd1c052f9fe7"

PS20 Kılıç S., Gökoğlu S., Öztürk M.,"A Valid and Reliable Scale for Developing Programming-Oriented Computational Thinking",2020,"Journal of Educational Computing Research","10.1177/0735633120964402","https://www.scopus.com/inward/record.uri?eid=2-s2.0-85092671278&doi=10.1177%2f0735633120964402&partnerID=40&md5=a767d80ae448790747589c61c0fe8ac6"

PS21 Korkmaz, Ö., Çakir, R., & Özden, M. Y. (2017). A validity and reliability study of the computational thinking scales (CTS). In *Computers in Human Behavior* (Vol. 72, pp. 558–569). https://doi.org/10.1016/j.chb.2017.01.005

PS22 Kwon, J., & Kim, J. (2018). A study on the design and effect of computational thinking and software education. *KSII Transactions on Internet and Information Systems*, Vol. 12, pp. 4057–4071. https://doi.org/10.3837/tiis.2018.08.028

PS23 Lee, Y., & Cho, J. (2019). Knowledge representation for computational thinking using knowledge discovery computing. *Information Technology and Management*. https://doi.org/10.1007/s10799-019-00299-9

PS24 Li, M., & Hou, D. (2014). Network autonomous learning based on computational thinking. *World Transactions on Engineering and Technology Education*, Vol. 12, pp. 576–580. Retrieved from https://www.scopus.com/inward/record.uri?eid=2-s2.0-84916237711&partnerID=40&md5=3d31af2cb32278ebb421ef9de6f7271f

PS25 Lin, P. H., & Chen, S. Y. (2020). Design and Evaluation of a Deep Learning Recommendation Based Augmented Reality System for Teaching Programming and Computational Thinking. *IEEE Access*, *8*, 45689–45699. https://doi.org/10.1109/ACCESS.2020.2977679

PS26 Ma, J. Bin, Teng, G. F., Zhou, G. H., & Sun, C. X. (2017). Practical teaching reform on computational thinking training for undergraduates of computer major. *Eurasia Journal of Mathematics, Science and Technology Education*, *13*(10), 7121–7130. https://doi.org/10.12973/ejmste/78738

PS27 Magana, A. J., & Silva Coutinho, G. (2017). Modeling and simulation practices for a computational thinking-enabled engineering workforce. *Computer Applications in Engineering Education*, *25*(1), 62–78. https://doi.org/10.1002/cae.21779

PS28 Mouza, C., Yang, H., Pan, Y.-C., Yilmaz Ozden, S., & Pollock, L. (2017). Resetting educational technology coursework for pre-service teachers: A computational thinking approach to the development of technological pedagogical content knowledge (TPACK). *Australasian Journal of Educational Technology*, Vol. 33, pp. 61–76. https://doi.org/10.14742/ajet.3521

PS29 Page, R., & Gamboa, R. (2013). How Computers Work: Computational Thinking for Everyone. *Electronic Proceedings in Theoretical Computer Science*, *106*(Tfpie 2012), 1–19. https://doi.org/10.4204/eptcs.106.1

PS30 Pala, F. K., & Mıhçı Türker, P. (2019). The effects of different programming trainings on the computational thinking skills. *Interactive Learning Environments*. https://doi.org/10.1080/10494820.2019.1635495

PS31 Qin, H. (2009). Teaching computational thinking through bioinformatics to biology students. *SIGCSE Bulletin Inroads*, Vol. 41, pp. 188–191. https://doi.org/10.1145/1539024.1508932

PS32 Rodríguez-García J.D., Moreno-León J., Román-González M., Robles G.,"LearningML: A tool to foster computational thinking skills through practical artificial intelligence projects",2020,"Revista de Educacion a Distancia","10.6018/RED.410121","https://www.scopus.com/inward/record.uri?eid=2-s2.0-85085603174&doi=10.6018%2fRED.410121&partnerID=40&md5=8629359de325467d947de6634fd4b859"

