Computational Thinking Interventions in Higher Education

A Scoping Literature Review of Interventions Used to Teach Computational Thinking

Imke de Jong
i.dejong1@uu.nl
Utrecht University
Utrecht, The Netherlands

Johan Jeuring
j.t.jeuring@uu.nl
Utrecht University
Utrecht, The Netherlands

ABSTRACT

Computational Thinking is seen as a crucial skill in an increasingly digital society. Researchers and educators in higher education therefore aim to improve the Computational Thinking (CT) skills of students using appropriate interventions. However, there is currently no overview of interventions used to teach CT and how effective they are. With this scoping literature review, we provide such an overview by identifying articles that discuss interventions used to teach CT in higher education. We identify the teaching approaches used in these interventions, and discuss their effectiveness and how this is assessed. Furthermore, we look at the use of adaptive interventions. Our search of three academic databases (Scopus, ACM and ERIC) resulted in 1839 articles. After screening, 49 articles remained. A detailed examination of the interventions discussed in these articles showed that CT is still often taught through programming assignments. The interventions are evaluated in a myriad of ways, making it difficult to compare the effectiveness of interventions. We therefore suggest making use of more standardized instruments to evaluate the effectiveness. Finally, although scaffolding is applied, interventions are not often adapted to the actual proficiency level of a student.

CCS CONCEPTS

• **Social and professional topics** → **Computer science education**; **Information science education**; **Computational thinking**.

KEYWORDS

Computational thinking, higher education, interventions, effectiveness

ACM Reference Format:
Imke de Jong and Johan Jeuring. 2020. Computational Thinking Interventions in Higher Education: A Scoping Literature Review of Interventions Used to Teach Computational Thinking. In *Koli Calling '20: Proceedings of the 20th Koli Calling International Conference on Computing Education Research (Koli Calling '20), November 19–22, 2020, Koli, Finland.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3428029.3428055

1 INTRODUCTION

Computational Thinking (CT) skills are viewed as a set of essential problem-solving skills that individuals need in the 21st century [18, 70]. Computing technologies have become an integral part of society. They are used to tackle complex problems, and to collect and visualize data. Being able to think computationally is essential to optimally make use of these technologies. The underlying skills of CT (such as decomposition and algorithmic thinking) enable problem solvers to optimally recognize and utilize the possibilities offered by computational tools. Organizations are therefore looking for graduates who can apply these skills [55]. In turn, educators and researchers aim to find suitable ways to help students develop these problem-solving skills. In the Netherlands, improving students' computational skills is therefore one of the focus areas for universities, applied universities, and the Ministry of Education, Culture and Science [68].

Different interventions and approaches to teach CT are being developed. However, a comprehensive overview of interventions and their effectiveness is currently lacking. Few overview studies are available that present evidence for the effectiveness of interventions used to teach CT in higher education; literature reviews often focus on primary and secondary education [25, 62]. Recently, Agbo et al. [1] did provide an overview of interventions in higher education. However, this study does not offer an inventory of CT interventions aimed at a broad audience, but focuses on programming education. Because CT is seen as a skill set that is important for a broader population of students, not just computer scientists, a literature study with a broader focus is needed.

With this scoping literature review we aim to fill this gap by providing an overview of experiments with CT interventions in diverse higher education contexts. We show how their effectiveness is assessed and what results are obtained with these interventions. This overview study gives lecturers and researchers more insight into tested ways of incorporating CT. Furthermore, we are also interested in examining whether interventions for teaching CT are adapted to students' proficiency levels to optimally support the learner. For related domains like problem solving [9, 29] and programming [54], it has been noted that scaffolding and adapting interventions to students' proficiency levels is beneficial, and this may also improve the development of CT skills.

The research questions (RQs) underlying this study are therefore:

(1) What interventions are used to teach Computational Thinking in higher education?
(2) How is the effectiveness of Computational Thinking interventions in higher education assessed?
(3) How effective are Computational Thinking interventions in higher education?

(4) To what extent, and if so in what way, are Computational Thinking interventions in higher education adapted to students' proficiency levels?

In this article, we will first give a brief introduction of CT and the underlying skills, and then discuss the method of our scoping review and its results.

2 COMPUTATIONAL THINKING

If we want to help students develop their Computational Thinking skills, we first need to define what those skills are. But, although being able to think computationally is deemed important, what this entails exactly and how to teach it are both still subject to debate.

2.1 Defining Computational Thinking

Different definitions for CT have been proposed over the years [7, 56, 62]. One of the first definitions was proposed by Wing [71], who defined CT as "[...] the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent". Here, the processing agent could be a computer or a human. In other definitions, the computer plays a more central role, for example in the more recent definition of Denning and Tedre [21] who define CT as "the mental skills and practices for designing computations that get computers to do jobs for us, and explaining and interpreting the world as a complex of information processes". In general, researchers and educators agree that CT leads to new ways of thinking about problems and their solutions, by providing problem solvers with additional tools (e.g. computing devices) and strategies or skills (e.g. algorithms, decomposition) to solve complex problems.

Besides the definition, the skills needed to make someone a Computational Thinker are also subject to debate. Hsu et al. [25] have looked at different studies and identified 19 different competences involved in CT, among which abstraction, data analysis, pattern recognition, algorithm design, debugging and error detection, and problem solving. Selby and Woollard [56] combined the results of different studies and conclude that CT consists of the ability to think in abstraction, decomposition, algorithms, evaluation, and generalization. Meanwhile, Shute et al. [62] identify six general facets: decomposition, abstraction, algorithms, debugging, iteration, and generalization. The Computer Science Teachers Association (CSTA) and the International Society for Technology in Education (ISTE) propose that CT also encompasses dispositions and predispositions like confidently dealing with complexity, being able to handle open-ended questions, and perseverance when solving difficult problems [18]. In line with this, a distinction is sometimes made between knowing computational concepts (e.g. loops, operators, conditionals), applying computational practices (e.g. abstracting and modularizing, testing and debugging) and having computational perspectives (e.g. questioning, expressing) [11].

2.2 Teaching Computational Thinking

Looking at the competences or skills that are part of CT and the proposed definitions, it seems CT involves more than instructing a computer by means of a programming language. Among other things, it involves abstracting the relevant information from a problem, knowing which tools to use to solve a problem, and creating solutions that are reusable [62]. However, an analysis of the CT literature by Hsu et al. [25] shows that before 2018 the most important teaching instrument used to teach children and young adults was still a programming language, even though unplugged alternatives (like Bebras tasks [6]) have also been developed. One can question whether the more generic problem solving skills involved in CT can be taught through programming only. The literature on this topic shows conflicting results. Some researchers who investigated whether programming assignments can lead to increased cognitive skills like reasoning and planning found no or only limited increase [34]. On the other hand, a meta-analysis by Liao and Bright [39] showed programming education can increase skills like reasoning, logical thinking, planning, and more general problem solving. It is, however, questioned whether this is the most efficient or effective way to teach these skills. Thus, it seems preferable to not solely rely on programming assignments to develop CT skills. With this literature review, we aim to uncover what approaches are currently used in higher education, and provide an overview of the effectiveness of both programming and non-programming interventions.

