Informatics in Education, 2019, Vol. 18, No. 1, 41–67

DOI: 10.15388/infedu.2019.03

Integrating the Constructionist Learning Theory with Computational Thinking Classroom Activities

Andrew CSIZMADIA[1], Bernhard STANDL[2], Jane WAITE[3]
[1]*Newman University Birmingham, UK*
[2]*Karlsruhe University of Education, Germany*
[3]*Queen Mary University of London, UK*
email: a.p.csizmadia@staff.newman.ac.uk, bernhard.standl@ph-karlsruhe.de, j.l.waite@qmul.ac.uk

Received: January 2019

Abstract. In computer science education at school, computational thinking has been an emerging topic over the last decade. Even though, computational thinking is interpreted and integrated in classrooms in different ways, an identification process about what computational thinking is about has been in progress among computer science school-teachers and computer science education researchers since Wing's initial paper on the characteristics of computational thinking. On the other hand, the constructionist learning theory by Papert, based on constructivism and Piaget, has a long tradition in computer science education for describing the students' learning process by hands-on activities. Our contribution, in this paper, is to present a new mapping tool which can be used to review classroom activities in terms of both computational thinking and constructionist learning. For the tool, we have reused existing definitions of computer science concepts and computational thinking concepts and combined these with our new constructionism matrix. The matrix's most notable feature is its scale of learners' autonomy. This scale represents the degree of choices learners have at each stage of development of their artefact. To develop the scale definitions, we trialed the mapping tool, coding twenty-one popular international computing activities for pupils aged 5 to 11 (K-5). From our trial, we have shown that we can use the mapping tool, with a moderate to high degree of reliability across coders, to analyse classroom activities with regard to computational thinking and constructionism, however, further validation is needed to establish its usefulness. Despite a small number of activities (n = 21) being analysed with our mapping tool, our preliminary results showed several interesting findings. Firstly, that learner autonomy was low for defining the problem and developing their own design. Secondly that the activity type (such as lesson plan rather than online activity) or artefact created (such as physical artefact rather than onscreen activity or unplugged activity), rather than the computational thinking or computer science concept being taught was related to learner autonomy. This provides some tentative evidence, which may seem obvious, that the learning context rather than the learning content is related to degree of constructionism of an activity and that computational thinking per se may not be related to constructionism. However, further work is needed on a larger number of activities to verify and validate this suggestion.

Keywords: computational thinking, constructionism, classroom activities, computer science education.

1. Introduction

As widely known, Wing stated in her refined definition of Computational Thinking (CT) (Wing, 2008), that CT is an approach for solving problems that draws on concepts fundamental to computing. Later, Aho (2012) described the term CT as including algorithm-design and problem-solving techniques that can be used to solve common problems arising in computing. As Yadav *et al.* (Yadav, Gretter, Hambrusch, & Sands, 2017; Yadav, Zhou, Mayfield, Hambrusch, & Korb, 2011) reminded Wing's initial paper (Wing, 2006) points out that CT involves three key elements Algorithms, Abstraction, and Automation. The term CT has been grown since then to a variety of interpretations.

Turning to constructionism, Ackerman (2001) compared Piaget's constructivism (Piaget & Duckworth, 1970) and Papert's development of this in a constructionist way. Drawing the two views together as learning in a constructionism way 'as Piaget and Papert do, that knowledge is actively constructed by a child in interaction with its world, then we are tempted to offer opportunities for kids to engage in hands-on explorations that fuel the constructive process.' (p.1, Ackerman, 2001). Papert's core message that the learner is 'projecting out our inner feelings and ideas is a key to learning. Expressing ideas makes them tangible and shareable which, in turn, informs, i.e., shapes and sharpens these ideas, and helps us communicate with others through our expressions.' (p.4, Ackermann, 2001) This means, that new insights are the sum of single experiences made by applying existing knowledge for enhancing it.

Considering both parts discussed above, constructionism and computational thinking, this paper's intentions are based on the combination of both for selecting and investigating classroom activities. To do this, we designed a matrix, where aspects from both, computational thinking and a constructionist learning approach, could be analysed. The matrix is designed to identify, categorize and examine classroom activities and encompasses three parts: Computer Science Concepts, Problem-Solving Concepts and Levels of Abstraction. This paper's intention is to present an approach of a systematic exploration of classroom activities in terms of constructionist learning and computational thinking. Therefore, we focused on the research questions as stated below.

Considering a mapping tool, which was used to record and code classroom activities to start to investigate and compare activities for their computational thinking and constructionist attribute and student groups at age 5–11:

- *Is our model applicable in investigating computational thinking classroom activities?*
- *Is our model applicable in assessing constructionism classroom activities?*
- *In what way do classroom activities teach computational thinking in a constructionist way?*

This paper is structured as follows. First, we describe the background of our work in literature and further describe an overview on computational thinking, constructionism and the combination of both parts. In the next section, we describe the methods we used to gather and assess the classroom activities. This includes a detailed description of the mapping tool we have developed. This is followed by a presentation of the results and a final discussion of this paper.

2. Background

Computational Thinking (CT) has been widely discussed since Jeannette Wing published her article “Computational Thinking” in 2006 (Wing, 2006). There have been several attempts to define this concept more precisely since then with the discussion converging to a handful of skills which characterize the thinking concepts associated with CT. These concepts include abstraction, decomposition, algorithmic thinking, generalization and evaluation. Algorithmic thinking education has a long tradition in constructionist education. Logo, Scratch and other programming tools invite creative learning of programming and algorithmic thinking. But CT is considered very broad, it covers not only programming and algorithmic skills, but also activities like problem formulation, system modelling and solution evaluation. In identifying an underpinning theory, we first investigate the two parts of the title of our paper:

- Constructionism.
- Computational Thinking.

Despite both topics have their own deep background, of which a description would go far beyond this paper’s scope and word length, the next two subsections will give a short overview and in particular connections to this work.

2.1. *Constructionism*

Ackerman (Ackermann, 2001) in her comparison of Piaget’s constructivism and Papert’s development of constructivism in a constructionist way, drew the two views together describing learning in a constructionism way as *‘that knowledge is actively constructed by the child in interaction with her world, then we are tempted to offer opportunities for kids to engage in hands-on explorations that fuel the constructive process.’* She further asserted Papert’s core message that the learner ‘*projecting out our inner feelings and ideas. is a key to learning. Expressing ideas makes them tangible and shareable which, in turn, informs, i.e., shapes and sharpens these ideas, and helps us communicate with others through our expressions.’* (p. 1, Ackerman, 2001). Similar to Piaget, Papert identifies learning by constructing and reconstructing knowledge through experience. In particular, Papert’s constructionism sets a focus from learning in situations ‘*rather than looking at them from a distance, that connectedness rather than separation are powerful means of gaining understanding’* (p. 8, Ackerman, 2001*)*. This means, that new insights are the sum of single experiences made by applying existing knowledge for enhancing it. Therefore, ‘*hands-on activities are the best for the classroom applications of constructivism, critical thinking and learning* (p. 2, Ackermann, 2001)

In our study we emphasize on examining learners’ meaning-making while constructing an external artefact, as a consequence of the learners’ prior learning and sense-making in mind. However, what that something is, and what degree of autonomy is associated with the process of making is a focus of our study along with the relationship of constructionism and computational thinking.