PS33 Romero M, Lepage A, Lille B. Computational thinking development through creative programming in higher education. International Journal of Educational Technology in Higher Education. 2017 Dec;14(42): 1-15

PS34 Rubinstein, A., & Chor, B. (2014). Computational Thinking in Life Science Education. *PLoS Computational Biology*, Vol. 10. https://doi.org/10.1371/journal.pcbi.1003897

PS35 Shih, H., Jackson, J. M., Hawkins-Wilson, C. L., & Yuan, P.-C. (2015). Promoting computational thinking skills in an emergency management class with MIT app inventor. *Computers in Education Journal*, Vol. 6, pp. 82–91. Retrieved from https://www.scopus.com/inward/record.uri?eid=2-s2.0-85052799029&partnerID=40&md5=df2b3b5a13cfeb93ab789a443cd5d7dd

PS36 Sondakh D.E., Osman K., Zainudin S.,"A proposal for holistic assessment of computational thinking for undergraduate: Content validity",2020,"European Journal of Educational Research","10.12973/eu-jer.9.1.33","https://www.scopus.com/inward/record.uri?eid=2-s2.0-85082097163&doi=10.12973%2feu-jer.9.1.33&partnerID=40&md5=54d1017f4af582d22b1edc9970ecd68f"

PS37 Taylor, N. G., Moore, J., Visser, M., & Drouillard, C. (2018). Incorporating computational thinking into library graduate course goals and objectives. *School Library Research*, Vol. 21. Retrieved from https://www.scopus.com/inward/record.uri?eid=2-s2.0-85053260681&partnerID=40&md5=15815233f96a5912185346fe719f6496

PS38 Wu, B., Hu, Y., Ruis, A. R., & Wang, M. (2019). Analysing computational thinking in collaborative programming: A quantitative ethnography approach. *Journal of Computer Assisted Learning*, Vol. 35, pp. 421–434. https://doi.org/10.1111/jcal.12348

PS39 Yuen, T. T., & Robbins, K. A. (2014). A qualitative study of students' computational thinking skills in a data-driven computing class. *ACM Transactions on Computing Education*, Vol. 14. https://doi.org/10.1145/2676660

PS40 Zha, S., Jin, Y., Moore, Gaston J. (2020). Hopscotch into Coding: Introducing Pre-Service Teachers Computational Thinking*. TechTrends,* 64, 17–28. https://doi.org/10.1007/s11528-019-00423-0

PS41 Zha S., Jin Y., Moore P., Gaston J. (2020). A cross-institutional investigation of a flipped module on preservice teachers' interest in teaching computational thinking (2020), *Journal of Digital Learning in Teacher Education.* https://doi.org/10.1080/21532974.2019.1693941

References

Adler, R. F., & Kim, H. (2018). Enhancing future K-8 teachers' CT skills through modeling and simulations. *Education and Information Technologies*, *23*, 1501–1514. https://doi.org/10.1007/s10639-017-9675-1

Alfayez, A. A., & Lambert, J. (2019). Exploring Saudi Computer Science Teachers' Conceptual Mastery Level of CT Skills. *Computers in the Schools*, *36*, 143–166. https://doi.org/10.1080/07380569.2019.1639593

Angeli, C., Voogt, J., Fluck, A., Webb, M., Cox, M., Malyn-Smith, J., & Zagami, J. (2016). A K-6 CT curriculum framework: Implications for teacher knowledge. *Educational Technology and Society*, *19*, 47–57. https:// doi.org/10.1016/j.chb.2019.106185

Angeli, Charoula, & Giannakos, M. (2020). Computational thinking education: Issues and challenges. *Computers in Human Behaviour*, *105*. https://doi.org/10.1016/j.chb.2019.106185

Atmatzidou, S., & Demetriadis, S. (2016). Advancing students' CT skills through educational robotics: A study on age and gender relevant differences. *Robotics and Autonomous Systems*, 75, 661–670. https://doi.org/10.1016/j.robot.2015.10.008