2.3 Adaptive interventions for Computational Thinking

We also want to investigate if, and if so how, interventions are adapted to the skill level of the students. As mentioned above, for children in the K-12 age range, a set of different non-programming assignments has been created for the so-called Bebras challenges [6]. The Bebras tasks consist of different unplugged assignments through which children and young adults can learn specific CT skills. Different Bebras have been developed for different age groups or school grades. The developers of these assignments are thereby catering for the expected proficiency level of students of particular age groups. These assignments do not take the individual, actual skill level of the student into account. They are rather static; every student in a class receives the same assignment. However, research has shown that the current proficiency level of a student influences what teaching method is most appropriate [29]. Scaffolding or adapting the teaching method to a student's proficiency has positive learning effects in programming education [42]. Matching the teaching method to the proficiency level of the student may therefore also be important when teaching CT. Tedre and Denning [65], for example, highlight the importance of distinguishing different proficiency levels while teaching and assessing CT skills. In recent years, some researchers have taken concepts like scaffolding and proficiency levels and applied them to CT. Pollock et al. [50] created a rubric for higher education that distinguishes proficiency levels. Also, Basu et al. [5] have used the concept of scaffolding to create a framework with which to assist students in accomplishing CT assignments. Finally, Basawapatna et al. [4] applied scaffolding techniques and found that they help to keep students engaged in learning. We therefore expect it would be beneficial for the students if their current skill level is taken into account when CT interventions are presented, and we are interested to find out if this technique is currently being used. This is one of the elements we investigate in this literature review.

3 METHOD

This review is set-up using the methodology proposed by the PRISMA extension for scoping literature reviews [66], and the Joanna Briggs Institute review guidelines [48]. The next sections describe the setup of the review.

3.1 Information sources

In this scoping review, we used a number of different databases: one focused on the domain of education, one focused on Computer Science and one generic academic database. Doing so, we aim to provide a comprehensive overview of the interventions currently used to teach Computational Thinking. The databases used in this review are:

- ERIC (journal database aimed at research into education - https://eric.ed.gov/)
- SCOPUS (Elsevier journal database - https://www.scopus.com)
- ACM digital library (article database aimed at research into computing - https://dl.acm.org/)

3.2 Inclusion criteria

The articles identified through the literature search were screened and only selected if they match the following criteria:

(1) *The year of publication is 2006 or later.* Although the individual CT skills have been the focus of research before 2006, the article of Jeannette Wing published in this year [70] (re)introduced CT as an important skill set.
(2) *The interventions must be aimed at teaching CT.* Articles were only included when the authors indicate that the interventions used in the article are aimed at teaching CT. This means articles discussing general programming education were excluded.
(3) *The research domain is higher education.* We aim to identify interventions that can be used to teach CT to undergraduate students. These students have received multiple years of elementary and high school education, which means that interventions for the K-12 levels will probably be too simple.
(4) *The effect of the intervention should be measured.* We aim to present an overview of interventions and their effectiveness in influencing CT skills. Therefore, the article must discuss how the effect(iveness) of the intervention was measured. The only exceptions to this criterion are papers describing interventions adapted to the proficiency level of the student. These papers were always included due to our interest in adaptive CT interventions.
(5) *The paper must be published in a peer-reviewed journal or peer-reviewed proceedings of a conference.*
(6) *Only full papers, i.e. no panel announcements, work-in-progress papers, workshop overviews etc.*
(7) *The paper must be written in English.*

3.3 Search method & search strategy

We selected the papers in three steps. Before the start of the search, a number of available articles were analyzed to determine which keywords should be used to find articles discussing CT interventions in higher education. This resulted in the following keywords:

Table 1: Query result count

Database	Number of results
ERIC	12
SCOPUS	391
ACM	1811
Total	*2214*

- Computational Thinking, AND
- K-16 OR higher education, AND
- interventions OR tool OR intervention tool OR teaching method OR assignment OR activity OR activities OR learning strategy OR learning strategies OR instructional strategy OR teaching tool OR learning method OR exercise

As a second step we used these search terms and the inclusion criteria (discussed in Section 3.2) to devise a search strategy (i.e. search queries) for each database. The number of search results per database can be found in Table 1. A complete overview of the papers found is available from the first author upon request. After removal of duplicates and an initial title screening (e.g. titles referencing only K-12 were excluded), one of the authors screened the resulting 1839 articles using the inclusion criteria. First, the relevance was determined by examining the title. If the article was likely to be relevant, or the result was inconclusive, the abstract of the article was screened. As a third step, the remaining articles were screened more extensively by reading (parts of) the papers.

3.4 Coding and extracting information

An initial coding scheme was set up to extract information on relevant categories from the articles and classify them. The first author coded the papers and used emergent coding, which means that additional sub-codes were created based on the contents of the articles. The academic discipline, higher education level (undergraduate, graduate or post-graduate) and the CT definition (specifically mentioning CT skills or generic) were coded to get some background information on the context of CT research in higher education. Then, we coded specific information in each article to answer our research questions. The final coding scheme for the RQs can be found in Table 2.

For RQ1 we investigated the intervention type (e.g. assignments, lectures with short activities), the learning objectives, the intervention length, and the scope of the intervention (e.g. a course or workshop, a curriculum, stand-alone assignments). If the intervention consisted of assignments, we examined the type of assignments used. We made a distinction between programming assignments (both text- and bock-based) and other, non-programming assignments. The latter can be unplugged assignments or assignments using computers but without the need to program.

To answer RQ2 and RQ3, we coded two aspects of the evaluation of the interventions: (1) how evidence of the effectiveness was collected and (2) what kind of evidence the evaluation focused on. We did not use sub-codes to code the outcomes of the evaluations.

Finally, for RQ4 we examined if adaptation was applied and how the student's level was assessed (the adaptation strategy). If applicable, we also coded the form of adaptation to understand what the adaptation entailed.

To confirm the reliability of the coding, the second author coded 5 articles (10% of the selected papers) as well. Because one-to-many or non-exclusive coding was applied (more than one sub-code of a category could apply to a paper), inter-reliability measures like Cohen's Kappa and Krippendorff could not be used. We therefore used fuzzy Kappa as a measure of coding reliability [33]. The results showed substantial agreement (fuzzy κ = 0.66). Out of the 60 items, on 37 items both coders completely agreed, for 7 items there was partial agreement, and on 16 items the coders disagreed. After discussion between the authors, the agreement with the coding of the first author was raised to almost perfect agreement (fuzzy κ = 0.93), with 5 out of 60 codes needing (partial) re-adjustment. It was therefore concluded that the coding of the first author was sufficiently reliable.