2.2. *Computational Thinking*

Computational thinking has become a popular term in computer science education with, definitions varying depending on perspective (Tedre & Denning, 2016). Previous works present at least three types of approaches to defining CT: it is a set of skills to help solve problems (Wing, 2006), it is a thought process (Aho, 2012), or it is a problem-solving process (Voogt, Fisser, Good, Mishra, & Yadav, 2015). As widely known, Wing (2008) stated in her refined definition of CT that it is an approach for solving problems that draws on concepts fundamental to computing. Later, Aho (2012, p. 832) described the term CT as including "algorithm-design and problem-solving techniques that can be used to solve common problems arising in computing". As Yadav *et al.* (2017, 2011) reminded Wing's initial paper (Wing, 2006) points out that CT involves three key elements Algorithms, Abstraction, and Automation. The term CT has been grown since then to a variety of interpretations. Settle and Perković (2010), who proposed seven principles for CT across the curriculum, added that CT also involves computation, communication, coordination, recollection, evaluation, and design. For Lee *et al.* (2011) CT involves defining, understanding, and abstraction. Barr *et al.* (2011) suggested that CT involves the design of solutions, implementation of designs, testing, running analysing, reflecting, abstraction, creativity, and group problem solving. Grover *et al.* (Grover, 2013; Grover & Pea, 2018) stated that CT should include among others abstraction, information processing, structured problem-solving decomposition as modularization, iterative recursive thinking, and efficiency. Again, for Lee *et al.* (2011) CT involves defining, understanding, and solving problems, reasoning at multiple levels of abstraction, understanding and applying automation, and analysing the appropriateness of the abstractions made. According to Brennan & Resnick (2012), CT involves three dimensions such as computational concepts (the concepts designers employ as they program), computational practices (the processes of construction), and computational perspectives (the perspectives designers form about the world around them and about themselves).

2.3. *Constructionism in Computational Thinking*

Considering both parts discussed above, constructionism and computational thinking, this papers's intentions are based on the combination of both for selecting and evaluating classroom activities. Therefore, we designed a matrix, where aspects from both, computational thinking and a constructionist learning approach, can be analysed. The matrix is designed to identify, categorize and evaluate such classroom activities and encompasses three parts:

1. Computer Science Concepts.
2. Computational Thinking Concepts.
3. Levels of Abstraction matrix.

Below we are discussing these three dimensions in detail.

Computer Science Concepts

From the viewpoint of computer science education, teaching and learning of computer science (CS) concepts is more important than learning how to use computer systems. Dagienė, Sentance and Stupurienė (2017) categorized in their paper "Developing a Two-Dimensional Categorization System for Educational Tasks in Informatics" the CS concepts in following 5 categories:

- ALP: Algorithms and Programming.
- DDS: Data, Data Structures and Representation.
- CPH: Computer, Processes and Hardware.
- C&N: Communications and Networks.
- ISS: Interactions, systems and society.

Since CS tasks often involve more than one concept, a task may be assigned to more than one category.

Computational Thinking Concepts (CT)

Computational Thinking Skills are considered separately from CS concepts. Computational thinking originates from solving CS tasks, but the essence of these thinking skills can be applied also in all other disciplines. Based on the work of Selby and Woollard (2013), Dagiene *et al*. (2017) categorized also the Computational Thinking Skills into 5 categories:

- ABS: Abstraction.
- ALT: Algorithmic Thinking.
- DEC: Decomposition.
- EVA: Evaluation.
- GEN: Generalisation.

Simple definitions of each of these concepts are (Selby, 2013; Standl, 2017):

- **Abstraction:** ignoring unnecessary detail. When abstracting a problem in a way that helps to solve it, if we had to keep all the details in our heads, we could never get anything done. As we have described above, abstraction is mentioned a CT concept and some have identified it as a core element (Grover and Pea, 2018).
- **Algorithmic Thinking:** considering the sequence of steps. This includes a design of an algorithm to develop the step-by-step instructions for solving the problem. Starting from what already is known and working outward from there by making a plan how to approach to solve the problem.
- **Decomposition:** breaking a problem down into parts. Decomposition involves finding structure in the problem and determining how the various components will fit together in the final solution. Doing decomposition well makes it easier to modify the solution later by changing individual components, and also enables the reuse of components in solutions to other problems.

- **Evaluation:** comparing alternatives and how does the solution work in practice and are there alternatives? Trying to give the problem different inputs to look how the solution works.
- **Generalisation:** creating things that can be reused in more than one scenario. Is the solution to similar problems also applicable and what is needed to do so? How can the solution be generalized and automated? Which parts turned at the evaluation out to be not necessary?

Levels of abstraction (LOA) matrix

The degree of autonomy of learners to make choices about their constructed artefacts was viewed as an important aspect to capture in order to evaluate the opportunity for learners to engage in constructionist learning in computational thinking activities. Three frameworks have been combined and further developed to create a levels of abstraction matrix which provide an opportunity to evaluate autonomy for the different 'levels' or 'stages' for a programming project. The work that we build upon is the Levels Of Abstraction (LOA) framework (Perrenet *et al.,* 2005; Perrenet & Kaasenbrood 2006), the Abstraction Transition (AT) Taxonomy (Cutts *et al.,* 2012) and the Use -Modify- Create approach (Lee *et al.,* 2011). A levels of abstraction (LOA) model (Perrenet *et al.,* 2005; Perrenet & Kaasenbrood 2006) has been suggested to support novice university students in thinking about algorithms in programming of: problem, object, program and execution. This model has been situated for high school learners by renaming the object level as algorithm (Armoni 2013) and with younger K-5 learners by renaming levels as: task for problem; design for object; code for program and running the code for execution (Waite *et al.,* 2016, Waite *et al.,* 2017). Cutt's *et al*. (2012) investigated novice undergraduate 'talk' of programming activities in response to peer instruction questions and proposed the three leveled Abstraction Transition Taxonomy of English, CS speak and Code, where English is used to define the goal, 'CS Speak' for the technical description and code to accomplishes the goal. The *Use, Modify, Create* approach has been suggested as a learning framework, where learners first use products created by other which are not 'theirs', move on to modify products and finally create their 'own' products (Lee *et al.,* 2011).

3. Method

3.1. *Developing the Mapping Tool*

A mapping tool was required to provide a means to classify and evaluate CT and constructionism. As shown in Fig. 1, the tool incorporated three distinct parts:

- Computer science concepts.
- Computation thinking categorization.
- Constructionism matrix.