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? In *ACM Inroads*, *2*(1), 48–54. https://doi.org/10.1145/1929887.1929905

Bower, M., Wood, L. N., Lai, J. W. M., Howe, C., & Lister, R. (2017). Improving the CT pedagogical capabilities of school teachers. *Australian Journal of Teacher Education*, *42*(3), 53–72. https://doi.org/10.14221/ajte.2017v42n3.4

Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of CT. *Annual American Educational Research Association Meeting, Vancouver, BC, Canada*, 1–25. Retrieved from http://web.media.mit.edu/~kbrennan/files/Brennan_Resnick_AERA2012_CT.pdf

Bui, L. D., Kim, Y. G., Ho, W., Ho, H. T. T., & Pham, N. K. (2018). Developing WebQuest 2.0 model for promoting CT skill. *International Journal of Engineering and Technology (UAE)*, *7*(2), 140–144. https://doi.org/10.14419/ijet.v7i2.29.13304

Buitrago Flórez, F., Casallas, R., Hernández, M., Reyes, A., Restrepo, S., & Danies, G. (2017). Changing a Generation's Way of Thinking: Teaching CT Through Programming. *Review of Educational Research*, *87*, 834–860. https://doi.org/10.3102/0034654317710096

Cachero, C., Barra, P., Melia, S., & Lopez, O. (2020). Impact of Programming Exposure on the Development of Computational Thinking Capabilities: An Empirical Study. *IEEE ACCESS*, *8*, 72316–72325. https://doi.org/10.1109/ACCESS.2020.2987254

Cetin, I. (2016). Preservice Teachers' Introduction to Computing: Exploring Utilization of Scratch. *Journal of Educational Computing Research*, *54*, 997–1021. https://doi.org/10.1177/0735633116642774

Chao, P. Y. (2016). Exploring students' computational practice, design and performance of problem-solving through a visual programming environment. *Computers and Education*, *95*, 202–215. https://doi.org/10.1016/j.compedu.2016.01.010

Choi, S.-Y. (2019). Development of an instructional model based on constructivism for fostering CT. *International Journal of Innovative Technology and Exploring Engineering*, *8*, 381–385. Retrieved from https://www.scopus.com/inward/record.uri?eid=2-s2.0-85064633190&partnerID=40&md5=9c6da0548c789dfbb24134edc2cbfdc7

Clark, D. B., & Sengupta, P. (2019). Reconceptualizing games for integrating CT and science as practice: collaborative agent-based disciplinarily-integrated games. *Interactive Learning Environments*, *28*(3), 328-346. https://doi.org/10.1080/10494820.2019.1636071

Cooper, S., Grover, S., Guzdial, M., & Simon, B. (2014). Education: A future for computing education research. *Communications of the ACM*, *57*(11), 34–36. https://doi.org/10.1145/2668899

CSTA & ISTE. (2011). Operational definition of computational thinking. Retrieved from https://www.iste.org/explore/Solutions/Computational-thinking-for-all.

Cutumisu, M., & Guo, Q. (2019). Using Topic Modeling to Extract Pre-Service Teachers' Understandings of CT From Their Coding Reflections. 62(4), *IEEE Transactions on Education*, *62*(4), 325-332. https://doi.org/10.1109/TE.2019.2925253

Czerkawski, B. C., & Lyman, E. W. (2015). Exploring Issues About CT in Higher Education. *TechTrends*, *59*, 57–65. https://doi.org/10.1007/s11528-015-0840-3

Denning, P. J. (2017). Remaining trouble spots with computational thinking. Communications of the ACM, 60(6), 33–39. https://doi.org/10.1145/2998438 Dolgopolovas, V., Dagien˙ e, V., Jasut˙ e, E., & Jevsikova, T. (2019). Design science research for computational thinking in constructionist education: A pragmatist perspective. Problemos, 95, 144–159. https://doi.org/10.15388/Problemos.95.12