4 RESULTS

The screening and eligibility checks discussed in section 3.3 resulted in the selection of 49 papers. Details of the screening process can be found in Figure 1. Table 3 provides an overview of the final selection of papers.

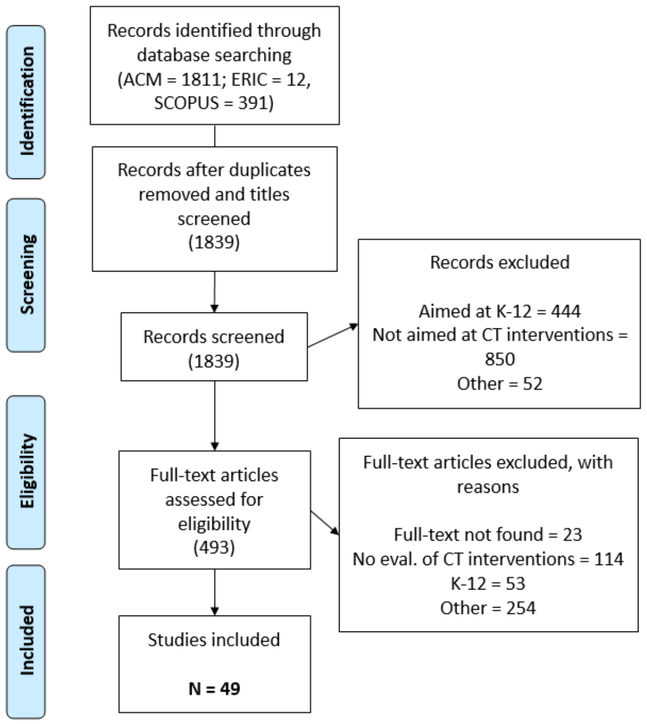
Identification
Screening
Eligibility
Included
Records identified through database searching (ACM = 1811; ERIC = 12, SCOPUS = 391)
Records after duplicates removed and titles screened (1839)
Records screened (1839)
Records excluded
Aimed at K-12 = 444
Not aimed at CT interventions = 850
Other = 52
Full-text articles assessed for eligibility (493)
Full-text articles excluded, with reasons
Full-text not found = 23
No eval. of CT interventions = 114
K-12 = 53
Other = 254
Studies included
N = 49

Figure 1: PRISMA diagram [66] showing the evolution of the search

In this section we will first present the types of CT interventions used in higher education (RQ 1) and then discuss the effectiveness of these interventions and how this is assessed (RQ 2 and 3). Finally, we will elaborate on the use of adaptive interventions (RQ 4).

Table 2: Coding scheme

RQ	Category	Sub-codes
RQ1	Intervention scope	Stand-alone assignment
		Course
		Curriculum
		Other
		Not specified
RQ1	Intervention length	Less than a day
		One day to a week
		More than a week to a month
		More than a month to a year
		More than a year
		Not specified
RQ1	Intervention type	Assignments
		Lectures (with activities)
		Lectures and assignments
		Other
		Not specified
RQ1	Assignment type	Programming
		Other
		Not specified
RQ1	Learning objective	General CT skills
		Misc. computing skills
		Programming skills
		Other
		Not specified
RQ2	Evaluation - how	Anecdotal
		Empirical - course grades
		Empirical - interviews
		Empirical - pre- and post-test
		Empirical - survey
		Usage, particip. & effort analytics
		Other
		Not specified
RQ2	Evaluation - what	Attitude towards CS
		Attitude towards CT
		Course grades
		CT knowledge
		CT skills
		Programming knowledge
		Programming skills
		Usage, particip. & effort analytics
		Other
		Not specified
RQ3	Evaluation - outcome	-
RQ4	Adaptation strategy	Measurement of student's level
		Self-assessment of level by student
		No adaptation
RQ4	Form of adaptation	Order of assignments
		Programming level of assignment
		Type of instruction

Table 3: Overview of papers - intervention and assignment types

Authors	Year	Intervention type	Assignment type
Alrashidi et al.[2]	2017	Assignments	Programming
Aristawati et al.[3]	2018	Assignments	Programming
Basawapatna et al.[4]	2019	Assignments	Programming
Behnke et al.[8]	2016	Not specified	Not specified, Programming
Benakli et al.[10]	2017	Assignments	Programming
Cabo and Lansiquot[12]	2016	Assignments	Other, Programming
Cai et al.[13]	2018	Lectures and assignments	Not specified
Calderon[14]	2018	Lectures and assignments	Other
Calderon et al.[15]	2020	Assignments	Other
Cetin and Andrews-Larson[16]	2016	Lectures and assignments	Programming
Choi[17]	2019	Not specified	Other, Programming
Corral[19]	2018	Lectures and assignments	Programming
Curzon et al.[20]	2014	Lectures (with activities)	
Dodero et al.[22]	2017	Assignments	Programming
Gabriele et al.[23]	2019	Lectures and assignments	Other, Programming
Gao et al.[24]	2019	Assignments	Other, Programming
Jaipal-Jamani and Angeli[26]	2017	Assignments	Programming
Jung et al.[27]	2017	Lectures and assignments	Other, Programming
Kafura et al.[28]	2015	Lectures and assignments	Other, Programming
Kazimoglu et al.[30]	2012	Not specified	Programming
Kim et al.[31]	2013	Lectures and assignments	Other
Kim and Kim[32]	2017	Lectures and assignments	Programming
Kwon and Sohn[35]	2018	Lectures and assignments	Other, Programming
L'Heureux et al.[41]	2012	Assignments	Other, Programming
Lamprou and Repenning[36]	2018	Lectures and assignments	Programming
Lee and Lovvorn[38]	2016	Assignments	Programming
Lee et al.[37]	2019	Assignments	Programming
Libeskind-Hadas and Bush[40]	2013	Lectures and assignments	Other, Programming
Meysenburg et al.[43]	2018	Lectures and assignments	Programming
Miller et al.[44]	2014	Assignments	Other
Pala and Türker[45]	2019	Lectures and assignments	Programming
Pasquinelli and Joines[46]	2011	Assignments	Other, Programming
Peteranetz et al.[47]	2019	Lectures and assignments	Not specified, Other
Philip et al.[49]	2013	Lectures and assignments	Other
Pollock et al.[50]	2019	Assignments, Lectures and assignments, Other	Other, Programming
Pulimood et al.[51]	2016	Assignments	Other, Programming
Qin[52]	2009	Lectures and assignments	Other, Programming
Senske[57]	2017	Not specified	Other, Programming
Settle[58]	2011	Lectures and assignments	Other
Shanmugam et al.[59]	2019	Assignments	Other, Programming
Shell et al.[60]	2017	Assignments	Other
Shih et al.[61]	2014	Assignments	Programming
Socher et al.[63]	2019	Assignments	Other, Programming
Sung and Jeong[64]	2019	Lectures and assignments	Programming
Van Dyne and Braun[67]	2014	Lectures and assignments	Not specified, Programming
Walden et al.[69]	2013	Not specified	
Wu et al.[72]	2019	Assignments	Programming
Yadav et al.[74]	2011	Lectures (with activities)	
Yadav et al.[73]	2014	Lectures (with activities)	