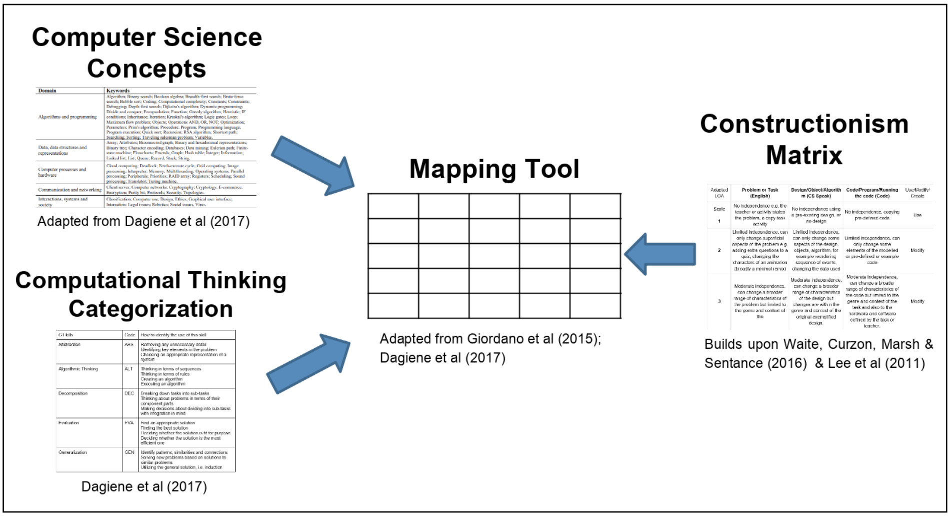

Fig. 1. Dimensions of the mapping tool.

3.2. *Computer Science (CS) Concepts*

In this paper we adopted the computer science concept categorization of school computing as proposed by Dagienė *et al.* (2017) as that of:

1. Algorithms and Programming.
2. Data, Data Structures and Representation.
3. Computer Processes and Hardware.
4. Communications and Networking.
5. Interactions, Systems and Society.

In addition, as guidance from Dagienė *et al.* (2017) was adhered that for practical purposes, a precise definition of each category is required, and this can be achieved by the usage of keywords as shown in Table 1.

Apart from assisting in the categorization of tasks, keywords are helpful to teachers who wish to find tasks that assist in introducing, teaching or formative assessing a specific computing topic (Dagienė and Sentance 2016; Yang and Park 2014).

3.3. *Computational Thinking (CT): Skills Level*

Next, we adopted the categorization of computational thinking as suggested by Selby and Woolard (2013). This categorisation has been adopted by *Computing At School* in the UK in developing guidance for teachers on computational thinking (Csizmadia *et al.,* 2015). Similarly, this approach was adopted by both Giordano *et al.* (2015) and Dagienė *et al.* (2017) in design of a framework for classifying computational thinking skills and computing concepts.

Table 1

Computer Science (CS) Concept Categories and Keywords (Adapted from Dagienė *et al*. (2017))

CS Concept Categories	Code	Keywords
Algorithms and Programming	APL	Algorithm; Binary search; Boolean algebra; Breadth-first search; Brute-force search; Bubble sort; Code; Coding; Computational complexity; Constants; Constraints; Debugging; Depth-first search; Dijkstra's algorithm; Dynamic programming; Divide and conquer; Encapsulation; Function; Greedy algorithm; Heuristic; IF conditions; Inheritance; Iteration; Kruskal's algorithm; Logic gates; Loops; Maximum flow problem; Objects; Operations AND, OR, NOT; Optimization; Parameters; Prim's algorithm; Procedure; Program; Programming Language; Program execution; Quick sort; Recursion; RSA algorithm; Shortest path; Selection; Sequence; Sorting; Steps; Traveling salesman problem; Variables Steps, sequence, algorithm, code, program
Data, Data Structures and Representation	DDS	Array; Attributes; Biconnected graph; Binary and hexadecimal representations; Binary tree; Character encoding; Databases; Data; Data mining; Eulerian path; Finite-state machine; Flowcharts; Fractals; Graph; Hash table; Integer; Information; Linked list; List; Queue; Record; Stack; String
Computer Processes and Hardware	CPH	Cloud computing; Deadlock; Fetch-execute cycle; Grid computing; Image processing; Interpreter; Memory; Multithreading; Operating system; Parallel processing; Peripherals; Priorities; RAID array; Registers; Scheduling; Sound processing; Translator; Turing machine
Communications and Networking	C&N	Client/server; Computer network; Cryptography; Cryptology; E-commerce; Encryption; Parity; Protocols; Security; Topologies
Interactions, Systems and Society	ISS	Classification; Computer use; Design; Ethics; Graphical User Interface; Human Computer Interaction (HCI); Legal issues; Robotics; Social issues;

Table 2 summarizes the five categories of computational thinking:

- Abstraction.
- Algorithmic Thinking.
- Decomposition.
- Evaluation.
- Generalization.

In classifying classroom activities, classifiers needed to know how to identify whether a particular CT skill might be used to solve a given task (Table 2).

One of the difficulties encountered when classifying tasks was that the classifier had to presume how an individual learner solves a specific task. This presumption could differ from the way that either the task setter or other classifiers might solve the task. In practical terms, there may be more than one computational thinking skill associated with each task. Therefore, we followed the guidance suggested by Dagienė *et al.* (2017) of recording a maximum of three computational thinking skills per task.

Table 2

CT Skills and ways to identify them (adapted from Dagienė *et al.* (2017))

CT skills	Code	How to identify the use of this skill
Abstraction	ABS	Removing any unnecessary detail Identifying key elements in the problem Choosing an appropriate representation of a system
Algorithmic Thinking	ALT	Thinking in terms of sequences Thinking in terms of rules Creating an algorithm Executing an algorithm
Decomposition	DEC	Breaking down tasks into sub-tasks Thinking about problems in terms of their component parts Making decisions about dividing into sub-tasks with integration in mind
Evaluation	EVA	Find an appropriate solution Finding the best solution Deciding whether the solution is fit for purpose Deciding whether the solution is the most efficient one
Generalization	GEN	Identify patterns, similarities and connections Solving new problems based on solutions to similar problems Utilizing the general solution, i.e. induction

3.4. *Developing the Constructionism Matrix*

To evaluate classroom activities for their 'degree' of constructionism a new constructionism matrix was devised. The matrix will be evaluated as part of this paper.

Waite *et al.* (2016) suggested that the LOA model might be mapped to the AT taxonomy, with Problem being matched to English, and object to CS speak and Program and Execution to Code. We have combined the Program and Execution level, as our focus is on the autonomy of learners at each level, and they can have no influence on how the code runs. Therefore, we suggest a modified LOA suitable for K-12 of:

- Problem or task (English).
- Object, algorithm or design (CS Speak).
- Program or code and Execution or running the code (Code).

Each of these levels provide the first dimension of our constructionism matrix to which we have added a dimension of *Use*, *Modify*, *Create* (UMC) (Lee *et al.* 2011). As shown in Table 3, for each of these dimensions' descriptions of learner ownership and autonomy were added for each intersection of modified LOA and UMC matrix. An iterative process was undertaken to develop these descriptions. As classroom activities were coded, the scale was reviewed and adapted to enable differences between the activities to be highlighted and then further subdivided to a 1 to 5 scale to describe a gradation of learner autonomy in the construction process. Our final constructionism matrix is shown in Table 3.