Denner, J., Werner, L., & Ortiz, E. (2012). Computer games created by middle school girls: Can they be used to measure understanding of computer science concepts? *Computers and Education*, *58*(1), 240–249. https://doi.org/10.1016/j.compedu.2011.08.006

Dolgopolovas, V., & Jevsikova, T. (2015). On Evaluation of CT of software engineering novice students. Proceedings of The IFIP TC3 Working Conference "A New Culture of Learning: Computing and next Generations" At: Vilnius, *4*(2), 105–112. https://doi.org/10.13140/RG.2.1.2855.9206

Durak, H. Y., & Saritepeci, M. (2018). Analysis of the relation between CT skills and various variables with the structural equation model. *Computers and Education*, *116*, 191–202. https://doi.org/10.1016/j.compedu.2017.09.004

Fang, A.-D., Chen, G.-L., Cai, Z.-R., Cui, L., & Harn, L. (2017). Research on blending learning flipped class model in colleges and universities based on CT - "Database principles" for example. *Eurasia Journal of Mathematics, Science and Technology Education*, *13*, 5747–5755. https://doi.org/10.12973/eurasia.2017.01024a

Fernández, J. M., Zúñiga, M. E., Rosas, M. V., & Guerrero, R. A. (2018). Experiences in Learning Problem-Solving through CT. *Journal of Computer Science and Technology*, *18*(02). https://doi.org/10.24215/16666038.18.e15

Fronza, I., El Ioini, N., & Corral, L. (2017). Teaching computational thinking using agile software engineering methods: A framework for middle schools. ACM Transactions on Computing Education, 17. https://doi.org/10.1145/3055258

Gabriele, L., Bertacchini, F., Tavernise, A., Vaca-Cárdenas, L., Pantano, P., & Bilotta, E. (2019). Lesson planning by CT skills in Italian pre-service teachers. *Informatics in Education*, *18*, 69–104. https://doi.org/10.15388/infedu.2019.04

Garneli, V., & Chorianopoulos, K. (2018). Programming video games and simulations in science education: exploring CT through code analysis. *Interactive Learning Environments*, *26*, 386–401. https://doi.org/10.1080/10494820.2017.1337036

Giannakos, M. N., Doukakis, S., Pappas, I. O., Adamopoulos, N., & Giannopoulou, P. (2015). Investigating teachers' confidence on technological pedagogical and content knowledge: an initial validation of TPACK scales in K-12 computing education context. Journal of Computers in Education, 2(1), 43–59. https://doi.org/10.1007/s40692-014-0024-8Grover, S., & Pea, R. (2013). Computational Thinking in K-12: A Review of the State of the Field. *Educational Researcher*, *42*(1), 38–43. https://doi.org/10.3102/0013189X12463051

Grover, S., Pea, R., & Cooper, S. (2015). Designing for deeper learning in a blended computer science course for middle school students. *Computer Science Education*, *25*(2), 199–237. https://doi.org/10.1080/08993408.2015.1033142

Hambrusch, S., Hoffmann, C., Korb, J. T., Haugan, M., & Hosking, A. L. (2009). A multidisciplinary approach towards CT for science majors. *SIGCSE Bulletin Inroads*, *41*, 183–187. https://doi.org/10.1145/1539024.1508931

Hou, H.-Y., Agrawal, S., & Lee, C.-F. (2020). Computational thinking training with technology for non-information undergraduates. *Thinking Skills and Creativity*, *38*. https://doi.org/10.1016/j.tsc.2020.100720

Huang, X.-P., & Leng, J. (2019). Design of database teaching model based on CT training. *International Journal of Emerging Technologies in Learning*, *14*, 52–69. https://doi.org/10.3991/ijet.v14i08.10495

Israel, M., Pearson, J. N., Tapia, T., Wherfel, Q. M., & Reese, G. (2015). Supporting all learners in school-wide CT: A cross-case qualitative analysis. *Computers and Education*, *82*, 263–279. https://doi.org/10.1016/j.compedu.2014.11.022