Table 4: Overview of academic disciplines

Academic discipline	Papers
Architecture	[57]
Biochemistry	[43]
Biology	[43],[40]
Business Administration	[59]
Communication	[69]
Computer Science	[60],[69],[2],[32],[44],[10],[43] [14],[3],[51],[30],[63],[15],[37]
Teacher Education	[73],[74],[20],[32],[36],[26],[35] [22],[16],[31],[72],[64],[45],[23]
Emergency management	[61]
Engineering	[10],[14],[38],[49],[15], [46]
Fine arts	[19]
General	[8],[67],[44],[13],[24],[12],[28] [58],[27],[17]
Information Systems	[69],[41]
Journalism	[51]
Life Sciences	[52]
Mathematics	[50],[10]
Music	[50]
Not specified	[4],[47]
Sociology	[50]
Tourism	[63]

4.1 CT interventions in higher education

As can be seen in Table 4, our selection of papers discuss CT interventions applied in a range of different academic disciplines. A total of 18 different disciplines are mentioned. Most interventions are aimed at Computer Science (N = 14) or pre- or post-service Teacher Education (N = 14). In 10 articles a generic intervention not aimed at a specific target audience (often described as CS for non-majors) is presented.

The selected papers mostly discuss CT interventions targeted at undergraduate students (N = 39), and often describe the design or redesign of a course (N = 32) or a stand-alone assignment (N = 13). We also examined the length of the interventions. Although 17 articles did not explicitly specify this, most articles (N = 22) indicated the length of the intervention was between a month and a year. Usually those interventions were applied for the duration of one semester, which matches our finding that the interventions are often course (re)designs. We encountered only two studies that describe interventions in which a curriculum for CT (i.e. more than a single course) is addressed, and no interventions that lasted longer than one year.

The learning objectives of the interventions vary, although most papers (N = 30) indicate that developing one or more CT skills was indeed the objective. Learning objectives also include programming (N = 14) and other or miscellaneous (computer) skills (N = 19) like using Microsoft Office applications [13, 46] and Git [43], or computer literacy [69]. Fourteen papers indicate a combination of these learning objectives. Three papers did not specify the learning objectives of the interventions. We also examined the definitions of CT used in the papers. Here, we discovered that although the specific definition used varies, most papers (N = 34) apply or build upon generally agreed skills (e.g. abstraction, decomposition, algorithms). Ten of the papers provide a more generic definition, for example "a kind of logical thinking" [31], "an outlook and set of skills that will help our students use computing well today and help them adapt to changes in computing as future professionals" [57], and "Computational thinking involves solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science." [67].

One of the main goals of this scoping review is to determine the types of interventions used. As can be seen in the overview in Table 2, while classifying the papers we noticed a distinction between programming and non-programming assignments as (part of) interventions. In our selection of papers, 18 papers use only programming assignments and 7 papers use only non-programming assignments (see Table 3). Also, 16 studies use a combination of programming and non-programming assignments.

Taking a closer look at the programming assignments, a variety of languages and tools are used to develop CT skills. In some studies, students directly write code in Python [2, 40, 43, 45, 50], R [10], HTML/JavaScript [19], C++ [45, 72], SQL [52], VBA [46], and Java [17]. Others interventions use graphical or block-based programming environments like Scratch [23, 32, 36, 64], MIT App Inventor [41, 59, 61], AgentCubes [4, 36], Alice [12, 41], VEDIL [22], Blockly [24], or Program your Robot [30]. Also, robotics kits like LEGO WeDo [26], LEGO Mindstorms [3], Finch [38], and Arduino [45] are used. Finally, some studies use a combination [27, 28, 57, 67], for example starting with a graphical environment and then moving to a text-based environment [67]. The artefacts being created during these assignments are very diverse, although it is not always clear from articles what exactly is being programmed. Some programming assignments focus more general on the creation of an application (e.g. [23, 59]). Others ask students to program a game [4, 38] or a robot [2, 45, 67]. More domain-specific programming assignments are also discussed, e.g. image processing [43], solving mathematical problems [10], web design for Fine Arts students [19] or an app for emergency management [61].

This diverse spectrum of tools and techniques is also visible when looking at the non-programming assignments. Different options to teach CT without programming have been used in the studies. Examples of these are using computational creativity exercises [47, 60], paper-and-pencil or unplugged activities [15, 31, 40, 59], and simulation environments [28]. Content-wise the assignments are again very different. Students are asked to analyze datasets and visualize their findings in graphs [50], or practice algorithmic thinking through Marching Orders Activities [15].

Combinations of programming and non-programming assignments are also used. In some papers, interventions are described that involved giving different types of interventions to students of academic disciplines working together on shared projects. For example, the Computer Science majors would be asked to program, while the Journalism majors [51] or Tourism majors [63] were given non-programming assignments like gathering user requirements.

4.2 Determining the effectiveness of interventions

A myriad of interventions has been developed to teach CT, and the same holds for the ways in which the effectiveness of interventions is determined. Table 2 shows that we have identified at least 7 different evaluation methods. Empirical methods like course grades, pre- and post-tests of knowledge or skills, surveys and interviews are used. Also, the participation or effort put in by the students is examined, and anecdotal evidence is gathered from researchers or teachers.

The implementation of these methods varies a lot between the studies. As an example, researchers using pre- and post-tests often devise their own set of questions to determine the effectiveness of the interventions. Out of the 23 pre- and post-tests, 22 different measurements instruments are used, details of which are not always included in the papers. Only one of the measurement instruments was used twice. This is probably because those papers had an author in common and examined the same kind of intervention [44, 60]

The studies not only differ in how, but also in what they measure. Though the selected papers all present interventions that aim to develop students' CT skills, when evaluating the results, the focus is not always on those CT skills. We refer to Table 2 again for the different measurements used. Half of the studies (N = 25) use two or more of these measurement types to examine the effect of their interventions. However, in only 19 out of the 49 papers the students' actual CT skills are measured in some way. Other measurements specifically aimed at CT include knowledge about CT (N = 14) or attitude towards CT (N = 8). And even though most researchers seem to agree that programming is not equivalent to CT, there are studies using programming skills or knowledge as measurement without looking at CT skills or knowledge (N = 7).

4.3 Results on effectiveness of CT interventions

For our third RQ, we examined how effective CT interventions in higher education are. Due to the different evaluation methods used, it is difficult to compare and contrast the effectiveness of interventions used in current studies. However, below we present some of the findings regarding CT skills, knowledge and the attitude towards CT.