Table 3
Constructionism Matrix

Scale	Adapted LOA Problem or Task (English)	Design/Object/Algorithm (CS Speak)	Code/Program/Running the code (Code)	Use/Modify/ Create
1	No independence e.g. the teacher or activity states the problem, a copy task activity	No independence using a pre-existing design, or no design	No independence, copying pre-defined code	Use
2	Limited independence, can only change superficial aspects of the problem e.g. adding extra questions to a quiz, changing the characters of an animation (broadly a minimal remix)	Limited independence, can only change some aspects of the design, objects, algorithm, for example reordering sequence of events, changing the data used	Limited independence, can only change some elements of the modelled or pre-defined or example code	Modify
3	Moderate independence, can change a broader range of characteristics of the problem but limited to the genre and context of the	Moderate independence, can change a broader range of characteristics of the design but changes are within the genre and context of the original exemplified design.	Moderate independence, can change a broader range of characteristics of the code but limited to the genre and context of the task and also to the hardware and software defined by the task or teacher.	Modify
4	Increasing independence, cannot change the 'Genre' but has full control of the context e.g. must be a quiz, or a physical computing problem but can be any context.	Increasing independence, cannot change the design approach to be used, but can adapt the objects, algorithm etc. to meet needs of the problem	Increasing independence, limited by type of hardware and software to be used, but could choose a different input type or type of microcontroller or different block based programming language.	Modify
5	Full pupil independence the pupil has full control over the problem to be considered e.g can be a quiz, game, physical product, unplugged etc.	Full pupil independence the pupil has full control over the design to be considered can choose format and approach to be taken	Full pupil independence the pupil has full control over the hardware, software and implementation choices	Create
6	Levels 1 and 2 seen in a unit of work	Levels 1 and 2 seen in a unit of work	Levels 1 and 2 seen in a unit of work	Use/Modify
7	Levels 1, 2 and 3 seen in a unit of work	Levels 1, 2 and 3 seen in a unit of work	Levels 1, 2 and 3 seen in a unit of work	Use/Modify
9	Not Applicable	Not Applicable	Not Applicable	

As the mapping tool was used further attributes were added, these included:

- Type of artefact made (Physical computing, onscreen, unplugged, concept).
- Type of resource (Lesson plan, game etc).
- Cost (Paid, for free).
- Activity/Approach URL and description.
- Target age range (see Table 4).

Table 4
Target Age Range

Years old	US grades	English year groups
5–6	K-1	Key Stage 1 Years 1–2
7–11	2–5	Key Stage 2 Years 3–6
12–14	6–8	Key Stage 3 Year 7–9
15–16	9–10	Key Stage 4 Years 10–11
17–18	11–12	Key Stage 5 Years 12–13

3.5. *Selection of Activities*

To select activities, where information was available of the popularity of classroom activities this data was used to inform choices. In the United Kingdom (UK), a review of computer science education identified teachers' most popular suppliers of resources (Royal Society, 2017). To select material provided by the most popular supplier, Barefoot, we contacted them and asked which were the most popular and used these. Also, resources from a spread of K-5 ages was selected from this resource set, the rational for selection is shown in Table 5. For some providers' material is only available with payment, here the materials were not included as there would be no means to then share and compare the approach taken with readers of the research. Our study is not rigorous in sampling, but our intention was to trial the approach for review of materials to suggest next steps for a more complete review. In Table 5 we list all activities involved in our analysis.

Table 5
Selection of Activities for analysis

Nr	Activity Short Name	Type	Paid /free	Selection criteria (e.g., popularity, used prior studies, enforced by educational stakeholders
1	CWCodyRoby	Game	Free	Developed as an unplugged resource for teaching programming concepts, intially for Code Week Italy and has been adopted by CodeEU for Code Week
2	BfCrazyCharacters	Lesson plan	Free	Top 10 of resources according to Royal Society report. Funded by DfE for resources for primary teachers to teacher new computing curriculum. Selected as specifically about teaching algorithms using an unplugged approach
3	BfBeeBotBasics	Lesson plan	Free	Top 10 of resources according to Royal Society report. Funded by DfE for resources for primary teachers to teacher new computing curriculum. Selected as specifically about teaching programming using programmable toys

Continued on next page

Table 5 – Continued from previous page

Nr	Activity Short Name	Type	Paid /free	Selection criteria (e.g., popularity, used prior studies, enforced by educational stakeholders
4	BfVikingRaid	Lesson plan	Free	Top 10 of resources according to Royal Society report. Funded by DfE for resources for primary teachers to teacher new computing curriculum. Selected as specifically about teaching programming using online programming language focused on teaching sequence and repetition
5	BfMathsQuiz	Lesson plan	Free	Top 10 of resources according to Royal Society report. Funded by DfE for resources for primary teachers to teacher new computing curriculum. Selected as specifically about teaching programming using online programming language focused on teaching selection and variables
6	BfNetworks	Lesson plan	Free	Top 10 of resources according to Royal Society report. Funded by DfE for resources for primary teachers to teacher new computing curriculum. Selected as specifically about teaching networks
7	CIScratchJnr	Lesson plan	Free	Top 20 of resources according to Royal Society report. Very popular resource created by a CAS master teacher, teacher trainer. This item selected as it is for programming with youngest age groups with scratch jnr
8	CIJamSandwich	Lesson plan	Free	Top 20 of resources according to Royal Society report. Very popular resource created by a CAS master teacher, teacher trainer. This item selected as it is for teaching algorithms – very popular.
9	CICrumble	Lesson plan	Free	Top 20 of resources according to Royal Society report. Very popular resource created by a CAS master teacher, teacher trainer. This item selected as it is for teaching physical computing with a programmable microcontroller
10	CIInternet	Lesson plan	Free	Top 20 of resources according to Royal Society report. Very popular resource created by a CAS master teacher, teacher trainer. This item selected as it is for teaching using search in an unplugged way
11	CIMagicCarpet	Lesson plan	Free	Top 20 of resources according to Royal Society report. Very popular resource created by a CAS master teacher, teacher trainer. This item selected as it is for teaching programming.
12	SWWeAreDetectivesYr3	Lesson Plan	Free sample	Top 10 of resources according to Royal Society report. Funded by a private education publisher. A popular and free sample.
13	SWLearnToCodeYr5	Lesson Plan	Free sample	Top 10 of resources according to Royal Society report. Funded by a private education publisher. A popular and free sample.
14	SWLearnToCodeYr5	Lesson Plan	Free sample	Top 10 of resources according to Royal Society report. Funded by a private education publisher. A popular and free sample.