Jaipal-Jamani, K., & Angeli, C. (2017). Effect of Robotics on Elementary Preservice Teachers' Self-Efficacy, Science Learning, and CT. *Journal of Science Education and Technology*, *26*, 175–192. https://doi.org/10.1007/s10956-016-9663-z

Jenson, J., & Droumeva, M. (2016). Exploring media literacy and CT: A game maker curriculum study. *Electronic Journal of E-Learning*, *14*, 111–121. Retrieved from https://www.scopus.com/inward/record.uri?eid=2-s2.0-84968835202&partnerID=40&md5=b2d18e2de52058bda3b76bca4b55a25e

Jeon, Y., & Kim, T. (2017). The effects of the CT-based programming class on the computer learning attitude of non-major students in the teacher training college. *Journal of Theoretical and Applied Information Technology*, *95*(17), 4330–4339.

Kafai, Y. B. (2016). From CT to computational participation in K-12 education. *Communications of the ACM*, *59*, 26–27. https://doi.org/10.1145/2955114

Kang, Y., & Lee, K. (2020). Designing technology entrepreneurship education using computational thinking. *Education and Information Technologies*, *25*(6), 5357–5377. https://doi.org/10.1007/s10639-020-10231-2

Katai, Z. (2020). Promoting computational thinking of both sciences- and humanities-oriented students: an instructional and motivational design perspective. *ETR&D-EDUCATIONAL TECHNOLOGY RESEARCH AND DEVELOPMENT*, *68*(5), 2239–2261. https://doi.org/10.1007/s11423-020-09766-5

Kılıç, S., Gökoğlu, S., & Öztürk, M. (2020). A Valid and Reliable Scale for Developing Programming-

Oriented Computational Thinking. *Journal of Educational Computing Research*, 073563312096440. https://doi.org/10.1177/0735633120964402

Kazimoglu, C., Kiernan, M., Bacon, L., & Mackinnon, L. (2012). A Serious Game for Developing CT and Learning Introductory Computer Programming. *Procedia - Social and Behavioral Sciences*, *47*, 1991–1999. https://doi.org/10.1016/j.sbspro.2012.06.938

Korkmaz, Ö., Çakir, R., & Özden, M. Y. (2017). A validity and reliability study of the CT scales (CTS). *Computers in Human Behavior*, *72*, 558–569. https://doi.org/10.1016/j.chb.2017.01.005

Kwon, J., & Kim, J. (2018). A study on the design and effect of CT and software education. *KSII Transactions on Internet and Information Systems*, *12*, 4057–4071. https://doi.org/10.3837/tiis.2018.08.028

Lee, Y., & Cho, J. (2020). Knowledge representation for computational thinking using knowledge discovery computing. Information Technology and Management, *21*(1), 15–28. https://doi.org/10.1007/s10799-019-00299-9

Li, M., & Hou, D. (2014). Network autonomous learning based on CT. *World Transactions on Engineering and Technology Education*, *12*, 576–580. Retrieved from https://www.scopus.com/inward/record.uri?eid=2-s2.0-84916237711&partnerID=40&md5=3d31af2cb32278ebb421ef9de6f7271f

Lin, P. H., & Chen, S. Y. (2020). Design and Evaluation of a Deep Learning Recommendation Based Augmented Reality System for Teaching Programming and CT. *IEEE Access*, *8*, 45689–45699. https://doi.org/10.1109/ACCESS.2020.2977679

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. https://doi.org/10.1016/j.chb.2014.09.012

Lyon, J. A., & Magana, J. A. (2020). Computational thinking in higher education: A review of the literature. *COMPUTER APPLICATIONS IN ENGINEERING EDUCATION*, *28*(5), 1174–1189. https://doi.org/10.1002/cae.22295