Most studies describe positive effects of their interventions. A number of studies evaluated the CT skills and found improvement on post-tests (both self-assessment and actual skill tests) [2, 15, 17, 32, 45, 47, 49, 51, 63, 67, 69], a relation between the number of assignments completed and an increase in CT skills [44, 60], increased awareness of using CT skills [37, 41], and evidence of CT skills through an investigation of programming concepts used in Scratch [23]. Studies also describe improved understanding of CT [13, 17, 20, 31, 47, 50, 73, 74], and conclude that the attitude towards CT was improved by the intervention [47, 57, 59, 64, 73, 74].

Less positive results are also reported. The interest in CT did not always increase after the intervention [50], nor the knowledge of CT [36]. Also, Pasquinelli and Joines [46] noted that the use of computing tools distracted students from learning the domain-specific problem solving, in their case in the domain of thermodynamics.

Some studies compared different approaches to teach CT. Kim et al. [31] compared a Paper-and-pencil Programming Strategy (PPS) with a LOGO based course, and found that the PPS increased the students' self-reported understanding of CT more than LOGO. Also, the interest in learning CS increased significantly more in the PPS-course. In a study comparing Arduino and C++ as programming tools, Pala and Türker [45] found that Arduino influenced computational thinking skills, but C++ did not.

4.4 Adapting interventions to the learner

To answer our fourth and final RQ, we examined the use of adaptive interventions. In our selection, very few studies applied interventions with some form of adaptation to the students' skill levels. Some studies did indicate that scaffolding was part of the way the course or assignment was set up. This scaffolding was fixed and the same for each student, however. Out of the 49 studies, only 3 studies [4, 8, 38] reported that the interventions were adapted to the actual current proficiency level of the student. This adaptation was always based on the self-reported proficiency level of a student. We encountered three forms of adaptation of the interventions: Behnke et al. [8] varied the order of assignments, Lee and Lovvorn [38] adjusted the programming level of the assignment, and Basawapatna et al. [4] differentiated the type of instruction students of different levels received.

5 DISCUSSION AND CONCLUSION

With this scoping review, we aim to provide insight into the interventions used to teach CT in diverse disciplines in higher education, and the ability of these interventions to actually improve students' CT skills. We searched three academic databases and screened the results using seven inclusion criteria. This resulted in the selection of 49 papers, published since 2006, in which an intervention to develop CT skills in higher education was described. We found that CT interventions are developed for students of numerous disciplines, but also for the general student population (CS for non-majors). Most of the interventions are targeted at students in Computer Science and Teacher Education.

For our first research question, we examined what kind of interventions are used to teach CT. Just like Hsu et al. [25], we conclude that programming assignments are still the most often used approach. However, we also noted the use of non-programming assignments, often in combination with the former. The assignments themselves are very diverse: both domain-specific (e.g. solve a mathematical problem using R) and more generic assignments (e.g. program a game) are used. We only encountered two studies that describe the creation of a curriculum for CT, and no interventions that lasted longer than one year. As CT is seen as a problem solving skill, and problem solving skills take time to develop [53], we hypothesize it would be useful to create longer interventions. This enables students to better develop their computational thinking skills. Also, re-examining the CT skills of students some time after the interventions may provide valuable information about the persistence of the developed skills.

Regarding methods used to evaluate the effectiveness of interventions (RQ2), we conclude that a lot of work still needs to be done to incorporate standardized measurements of CT into the research. In current studies, numerous different instruments are used to examine the outcome of interventions. Most often this entails researchers developing their own tests to assess elements like CT skills and knowledge. From the articles we examined, it is not always clear what these tests entail exactly (for example which questions were asked). It would benefit the CT research community and educators to start developing and using standardized instruments, as this would enable comparison of results and effectiveness between interventions for example by comparing effect sizes.

Looking at the effectiveness of the interventions (RQ3), we can conclude from our current review that the articles describe positive effects of their interventions on a myriad of different constructs. These range from direct measurement of CT skills and knowledge, to the assessment of programming skills and knowledge, or attitudes towards CS and CT. All papers provide indications that their solution works in relation to CT, but this is measured in so many different ways (both the instruments used and the type of data gathered) that it is sometimes difficult to objectively state that CT skills are actually improved after the interventions.

Finally, for RQ4, we investigated the use of interventions adapted to the proficiency levels of students. Only three papers in our selection apply some form of adaptation and tailor the intervention to a student's current proficiency level, even though this has shown to be beneficial for problem solving skills [29]. The types of adaptation applied are different in these studies, but the current level of the student was always assessed in the same way, namely through self-assessment by the student. The question is whether this method is reliable. It may be interesting to investigate other ways of assessing the student's current proficiency level, or to find a more dynamic way of tailoring interventions to the student.

With this review, we give a high-level overview of articles in which CT interventions for higher education are proposed and evaluated. We believe our research is an important addition to the field of CT education. For example, our review includes just six papers also included in the review by Agbo et al. [1]. It would be interesting to look more in-depth at the interventions and evaluations described in the final selection of papers. Examining the elements used in the interventions to develop specific CT skills, for instance, could provide additional insight into the operationalization of CT in higher education. A deeper investigation into the link between the applied definition of CT, the learning goals of the intervention, and the evaluation methods used could be valuable as well. It is important that these are aligned. Also, in our literature review we encountered numerous articles in which CT interventions are proposed but not evaluated (see Figure 1). Investigating those articles further, and evaluating their interventions, will provide additional insight into ways to teach CT.

For both researchers and educators, our review can be a valuable resource when searching for CT interventions in higher education. The articles in our selection offer a description of interventions aimed at developing CT skills, and evaluate those interventions. However, from this review we must also conclude that a lot of work still needs to be done if educators and researchers want to be able to compare and contrast the effectiveness of interventions. Many different interventions are applied in a lot of different settings and they are evaluated using numerous different tools and tests. It would be beneficial for the CT community if consensus could be reached on the definition of CT and its underlying skill set, and if standardized tests to measure CT skills would be developed. This would aid us in developing and testing interventions that help students develop their CT skills optimally.

REFERENCES

[1] Friday Joseph Agbo, Solomon Sunday Oyelere, Jarkko Suhonen, and Sunday Adewumi. 2019. A Systematic Review of Computational Thinking Approach for Programming Education in Higher Education Institutions. In *Proceedings of the 19th Koli Calling International Conference on Computing Education Research* (Koli, Finland) *(Koli Calling '19)*. Association for Computing Machinery, New York, NY, USA, Article 12, 10 pages. https://doi.org/10.1145/3364510.3364521

[2] Malek Alrashidi, Michael Gardner, and Vic Callaghan. 2017. Evaluating the Use of Pedagogical Virtual Machine with Augmented Reality to Support Learning Embedded Computing Activity. In *Proceedings of the 9th International Conference on Computer and Automation Engineering* (Sydney, Australia) *(ICCAE '17)*. Association for Computing Machinery, New York, NY, USA, 44–50. https://doi.org/10.1145/3057039.3057088

[3] Feri Ardiana Aristawati, Cucuk Budiyanto, and Rosihan Ari Yuana. 2018. Adopting Educational Robotics to Enhance Undergraduate Students' Self-Efficacy Levels of Computational Thinking. *Journal of Turkish Science Education* 15, Special Issue (2018), 42–50.