Continued on next page

Table 5 – Continued from previous page

Nr	Activity Short Name	Type	Paid /free	Selection criteria (e.g., popularity, used prior studies, enforced by educational stakeholders
15	CodeMonkeyPupilActiv	Online activity	Free (1–30 challenges) Paid	This game is recommended for primary school in Lithuania according the project "Informatics in primary education" `https://informatika.ugdome.lt/en/about-project/` CodeMonkey Awarded Best Coding & Computational Thinking Solution
16	ScratchJnrLessonPlan	Lesson plan	Free	This tool is recommended for primary school in Lithuania according the project "Informatics in primary education" `https://informatika.ugdome.lt/en/about-project/`
17	SJnrAnimatePlayground	Lesson plan	Free	`https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8363498` `https://dl.acm.org/citation.cfm?id=2532751`
18	LightBotIntro	Online activity	Free	According `https://venturebeat.com/2014/06/03/12-games-that-teach-kids-to-code/` and `https://dl.acm.org/citation.cfm?id=3017728`
19	KodableFuzzyJava	Lesson plan	Free	It focuses on excellent instruction with group and independent practice activities that build creativity, communication, and collaboration.
20	KodableGoogleForms	Lesson plan	Free	It focuses on excellent instruction with group and independent practice activities that build creativity, communication, and collaboration. Their goal is to reach all students and see computer science become part of a complete elementary education
21	CodeORGFrozen	Online activity	Free	`Code.org` provides sequences of videos and puzzles where users control characters from popular games like Rovio's Angry Birds or movies like Disney's Frozen with drag-and-drop programming.
22	KhanIntroJS	Online activity	Free	According Khan Academy answer about the most popular course
23		Lesson plan	Free	Hour of Code

To supplement the UK's top resources, a number of other popular resources were selected. One activity (CodeMonkey) was chosen because this game-based environment was awarded as the Best Coding and Computational Thinking Solution[1] in 2018. This classroom activity is recommended in informatics educational content in primary education in Lithuania as a tool that engages student to learn informatics concepts by practice not only during informatics lessons, but also during other subjects such as math, etc. (firstly, students solve problem without computer (e.g. measure the distance) and then

[1] `https://www.playcodemonkey.com/blog/2018/06/20/codemonkey-awarded-best-coding-computational-thinking-solution/`

Fig. 2. Students solve problem without computers and then solve it with computers, discusses about coding with CodeMonkey (Taurage “Saltinis” Progymnasium, Lithuania).

repeat the same task using CodeMonkey tool as depicted in Fig. 2). Other activities are applied or recommended as classroom activities in primary or middle school in Lithuania as tools to teach computer science concepts or programming. Several activities were the most popular during the Hour of code[2] (Code Event) in 2015. In addition, some resources (Scratch, Khan Academy, Code.org) are mentioned as the best free resources for teaching youngsters to code.[3]

3.6. *Using the Mapping Tool.*

Three researchers, including two of the authors used the mapping tool to review activities. One researcher, one of the authors, reviewed all activities and then a second researcher double blind coded each activity. Therefore, each activity was independently coded twice. The results were copied into SPSS for Inter reliability evaluation using Cohen’s kappa (Cohen, 2011).

4. Results

We report on descriptive statistics for the coding of activities: 57% (n = 12) of the activities were for 7 to 11 year olds, 23% (n = 5) for 5 to 6 year olds, 2 were for any age from 5 years, 1 was for any age between 5 and 11 and 1 activity for students from 10 years onwards. Therefore, all activities were judged to be suitable for some set of pupils in the K-5 (primary age range in England).

Over 75% of the activities selected were lesson plans (n = 16), the remainder were online activities (n = 4) except for one board game. Two thirds of the activities originated from England (n = 14), 29% from the US (n = 6) and one from Italy. Nearly 30% of the activities employed Scratch (n = 6), 15% ScratchJr (n = 3), there were 2 route-based programming languages where the student moved an onscreen character

[2] https://blogs.sas.com/content/sascp/2015/12/07/our-favorite-hour-of-code-resources-for-csedweek15/

[3] https://codakid.com/top-5-free-kids-coding-websites-of-all-time/

or programmable toy with direction keys and there was one activity for each of the programming languages of blocky, CoffeeScript, Python, Java and a physical computing software called Crumble. There was also one activity that used Google Forms and 6 activities which used no programming language or specific software to make something.

In order to give a % for each concept type we have taken the average number of coded activities for each concept and present this as a percentage out of the 21 activities as shown in Table 6. On average, the most popular CS concept taught in the sample of activities codes was ALP (Algorithms and Programming) with 90% of the activities coded to this concept. No activities were coded to Computer Processes and Hardware concept. For CT concepts, the most popular coded concept was ALT (Algorithms) with, 83% of the activities coded as teaching this concept, the second most popular CT is DEC (decomposition) with 45% of activities coded for this. The most frequent artefact type coded was concept at 79%, followed by 64% for on screen artefact types.

As shown in Table 7, the majority of activities were coded at level 1 scale for Problem or Task level at 52% and 61% for coders, followed by 24% to 29% at level 2 and 9%

Table 6

CS concept, CT concept and artefact type counts by coder and % out of 21 activities

	CS concepts					CT concepts					Artefact type			
	ALP	DDS	CPH	C&N	ISS	ABS	ALT	DEC	EVA	GEN	Physical	OnScreen	UnPlugged	Concept
Coder 1	19	2	0	2	3	4	19	9	9	1	2	13	10	16
Coder 2	19	1	0	2	4	5	16	10	7	3	2	14	7	17
Average	19	1.5	0	2	3.5	4.5	17.5	9.5	8	2	2	13.5	8.5	16.5
%	90%	7%	0%	10%	17%	21%	83%	45%	10%	10%	10%	64%	40%	79%

Table 7

Percentage and count (n) of coded activities by constructionism scale

Adapted LOA Scale	Problem or Task (English)		Design/Object/ Algorithm (CS Speak)		Code/Program/Running the code (Code)	
	Coder 1	Coder 2	Coder 1	Coder 2	Coder 1	Coder 2
1	61% (13)	52% (11)	24%(5)	14% (3)	9%(2)	5% (1)
2	*5% (1)*	*0*	*28%(6)*	*33% (7)*	*52%(11)*	*48%(10)*
1 to 2	*19% (4)*	*29% (6)*	*24% (5)*	*38%(8)*	*5% (1)*	*13%(3)*
2 or 1 to 2	24% (5)	29% (6)	53% (11)	71%(15)	57% (12)	61%(13)
3	*5% (1)*	*0*	*5% (1)*	*5% (1)*	*5% (1)*	*0*
1 to 3	*5% (1)*	*9%(2)*	*0*	*5 (22%)*	*0*	*5% (1)*
3 or 1 to 3	10% (2)	9%(2)	5% (1)	5%(1)	5%(1)	5% (1)
4	*0*	*0*	*0*	*0*	*0*	*0*
5	*0*	*5% (1)JS*	*0*	*0*	*0*	*5% (1)JS*
4 or 5	0	5% (1)	0	0	0	5%(1)
coded as not applicable	5% (1)	5% (1)	19% (4)	10%(2)	29%(6)	24%(5)
n	21	21	21	21	21	21

to 10% for level 3 with only 1 activity assigned a 4 or 5 level. The majority of designs where at scale 2 at 53% to 71%, with 14% to 24% at level 1 and only 5% at level 3. As with the design level, the majority of activities were rated at a level 2 scale for the code dimension at 57% and 61% with between 5% and 9% at level 1, and one activity assigned a level 5. For the coding dimension a quarter, 24% to 29% were not assigned a level (for example as they were unplugged activities).

Cohen's κ was run to determine if there was agreement between the authors coding of activities. Inter-reliability, agreement was almost perfect on artefact type ($\kappa = 0.82$, $p < 0.005$) and computer science concepts ($\kappa = .868$, $p < 0.005$). For CT concepts the agreement was less but was still a substantial agreement ($\kappa = .74175$, $p < 0.005$) and for the Constructionism matrix agreement was moderate ($\kappa = .51$, $p < 0.005$).