Ma, J. Bin, Teng, G. F., Zhou, G. H., & Sun, C. X. (2017). Practical teaching reform on CT training for undergraduates of computer major. *Eurasia Journal of Mathematics, Science and Technology Education*, *13*(10), 7121–7130. https://doi.org/10.12973/ejmste/78738

Magana, A. J., & Silva Coutinho, G. (2017). Modeling and simulation practices for a CT-enabled engineering workforce. *Computer Applications in Engineering Education*, *25*(1), 62–78. https://doi.org/10.1002/cae.21779

Moreno León, J., Robles, G., & Román González, M. (2015). Dr. Scratch: Automatic Analysis of Scratch Projects to Assess and Foster CT. *RED: Revista de Educación a Distancia*, *46*.

Mouza, C., Yang, H., Pan, Y.-C., Yilmaz Ozden, S., & Pollock, L. (2017). Resetting educational technology coursework for pre-service teachers: A CT approach to the development of technological pedagogical content knowledge (TPACK). *Australasian Journal of Educational Technology*, *33*, 61–76. https://doi.org/10.14742/ajet.3521

Page, R., & Gamboa, R. (2013). How Computers Work: CT for Everyone. *Electronic Proceedings in Theoretical Computer Science*, *106*, 1–19. https://doi.org/10.4204/eptcs.106.1

Pala, F. K., & Mıhçı Türker, P. (2019). The effects of different programming trainings on the CT skills. *Interactive Learning Environments*. https://doi.org/10.1080/10494820.2019.1635495

Papert, S. (1980). Mindstorms: Children, computers, and powerful ideas. New York: Basic Books.

Petersen, K., Feldt, R., Mujtaba, S., & Mattsson, M. (2008). Systematic mapping studies in software engineering. *12th International Conference on Evaluation and Assessment in Software Engineering, EASE 2008*, 1–10.

Qin, H. (2009). Teaching CT through bioinformatics to biology students. *SIGCSE Bulletin Inroads*, *41*, 188–191. https://doi.org/10.1145/1539024.1508932

Repenning, A., Grover, R., Gutierrez, K., Repenning, N., Webb, D. C., Koh, K. H., … Gluck, F. (2015). Scalable Game Design. *ACM Transactions on Computing Education*, *15*(2), 1–31. https://doi.org/10.1145/2700517

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., … Kafai, Y. (2009). Scratch: Programming for all. *Communications of the ACM*, *52*(11), 60–67. https://doi.org/10.1145/1592761.1592779

Rodríguez-García, J. D., Moreno-León, J., Román-González, M., & Robles, G. (2020). LearningML: A tool to foster computational thinking skills through practical artificial intelligence projects. *Revista de Educacion a Distancia*, *20*(63). https://doi.org/10.6018/RED.410121

Romero, M., Lepage, A., & Lille, B. (2017). CT development through creative programming in higher education. *International Journal of Educational Technology in Higher Education*, *14*(42). https://doi.org/10.1186/s41239-017-0080-z

Rubinstein, A., & Chor, B. (2014). CT in Life Science Education. *PLoS Computational Biology*, *10*. https://doi.org/10.1371/journal.pcbi.1003897

Silva, E. (2009). Measuring Skills for 21st-Century Learning. *Phi Delta Kappan*, *90*(9), 630-634.

Selby, C. (2013). Computational Thinking : The Developing Definition. *ITiCSE Conference 2013*, 5–8.

Shih, H., Jackson, J. M., Hawkins-Wilson, C. L., & Yuan, P.-C. (2015). Promoting CT skills in an emergency management class with MIT app inventor. *Computers in Education Journal*, *6,* 82–91. Retrieved from https://www.scopus.com/inward/record.uri?eid=2-s2.0-85052799029&partnerID=40&md5=df2b3b5a13cfeb93ab789a443cd5d7dd

Shute, V. J., Sun, C., & Asbell-Clarke, J. (2017). Demystifying CT. *Educational Research Review*, *22*, 142–158. https://doi.org/10.1016/j.edurev.2017.09.003