[4] Ashok Basawapatna, Alexander Repenning, and Mark Savignano. 2019. The Zones of Proximal Flow Tutorial: Designing Computational Thinking Cliffhangers. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (Minneapolis, MN, USA) *(SIGCSE '19)*. Association for Computing Machinery, New York, NY, USA, 428–434. https://doi.org/10.1145/3287324.3287361

[5] Satabdi Basu, Gautam Biswas, and John S. Kinnebrew. 2017. Learner modeling for adaptive scaffolding in a Computational Thinking-based science learning environment. *User Modeling and User-Adapted Interaction* 27, 1 (2017), 5–53. https://doi.org/10.1007/s11257-017-9187-0

[6] Bebras. 2019. Bebras, International challenge on Informatics and computational thinking. Retrieved August 23, 2020 from https://www.bebras.org/

[7] Karl Beecher. 2017. *Computational Thinking : A Beginner's Guide to Problem-solving and Programming*. BCS, The Chartered Institute for IT, Swindon, UK.

[8] Kara Alexandra Behnke, Brittany Ann Kos, and John K. Bennett. 2016. Computer Science Principles: Impacting Student Motivation & Learning Within and Beyond the Classroom. In *Proceedings of the 2016 ACM Conference on International Computing Education Research* (Melbourne, VIC, Australia) *(ICER '16)*. Association for Computing Machinery, New York, NY, USA, 171–180. https://doi.org/10.1145/2960310.2960336

[9] Brian R. Belland. 2014. Scaffolding: Definition, current debates, and future directions. In *Handbook of research on educational communications and technology*, M. Spector, M.D. Merrill, J. Elen, and M.J. Bishop (Eds.). Springer-Verlag, New York, 505–518. https://doi.org/10.1007/978-1-4614-3185-5

[10] Nadia Benakli, Boyan Kostadinov, Ashwin Satyanarayana, and Satyanand Singh. 2017. Introducing computational thinking through hands-on projects using R with applications to calculus, probability and data analysis. *International Journal of Mathematical Education in Science and Technology* 48, 3 (2017), 393–427. https://doi.org/10.1080/0020739X.2016.1254296

[11] Karen Brennan and Mitchel Resnick. 2012. New frameworks for studying and assessing the development of computational thinking. In *Proceedings of the 2012 annual meeting of the American Educational Research Association*, Vol. 1. 1–25.

[12] Candido Cabo and Reneta D Lansiquot. 2016. Integrating Creative Writing and Computational Thinking to Develop Inter-disciplinary Connections. In *Proceedings of the 2016 ASEE Annual Conference*.

[13] Jin Cai, Harrison Hao Yang, Di Gong, Jason MacLeod, and Yao Jin. 2018. A Case Study to Promote Computational Thinking: The Lab Rotation Approach. In *Blended Learning. Enhancing Learning Success*, Simon K.S. Cheung, Lam-for Kwok, Kenichi Kubota, Lap-Kei Lee, and Jumpei Tokito (Eds.). Springer International Publishing, Cham, 393–403.

[14] Ana Calderon. 2018. *Susceptibility to Learn Computational Thinking Against STEM Attitudes and Aptitudes*. Springer International Publishing, Cham, 279–299. https://doi.org/10.1007/978-3-319-93566-9_14

[15] Ana C Calderon, Deiniol Skillicorn, Andrew Watt, and Nick Perham. 2020. A double dissociative study into the effectiveness of computational thinking. *Education and Information Technologies* 25, 2 (2020), 1181–1192.

[16] Ibrahim Cetin and Christine Andrews-Larson. 2016. Learning sorting algorithms through visualization construction. *Computer Science Education* 26, 1 (2016), 27–43. https://doi.org/10.1080/08993408.2016.1160664

[17] Sook-Young Choi. 2019. Development of an Instructional Model based on Constructivism for Fostering Computational Thinking. *International Journal of Innovative Technology and Exploring Engineering (IJITEE)* 8, 3C (2019), 381–385.

[18] Computer Science Teachers Association (CSTA), and International Society for Technology in Education (ISTE). 2011. Computational Thinking Leadership

Toolkit. Retrieved February 4, 2020 from http://www.iste.org/docs/ct-documents/ct-leadershipt-toolkit.pdf

[19] Luis Corral. 2018. Experience Report of a Software Development Course in a Faculty of Fine Arts. In *CC-TEL/TACKLE@ EC-TEL.*

[20] Paul Curzon, Peter W. McOwan, Nicola Plant, and Laura R. Meagher. 2014. Introducing Teachers to Computational Thinking Using Unplugged Storytelling. In *Proceedings of the 9th Workshop in Primary and Secondary Computing Education* (Berlin, Germany) *(WiPSCE '14)*. Association for Computing Machinery, New York, NY, USA, 89–92. https://doi.org/10.1145/2670757.2670767

[21] Peter J Denning and Matti Tedre. 2019. *Computational thinking.* MIT Press.

[22] Juan Manuel Dodero, José Miguel Mota, and Iván Ruiz-Rube. 2017. Bringing Computational Thinking to Teachers' Training: A Workshop Review. In *Proceedings of the 5th International Conference on Technological Ecosystems for Enhancing Multiculturality* (Cádiz, Spain) *(TEEM 2017)*. Association for Computing Machinery, New York, NY, USA, Article 4, 6 pages. https://doi.org/10.1145/3144826.3145352

[23] Lorella Gabriele, Francesca Bertacchini, Assunta Tavernise, Leticia Vaca-Cárdenas, Pietro Pantano, and Eleonora Bilotta. 2019. Lesson Planning by Computational Thinking Skills in Italian Pre-Service Teachers. *Informatics in Education* 18, 1 (2019), 69–104.

[24] Peipei Gao, Mingxiao Lu, Hong Zhao, and Min Li. 2019. A New Teaching Pattern Based on PBL and Visual Programming in Computational Thinking Course. In *2019 14th International Conference on Computer Science Education (ICCSE)*. 304–308.

[25] Ting-Chia Hsu, Shao-Chen Chang, and Yu-Ting Hung. 2018. How to learn and how to teach computational thinking: Suggestions based on a review of the literature. *Computers & Education* 126 (2018), 296–310. https://doi.org/10.1016/j.compedu.2018.07.004

[26] Kamini Jaipal-Jamani and Charoula Angeli. 2017. Effect of robotics on elementary preservice teachers' self-efficacy, science learning, and computational thinking. *Journal of Science Education and Technology* 26, 2 (2017), 175–192.