Summary Dimensions: To simplify reporting of the constructionism scale we have combined the '1 to 2' response with the 2 response and the '1 to 3 response' with the 3 response for each of the dimensions. We have also removed the 4 and 5 scale responses as there was only 1 activity coded at the 5 scale by one coder, and the second coder placed this same activity as 3, so we have classified this as an outlier and will further consider this case in discussion. In doing this we have created Summary Dimensions with scales of 1, 2 and 3.

We used the Mann-Whitney U statistic to investigate whether there was any statistically significant relationships between the constructionism scale and its CS concept or CT concept. Here the cases, were the coding of each activity by each researcher therefore the maximum population was 42 for these tests. The null hypothesis was that for each CT and CS concept there was no statistically significant difference between whether an activity was coded for a particular constructionism scale. The null hypothesis for all CS and CT concepts could not be rejected as all tests showed no statistically significant difference. The test statistics and cross tabulation are available on request from the authors.

Similarly, we performed the same statistic on the artefact types reported by the coders. As there was only one board game we removed this from the test and used a Mann-Whitney U to compare lesson plans online student activities for each of the dimensions of the constructionism matrix. The null hypothesis for these tests was that there would be no statistically significant differences in the grading of the constructionism scale for lesson plan based resources or online student resources. There were two statistically significant differences, for the Design level ($p = 0.018$, $n = 25$, U-54.5, $r = 3.98$) and the Code level ($p = .048$, $n = 29$, $U = 37$. $R = .398$) both of medium effect size, meaning we reject the null hypothesis. A cross tabulation of the data is shown in Table 8 and shows that the scale of autonomy was recorded at a higher level for lesson plans than for online resources. This may be significant in that it is may not be the CS or CT concept that is having a bearing on the constructionism aspect of a task, more what type of activity it was.

We also compared the constructionism matrix scales to different types of artefacts created by activities. This data is available on request from the authors. The null hypothesis was that there would be no difference in the reported autonomy of students for whether the artefact was of a particular type or not. A Kruskal Wallis with bonferroni

Table 8

Cross Tabulation of activity types by constructionism matrix dimensions
showing % of type and (n)

		Problem			Design			Code		
		1	2	3	1	2	3	1	2	3
Activity Type	Lesson Plan	61%(19)	29%(9)	10% (3)	14%(4)	82%(23)	4%(1)	4% (1)	92%(21)	4%(1)
	Online Student Activity	71%(5)	29%(2)	0	57%(4)	43%(3)	0	33%(2)	67%(4)	0
	Board Game	0	0	100%(1)	0	0	100%(1)	0	0	100%(1)

correction pairwise test showed 'unplugged compared to physical' (p = 0.013, n = 17. U = 5.678, r = .694) and 'on screen compared to physical' (p = 0.016, U = 5.328, n = 29, r = .517) had a statistically significant differences at the problem level both with large effect size, but no statistically significant difference for the others. However, caution is urged as there were only 2 activities creating this artefact type, each coded by a different coder. A cross tabulation (Table 9) showed that physical activities had a higher autonomy scale than non-physical for the problem dimension. This indicates that the type of artefact may be having a bearing on the degree of constructionism of a task, rather than the CT or CS concepts.

To investigate if there was any statistically significant difference in coders assignment of scales for each of the constructionism dimensions we performed the pair statistic of the Wilcoxon Signed Rank test, see Table 10. The null hypothesis was that there would be no statistically significant difference in scale assignment across the problem and design, design and code and problem and code dimensions. There was a statistically significant evidence to reject the null hypothesis, indicating that there was a relationship between coders allocation of scales across the dimensions. In our population coders rated the design scale as higher than the problem scale with moderate effect size, and the code scale higher than the design and the code scale higher than the problem. This indicates that our coders rated activities as being more 'constructionist in the dimensions' of coding, then design and then lowest in problem.

Table 9

Cross Tabulation of artefact types by constructionism matrix dimensions for both coders.
Showing % of type and (n) using Summary Dimensions

		Problem			Design			Code		
		1	2	3	1	2	3	1	2	3
Artefact type	Unplugged	75%(9)	17%(2)	8%(1)	12.5%(1)	75%(6)	12.5%(1)	25%(1)	50% (2)	25%(1)
	Onscreen	65%(15)	31%(7)	4%(1)	29%(7)	67%(16)	4%(1)	9%(2)	86%(19)	5%(1)
	Physical	0	50%(2)	50%(2)	0	100% (4)	0	0	100% (4)	0
	Concept	65%(19)	21%(6)	14%(4)	19%(5)	73%(19)	8%(2)	10%(2)	80%(16)	10%(2)

Table 10

Comparing the changes in coders rating of scale across the constructionism matrix dimensions using the Wilcoxon Signed Rank test.

Test statistics	Comparing constructionism scales		
	Problem to Design	Design to Code	Problem to Code
Increase in scale	12 (sum of ranks 90)	4 (sum of ranks 10)	14 (119)
Decrease in scale	2 (sum of ranks 15)	0 (sum of ranks 0)	2 (17)
Tied	20	26	14
n	34	30	30
p	p = 0.008	p = 0.046	p = .003
Z	-2.673	-2.00	-3.00
r	-.46	-.37	-.55

5. Discussion

To organize our discussion, we start by outlining limitations of our study, we then discuss our findings for each computational thinking concept and overall for the constructionism matrix and finish by reflecting on the research questions.

5.1. *Limitations*

An initial point to note was the difficulty in creating a method for examining the degree to which constructionism is incorporated in activities, a matrix has been suggested, but this requires further validation as a useful approach. When creating the matrix there was discussion of what were the most important features of a constructionist activity. According Papert (1980), important is learning experience that engage students in constructive activities that are meaningful to them. Furthermore, activities should be accessible to students with different styles of thinking and learning. Harel stressed that "students become deeply involved and gain deeper understanding… through the process of constructing, programming, and explaining their own representations" (Kafai, 1994, p. 24). Our focus led to the attribute of learner autonomy in creating artefacts at each of the main stages of a product development. Here an adapted levels of abstraction framework was suggested with dimensions of problem, design and code. A scale of autonomy from levels 1 to 5 was suggested with 1 representing the least amount of learner control of the artefact being created to 5 representing the learners having complete choice over what to make and how.

During the process of review of activities, it appeared that the scale criteria description was not sufficient to distinguish between the sample of activities. Some coders rated activities at '1 to 2' or '1 to 3', this was because the activities, rather than being a single small activity were a unit of work across several lessons, or within a single lesson there was a 'graduation' of pupil autonomy. This 'graduation' may have been over the course of the lesson as different 'tasks' occurred, such as an initial closely controlled task, followed by a task where there was more student control or it could be that over several lessons pupils were moved from use to modify (Lee *et al.* 2011a).

In order to deal with this added requirement, two extra levels were added to the scale of ranges of '1 to 2' or '1 to 3' as coding value options. However, a % or degree of each level may have been more useful. Within the qualitative data of 'Justification of coding' variable' one coder said "I was loathing to allocate a level 2, as this only occurred in the final 20th challenge and up to this point there had been absolutely no student autonomy at all "(Coder 2)

However, during reporting on the data, these two extra 'range' scales were merged back in with their respective highest level, this was because of the small number of activities which were surveyed and the need for broader groups for statistical analysis. In further work, a larger sample of activities is needed, and the more granular levels can then be used to draw out clearer distinctions between activities.