Sondakh, D. E., Osman, K., & Zainudin, S. (2020). A proposal for holistic assessment of computational thinking for undergraduate: Content validity. *European Journal of Educational Research*, *9*(1), 33–50. https://doi.org/10.12973/eu-jer.9.1.33

Souza, M. R. d. A., Veado, L., Moreira, R. T., Figueiredo, E., & Costa, H. (2018). A systematic mapping study on game-related methods for software engineering education. *Information and Software Technology*, *95*, 201–218. https://doi.org/10.1016/j.infsof.2017.09.014

Tang, X., Yin, Y., Lin, Q., Hadad, R., & Zhai, X. (2020). Assessing computational thinking: A systematic review of empirical studies. *Computers and Education*, *148*. https://doi.org/10.1016/j.compedu.2019.103798

Taylor, N. G., Moore, J., Visser, M., & Drouillard, C. (2018). Incorporating CT into library graduate course goals and objectives. *School Library Research*, *21*. Retrieved from https://www.scopus.com/inward/record.uri?eid=2-s2.0-85053260681&partnerID=40&md5=15815233f96a5912185346fe719f6496

Tikva, C., & Tambouris, E. (2021). Mapping Computational Thinking through Programming in K-12 Education: A Conceptual Model based on a Systematic Literature Review. *Computers & Education*, *162*. https://doi.org/https://doi.org/10.1016/j.compedu.2020.104083

Weintrop, D., Holbert, N., Horn, M. S., & Wilensky, U. (2016). CT in constructionist video games. *International Journal of Game-Based Learning*, *6*, 1–17. https://doi.org/10.4018/IJGBL.2016010101

Werner, L., Denner, J., Campe, S., & Kawamoto, D. C. (2012). The fairy performance assessment: Measuring computational thinking in middle school. SIGCSE'12 - proceedings of the 43rd ACM technical symposium on computer science education. https://doi.org/10.1145/2157136.2157200

Wu, B., Hu, Y., Ruis, A. R., & Wang, M. (2019). Analysing CT in collaborative programming: A quantitative ethnography approach. *Journal of Computer Assisted Learning*, *35*, 421–434. https://doi.org/10.1111/jcal.12348

Xing, W. (2019). Large-scale path modeling of remixing to CT. *Interactive Learning Environments*. https://doi.org/10.1080/10494820.2019.1573199

Yadav, A., Stephenson, C., & Hong, H. (2017). CT for teacher education. *Communications of the ACM*, *60*, 55–62. https://doi.org/10.1145/2994591

Voogt, J., Fisser, P., Good, J., Mishra, P., & Yadav, A. (2015). Computational thinking in compulsory education: Towards an agenda for research and practice. In *Education and Information Technologies, 20*(4), 715–728). https://doi.org/10.1007/s10639-015-9412-6

Yuen, T. T., & Robbins, K. A. (2014). A qualitative study of students' CT skills in a data-driven computing class. *ACM Transactions on Computing Education*, *14*. https://doi.org/10.1145/2676660

Wing, J. M. (2006). Computational thinking. In *Communications of the ACM 49*(3), 33–35. https://doi.org/10.1145/1118178.1118215

Zha, S., Jin, Y., Moore, P., & Gaston, J. (2020a). A cross-institutional investigation of a flipped module on preservice teachers' interest in teaching computational thinking. *Journal of Digital Learning in Teacher Education*, *36*(1), 32–45. https://doi.org/10.1080/21532974.2019.1693941

Zha, S., Jin, Y., Moore, P., & Gaston, J. (2020b). Hopscotch into Coding: Introducing Pre-Service Teachers Computational Thinking. *TechTrends*, *64*(1), 17–28. https://doi.org/10.1007/s11528-019-00423-0

Zhong, B., Wang, Q., Chen, J., & Li, Y. (2016). An exploration of three-dimensional integrated assessment for computational thinking. Journal of Educational Computing Research, 53. https://doi.org/10.1177/0735633115608444