[27] Andrew Jung, Jinsook Park, Andrew Ahn, and Mira Yun. 2017. CS for ALL: Introducing Computational Thinking with Hands-On Experience in College. In *2017 International Conference on Computational Science and Computational Intelligence (CSCI)*. IEEE Computer Society, Los Alamitos, CA, USA, 1073–1078. https://doi.org/10.1109/CSCI.2017.187

[28] Dennis Kafura, Austin Cory Bart, and Bushra Chowdhury. 2015. Design and Preliminary Results From a Computational Thinking Course. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education* (Vilnius, Lithuania) *(ITiCSE '15)*. Association for Computing Machinery, New York, NY, USA, 63–68. https://doi.org/10.1145/2729094.2742593

[29] Slava Kalyuga. 2007. Expertise Reversal Effect and Its Implications for Learner-Tailored Instruction. *Educational Psychology Review* 19, 4 (Dec 2007). https://doi.org/10.1007/s10648-007-9054-3

[30] Cagin Kazimoglu, Mary Kiernan, Liz Bacon, and Lachlan MacKinnon. 2012. Learning Programming at the Computational Thinking Level via Digital Game-Play. *Procedia Computer Science* 9 (2012), 522 – 531. https://doi.org/10.1016/j.procs.2012.04.056

[31] Byeongsu Kim, Taehun Kim, and Jonghoon Kim. 2013. Paper-and-Pencil Programming Strategy toward Computational Thinking for Non-Majors: Design Your Solution. *Journal of Educational Computing Research* 49, 4 (2013), 437–459. https://doi.org/10.2190/EC.49.4.b

[32] Jeong Ah Kim and Hee Jin Kim. 2017. Flipped Learning of Scratch Programming with Code.Org. In *Proceedings of the 2017 9th International Conference on Education Technology and Computers* (Barcelona, Spain) *(ICETC 2017)*. Association for Computing Machinery, New York, NY, USA, 68–72. https://doi.org/10.1145/3175536.3175542

[33] Andrei P Kirilenko and Svetlana Stepchenkova. 2016. Inter-coder agreement in one-to-many classification: fuzzy kappa. *PloS one* 11, 3 (2016). https://doi.org/10.1371/journal.pone.0149787

[34] D. Midian Kurland, Roy D. Pea, Catherine Clement, and Ronald Mawby. 1984. On the cognitive effects of learning computer programming. *New Ideas in Psychology* 2, 2 (1984), 137–168. https://doi.org/10.1016/0732-118X(84)90018-7

[35] Kil Young Kwon and Won-Sung Sohn. 2018. Software Education Model for Non-major Undergraduate Students using DBSEM. *International journal of advanced science and technology* 114 (2018), 35–48.

[36] Anna Lamprou and Alexander Repenning. 2018. Teaching How to Teach Computational Thinking. In *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education* (Larnaca, Cyprus) *(ITiCSE 2018)*. Association for Computing Machinery, New York, NY, USA, 69–74. https://doi.org/10.1145/3197091.3197120

[37] Lap-Kei Lee, Tsz-Kin Cheung, Lok-Tin Ho, Wai-Hang Yiu, and Nga-In Wu. 2019. Learning Computational Thinking Through Gamification and Collaborative Learning. In *Blended Learning: Educational Innovation for Personalized Learning*, Simon K. S. Cheung, Lap-Kei Lee, Ivana Simonova, Tomas Kozel, and Lam-For Kwok (Eds.). Springer International Publishing, Cham, 339–349.

[38] SB Lee and H Lovvorn. 2016. Building computational thinking skills using robots with first-year engineering students. In *Proceedings of the 2016 ASEE Annual Conference.*

[39] Yuen-Kuang Cliff Liao and George W. Bright. 1991. Effects of Computer Programming on Cognitive Outcomes: A Meta-Analysis. *Journal of Educational Computing Research* 7, 3 (1991), 251–268. https://doi.org/10.2190/E53G-HH8K-AJRR-K69M

[40] Ran Libeskind-Hadas and Eliot Bush. 2013. A first course in computing with applications to biology. *Briefings in Bioinformatics* 14, 5 (02 2013), 610–617. https://doi.org/10.1093/bib/bbt005

[41] Jaime L'Heureux, Deborah Boisvert, Robert Cohen, and Kamaljeet Sanghera. 2012. IT Problem Solving: An Implementation of Computational Thinking in Information Technology. In *Proceedings of the 13th Annual Conference on Information Technology Education* (Calgary, Alberta, Canada) *(SIGITE '12)*. Association for Computing Machinery, New York, NY, USA, 183–188. https://doi.org/10.1145/2380552.2380606

[42] Lauren E Margulieux, Brian Dorn, and Kristin A Searle. 2019. *Learning Sciences for Computing Education.* Cambridge University Press, Chapter 8, 208–230.

[43] Mark Meysenburg, Tessa Durham Brooks, Raychelle Burks, Erin Doyle, and Timothy Frey. 2018. DIVAS: Outreach to the Natural Sciences through Image Processing. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education* (Baltimore, Maryland, USA) *(SIGCSE '18)*. Association for Computing Machinery, New York, NY, USA, 777–782. https://doi.org/10.1145/3159450.3159537

[44] L. D. Miller, Leen-Kiat Soh, Vlad Chiriacescu, Elizabeth Ingraham, Duane F. Shell, and Melissa Patterson Hazley. 2014. Integrating Computational and Creative Thinking to Improve Learning and Performance in CS1. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education* (Atlanta, Georgia, USA) *(SIGCSE '14)*. Association for Computing Machinery, New York, NY, USA, 475–480. https://doi.org/10.1145/2538862.2538940

[45] Ferhat Kadir Pala and Pınar Mıhçı Türker. 2019. The effects of different programming trainings on the computational thinking skills. *Interactive Learning Environments* (2019). https://doi.org/10.1080/10494820.2019.1635495

[46] Melissa A. Pasquinelli and Jeff Joines. 2011. Integrating computing into thermodynamics: Lessons learned. In *Proceedings of the 2011 ASEE Annual Conference and Exposition.*

[47] Markeya S. Peteranetz, Leen-Kiat Soh, and Elizabeth Ingraham. 2019. Building Computational Creativity in an Online Course for Non-Majors. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (Minneapolis, MN, USA) *(SIGCSE '19)*. Association for Computing Machinery, New York, NY, USA, 442–448. https://doi.org/10.1145/3287324.3287346

[48] Micah DJ Peters, Christina M Godfrey, Hanan Khalil, Patricia McInerney, Deborah Parker, and Cassia Baldini Soares. 2015. Guidance for conducting systematic scoping reviews. *International journal of evidence-based healthcare* 13, 3 (2015), 141–146. https://doi.org/10.1097/XEB.0000000000000050

[49] Mintu Philip, VG Renumol, and R Gopeekrishnan. 2013. A pragmatic approach to develop computational thinking skills in novices in computing education. In *2013 IEEE International Conference in MOOC, Innovation and Technology in Education (MITE)*. IEEE, 199–204.