Selection of activities to sample was problematical. In the UK there are a large number of resources available for educators to select from and recent surveys which have reported on the most popular (Royal Society, 2017). However, in other countries the task of finding resources to evaluate was not so easy. In other countries teachers can use resources that are internationally available, such as the code.org materials but in some countries, teachers are more likely to create their own activities and that these are not then shared. For these countries, a generic set of activities were created to represent these countries resources. Whether or not these are representative of what is being actually used in class in difficult to assess. Similarly, in the UK, it is not clear as to whether teachers are using resources in their published form or if they are adapting them, in a recent survey of 207 teachers (data not yet published) 40% of teachers reported they created their own resources, including adapting resources from over 70 specifically named resource sets. The most popular three resource mentioned in this survey was the Barefoot materials (Berry *et al.* 2015) with 34% of the teachers mentioning using these resources followed by and 16% mentioning Code It and 13% Switched on, of which we have reviewed sample material in our survey here. Although, teachers may choose popular computer science or computational thinking activities sometimes the diversity of activities depends on policies, curriculum, grades, etc. For example, robotics has become very popular in secondary schools in Finland; computer science equipment is provided by the States and the local communities in Germany; programming with Scratch is advocated in primary schools in Ireland; learning objectives include programming skills and knowledge of computer hardware in required in secondary schools in Lithuania; a course that includes programming with Scratch or Kodu is available in Portugal computer science education (7th–8th grade) (Passey, 2017). Teaching of computer science integrated to other educational subjects is started in primary school in Lithuania in 2017. In some cases, teachers use computer science activities as a way to demonstrate how the same problem or task of real life could be solved by using computers. For example, students may solve problems during mathematics lesson, on their own or in pairs and at the end of the lesson solve the same problem using a particular tool, such as Scratch, code.org or Codemonkey. This gives opportunity for students to understand the problem at a deeper level, practice solving it in different ways and can promote discussion between students.

Similarly, there was debate as to whether to include activities in which no physical ‘artefact’ was constructed rather than learners ‘constructed conceptual understanding’. For one activity, a board game, the learners constructed their learning, as they ‘used’ other games, but then went on to create their own version, very much transitioning through a use, modify, create approach (Lee *et al.* 2011). This transition from using to making seemed to be the salient point in which constructivism (Piaget 1970) switches to constructionism (Papert 1970) and as long as activities supported this transition then they could be included as they provided a constructed artefact. Therefore, activities such as a Bebras task (Dagienė &. Sentance 2016) would not be included, as they support development of a CT concept but do not include an element of then making or constructing of an adapted or brand-new version of a task. This is a problematical issue as a teacher could use a Bebras task as a starting point and then adapt this context into a constructionist activity. How teachers are using activities in practice was not captured by our survey of materials. But our approach could be used as a starting point to further investigate teachers transition of resources from constructivist to constructionist activities.

In the case of creating Bebras tasks, Dagienė *et al.* (2016) mentioned a constructionist and deconstructionist learning ways. Constructionist way of learning is the creation of computer science task and deconstructionist way of learning is that a computer science concept is analysed and deconstructed in its main aspects (discovering why it is computer science).

Our review was of single activities rather than of sets of activities sitting with a progression of development of knowledge, skills and understanding. This is a limitation of our study. Similarly, some coders had considered the use of a construct such as a procedure or function was sufficient to warrant generalisation could be assigned, whereas for other coders, only if learners had themselves generalised rather than copied or used someone else’s generalisation could that attribute be assigned to an activity. There are opportunities to review the progression of the constructionism matrix and CT concepts through time. For example, the Solo taxonomy (Biggs, 1982) starts with the learner having no experience of a concept and moves to them being able to apply the concept in new and novel scenarios. There is opportunity to map this taxonomy, or other similar to the constructionism matrix.

5.2. *Computational Thinking Concepts & Constructionism*

In order to answering our research questions, we first summarise our findings by computational thinking concepts. For each concept we draw out key results related to concept allocation and the data from the use of the constructionism matrix and relate these to literature.

ALT – Algorithmic Thinking

Despite having no statistically significant statistics related to our data for the algorithmic thinking CT concept, there are interesting features of the descriptive data. ALT was our most popular CT concept coded for the 21 activities; 16 to 19 activities were coded as teaching this concept by our coders. Considering whether there might be specific reasons why algorithmic thinking might be particularly suited to constructionism then we turn our attention ideas on progression for this concept and the work by Rich, Strickland, Binkowski, Moran, & Franklin (2017) who suggested a learning trajectory for developing sequence. Rich *et al.* grouped initial learning about sequence into everyday activities related to ordering and precision, suggesting that learners transitioned from understanding the world around them for a particular concept to then applying this in a programming context with an intermediate phase of computational thinking. An example of a goal on this trajectory is "During this trajectory they need to learn that programs are made by assembling instructions from a limited set" (page 187). Whether this objective is more effectively met through CodeMonkey activities with no choice of the problem, the design or the code (as there is only one possible solution to a puzzle), or with Lightbot, with similar restrictions or through a more open exploration of a ScratchJr activity to draw a square, which still is a level 1 problem, but which allows learners to select the character that draws the square and the learner can choose where exactly to start, how big the square might be and can embellish this activity in next steps, so this becomes a level 2 design and code activity has not been evidenced by our study. However, we have now provided a means by which different activities can be coded and compared quantitatively.

DEC – Decomposition

Just less than half of our activities were assigned to the decomposition concept (dependent on coder per Table 6). These were relatively evenly split between level 1 and level 2 on the problem dimension with a couple of activities at level 3. Why decomposition might have lower pupil autonomy at the design level is interesting, whether this is because design is lacking as a clear and identified step may be a contributing factor. A recent review of resources for teaching computing, concluded that there was a lack of resources with design (Falkner & Vivian, 2015) and a study matching learning goals against research cited design as the most unmatched goal (Rich, Strickland & Franklin, 2017). We did not specifically ask coders to reflect on this, however one coder noted that 'Design element was STRONG' (coder 2) in activity 16 but non-existent in the puzzle-based activities which have a predetermined problem and solution algorithm. In undertaking design students are required to break their problem down into parts (decompose it), they need to consider the level of detail that is appropriate (and so are also abstracting) and order these items (and so are practising algorithmic thinking). Considering how students are experiencing design they may be copy an existing design, or working in a more constructionism way by creating their own, perhaps activities could be improved by increasing the element of independent design by learners in the same way that they are required to do when they are undertaking other subjects such as when learning to write (Waite, Curzon, Marsh, Sentance, Hawden- Bennett, 2018).

ABS – Abstraction

Only 4 to 5 activities were identified as teaching abstraction and the inter reliability was moderate (κ = .696, p = 0.001), yet abstraction is seen by many as the cornerstone of computational thinking. Wing wrote "The abstraction process, deciding what details we need to highlight and what details we can ignore, underlies computational thinking" (Wing 2008, p. 3718). The activities which have abstraction coded for them are predominantly at level 1 for problem and level 2 for design and code.

In our study we have developed upon a particular use of abstraction through our constructionism matrix and the levels of abstraction, where the dimensions of our matrix represent levels of detail for different purposes of an activity, the problem, design and code, however that does not relate to how activities engaged with the teaching and learning of abstraction.