[50] Lori Pollock, Chrystalla Mouza, Kevin R. Guidry, and Kathleen Pusecker. 2019. Infusing Computational Thinking Across Disciplines: Reflections & Lessons Learned. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (Minneapolis, MN, USA) *(SIGCSE '19)*. Association for Computing Machinery, New York, NY, USA, 435–441. https://doi.org/10.1145/3287324.3287469

[51] Sarah Monisha Pulimood, Kim Pearson, and Diane C. Bates. 2016. A Study on the Impact of Multidisciplinary Collaboration on Computational Thinking. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education* (Memphis, Tennessee, USA) *(SIGCSE '16)*. Association for Computing Machinery, New York, NY, USA, 30–35. https://doi.org/10.1145/2839509.2844636

[52] Hong Qin. 2009. Teaching Computational Thinking through Bioinformatics to Biology Students. *SIGCSE Bull.* 41, 1 (March 2009), 188–191. https://doi.org/10.1145/1539024.1508932

[53] Lauren B Resnick. 1987. *Education and learning to think.* National Academies.

[54] Anthony V Robins, Janet Rountree, and Nathan Rountree. 2003. Learning and Teaching Programming: A Review and Discussion. *Computer Science Education* 13, 2 (2003), 137–172. https://doi.org/10.1076/csed.13.2.137.14200

[55] Cynthia Selby. 2015. Relationships: computational thinking, pedagogy of programming, and Bloom's Taxonomy. In *Proceedings of the workshop in primary and secondary computing education.* 80–87.

[56] Cynthia Selby and John Woollard. 2010. *Computational thinking: the developing definition.* University of Southampton (E-prints). https://eprints.soton.ac.uk/346937/

[57] Nick Senske. 2017. Evaluation and Impact of a Required Computational Thinking Course for Architecture Students. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (Seattle, Washington, USA) *(SIGCSE '17)*. Association for Computing Machinery, New York, NY, USA, 525–530. https://doi.org/10.1145/3017680.3017750

[58] Amber Settle. 2011. Computational Thinking in a Game Design Course. In *Proceedings of the 2011 Conference on Information Technology Education* (West Point, New York, USA) *(SIGITE '11)*. Association for Computing Machinery, New York, NY, USA, 61–66. https://doi.org/10.1145/2047594.2047612

[59] Letchumanan Shanmugam, Siti Fatimah Yassin, and Fariza Khalid. 2019. Enhancing students' motivation to learn computational thinking through mobile

application development module (M-CT). *International Journal of Engineering and Advanced Technology* 8, 5 (2019), 1293–1303.

[60] Duane F. Shell, Leen-Kiat Soh, Abraham E. Flanigan, Markeya S. Peteranetz, and Elizabeth Ingraham. 2017. Improving Students' Learning and Achievement in CS Classrooms through Computational Creativity Exercises That Integrate Computational and Creative Thinking. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (Seattle, Washington, USA) *(SIGCSE '17)*. Association for Computing Machinery, New York, NY, USA, 543–548. https://doi.org/10.1145/3017680.3017718

[61] HuiRu Shih, Jacqueline M Jackson, Cassandra L Hawkins Wilson, and Pao-Chiang Yuan. 2014. Using MIT app inventor in an emergency management course to promote computational thinking. In *Proceedings of the 2014 ASEE Annual Conference & Exposition* (Indianapolis, Indiana, USA). https://peer.asee.org/23269

[62] Valerie J. Shute, Chen Sun, and Jodi Asbell-Clarke. 2017. Demystifying computational thinking. *Educational Research Review* 22 (2017), 142–158. https://doi.org/10.1016/j.edurev.2017.09.003

[63] Gudrun Socher, Sarah Ottinger, Veronika Thurner, and Ralph Berchtenbreiter. 2019. Future Skills: How to Strengthen Computational Thinking in all Software Project Roles.. In *SEUH 2019*. 56–64.

[64] Young-Hoon Sung and Young-Sik Jeong. 2019. Development and Application of Programming Education Model Based on Visual Thinking Strategy for Pre-service Teachers. *Universal Journal of Educational Research* 7, 5A (2019), 42–53.

[65] Matti Tedre and Peter Denning. 2016. The long quest for computational thinking. In *Proceedings of Koli Calling 2016 (Koli Calling '16)*. ACM, 120–129. https://doi.org/10.1145/2999541.2999542

[66] Andrea C Tricco, Erin Lillie, Wasifa Zarin, Kelly K O'Brien, Heather Colquhoun, Danielle Levac, David Moher, Micah DJ Peters, Tanya Horsley, Laura Weeks, et al. 2018. PRISMA extension for scoping reviews (PRISMA-ScR): checklist and explanation. *Annals of internal medicine* 169, 7 (2018), 467–473. https://doi.org/10.7326/M18-0850

[67] Michele Van Dyne and Jeffrey Braun. 2014. Effectiveness of a Computational Thinking (CS0) Course on Student Analytical Skills. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education* (Atlanta, Georgia, USA) *(SIGCSE '14)*. Association for Computing Machinery, New York, NY, USA, 133–138. https://doi.org/10.1145/2538862.2538956

[68] Vereniging van Universiteiten, Vereniging Hogescholen, and SURF. 2018. Versnellingsplan onderwijsinnovatie met ICT [Acceleration plan education innovation with ICT]. Retrieved August 12, 2020 from https://versnellingsplan.nl/wp-content/uploads/2019/11/Versnellingsplan-2018.pdf

[69] James Walden, Maureen Doyle, Rudy Garns, and Zachary Hart. 2013. An Informatics Perspective on Computational Thinking. In *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education* (Canterbury, England, UK) *(ITiCSE '13)*. Association for Computing Machinery, New York, NY, USA, 4–9. https://doi.org/10.1145/2462476.2483797

[70] Jeannette Wing. 2006. Computational thinking. *Commun. ACM* 49, 3 (2006), 33–35. https://doi.org/10.1145/1118178.1118215

[71] Jeannette Wing. 2010. Computational thinking: What and Why. Retrieved August 23, 2020 from https://www.cs.cmu.edu/~CompThink/resources/TheLinkWing.pdf

[72] Bian Wu, Yiling Hu, A.R. Ruis, and Minhong Wang. 2019. Analysing computational thinking in collaborative programming: A quantitative ethnography approach. *Journal of Computer Assisted Learning* 35, 3 (2019), 421–434. https://doi.org/10.1111/jcal.12348

[73] Aman Yadav, Chris Mayfield, Ninger Zhou, Susanne Hambrusch, and John T. Korb. 2014. Computational Thinking in Elementary and Secondary Teacher Education. *ACM Trans. Comput. Educ.* 14, 1, Article 5 (March 2014), 16 pages. https://doi.org/10.1145/2576872

[74] Aman Yadav, Ninger Zhou, Chris Mayfield, Susanne Hambrusch, and John T. Korb. 2011. Introducing Computational Thinking in Education Courses. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education* (Dallas, TX, USA) *(SIGCSE '11)*. Association for Computing Machinery, New York, NY, USA, 465–470. https://doi.org/10.1145/1953163.1953297