In looking at our coded activities, as with decomposition, learners are using abstractions as they are given a problem by their teacher or an online system, this is coded at scale 1 and what we saw for most of our small sample. If learners then follow a predefined design (as with the puzzle activities) they are using an abstraction and if they are copying or figuring out a pre-defined code solution, again they are using an abstraction at scale 1.

Despite learners using abstractions, few of our activities were coded with this concept. Perhaps this is because the concept is hidden from both the teacher and the learner, and only when a specific process of abstraction is mentioned would it be coded? We have 2 activities coded level 2 for design and 1 for level 3 at code and design. This is for activity 16, a ScratchJr which the coder justifies her allocation for by saying

> *'The design element is STRONG. I wish I could have given this a much higher score – maybe this needs to be reflected – but the genre was fixed, and the language used. I allocated more CT aspects as kids were deciding on their own designs so abstracting and decomposing as they were designing. I am not sure they did generalisations – maybe because they were doing repeats ... need to discuss more about how we allocated each of the CT concepts – need a clearer definition.'*
> (Coder 2)

However, for this same activity coder 1 disagreed and only gave the activity a level 2 for design and coding and did not allocate decomposition or abstraction as concepts taught.

The comments made by coder 2 implies a constructionism approach was being taken, as learners were 'deciding on their own designs' however, whether this means that learners will make more progress rather than using someone else's is yet to be robustly evidenced for this age group of learners.

GEN – Generalisation

This concept was assigned the fewest times from all of the CT concepts, only 1 to 3 activities were coded as GEN and the inter-reliability was not reported by SPSS as there were no agreements and numbers were small. Even where it was assigned there was a lack of certainty of the allocation, one coder remarked in the justification notes about Activity 18 (Lightbot) ‘‘Although this teaches procedures – is this generalisation or is it decomposition? As the purpose is not to reuse for a logical reason but to reduce code blocks used.’’ (Coder 2) The same coder raised the same question about Activity 19’ I am not sure they did generalisations – maybe because they were doing repeats ... need to discuss more about how we allocated each of the CT concepts – need a clearer definition.’’ Coder 2)

Again, this was raised for another activity:

> *‘If we are considering repeats as decomposition then this is used, I am assuming then as it moves to functions then we can say that generalisation is used HOWEVER the level of use of these is very low. On solo it’s really just first level as using what is provided – maybe unistructural, whether we think this use of a procedure, that someone else made, is generalisation I am really not sure. It will be interesting to see what the other coders thought and whether the learning of generalisation here has been done in a constructionist manner at all.”* (Coder 2)

Two issues are raised by these comments, firstly, is a function or procedure which is used solely to reduce the number of lines of code for something that can be repeated a generalisation, or a decomposition used within a repeat? Does generalisation need to be where we are “transferring a problem solving process to a wide variety of problems” CSTA & ISTE, 2011))? Has this confusion arisen as the term pattern has been associated with generalisation? This term may have been introduced to simplify language and as an instructional approach in a progression of learning for generalisation to encourage the spotting of patterns that might then become generalisations. However, during this progression reusing code to simplify code within a single solution, such as in the activity 16,18 and 19 is this generalisation? Secondly, does using elements created by someone mean we are learning about the process involved in creating those elements? Particularly if the the underlying concept is NOT brought to a learner’s attention?

EVA – Evaluation

Evaluation according to Bloom is the highest order thinking skill (Bloom, 1957) which learners encounter as they gradually spiral through a succeeding progression of more complex material. Of those activities assigned this concept, over half were rated as level 1 (54%) and a third (31%) level 3 for the problem, nearly three quarters (72%) were rated level 2 for design and 82% at level 2 for coding. This is quite a wide spread, with some tendency to the higher levels for code and design. Perhaps implying that evaluation of the code was more prevalent than evaluation of a design. But this is a broad assumption which needs more careful analysis particularly around what we might mean by an activity having evaluation being assigned as teaching that CT concept. In our definition of evaluation, the following phrases were provided to help coders decide if evaluation was a skill USED in the activity ‘Find an appropriate solution’ Finding the best solution’ ‘Deciding whether the solution is fit for purpose’ ‘Deciding whether the solution is the most efficient one’. This list implies an order of progression. Surely the first of which must have been seen in any activity that had some kind of solution. Therefor all activities, if they solved any kind of task or activity should have had an element of evaluation. It may be that coders discounted this first example and only assigned where there was a 'best solution' considered, or if there was a design to which the solution to could be compared to decide if it was fit for purpose. There appears to be more work to be done in defining the rationale for assigning a concept as being associated with an activity, perhaps as a grading of association.

Constructionism Matrix
Looking at the overall statistics for the constructionism matrix and its dimensions and autonomy scales, as shown in Table 7, there is as an overall pattern whereby autonomy for learners is most restricted at the problem stage with around half of activities coded at scale 1. Autonomy for learners is slightly better during design and code where scale 2 was assigned for around half to three quarters of the time. What this indicates, for our sample, is that learners have little control over the context of the activities they are undertaking, they have some opportunity to start to modify the design and the code, but very few opportunities to be involved in a freer form create where they have control over what they might make it and how. Therefore, the level of constructionism might be considered to be low if we measure it based on student autonomy, learners are not moving to the point where they might feel ownership (Lee *et al.* 2011) as they are not yet making their own new artefacts.

5.3. *Research Questions*

Is our model applicable in investigating computational thinking classroom activities?

Having adopted Dagienė et all's (2017) model, we were able to assign computational thinking concepts to classroom activities with a substantial agreement of coders and this provided a means by which we could then investigate potential correlations with constructionism. However, as outlined in Section 5.2 there were questions raised about assignment of some concepts, therefore further work is needed in this area.

Is our model applicable in assessing constructionism classroom activities?

Despite limitations, we were able to use our constructionism matrix and assign our autonomy scale to activities with a moderate degree of agreement between coders. Following on from our discussion in Section 5.2., further work is needed in this area to develop guidance for coders and to verify this approach as a way to view constructionism. However, our matrix provides a useful starting point.

In what way do classroom activities teach computational thinking in a constructionist way?

In Section 5.2 we reviewed each computational thinking concept, reflected on our findings and literature to suggest how the classroom activities investigated taught computational thinking in a constructivist way. Based on this there is still much to do to look at a wider range of activities and to particularly resolve the question of what as well as the autonomy scale needs to be considered to reflect constructionism in practice.

6. Conclusion and Further Research

Our quantitative analysis should be viewed with much caution as our sample size was small at only 21 activities, and only 3 authors coded, each doing a mixture of 1st and 2nd coding. However, this small number of activities became 42 cases to review researchers views of the attributes of activities. Our approach for classification appears to have some reliability, as there was substantial agreement across the variables and coders. However, for some activities there was very different allocation of the constructionism scale, such as for the activity 22 the JavaScript onscreen activity which was the only activity to be awarded more than a scale of 3. It was allocated a 1–2 by one coder for all three dimensions but a 5 for just the problem and coding and a 1 for design by the other coder. The rationale for this may be that the activity sat within an overall complete unit of work, which eventually might lead students to be given a chance to create a new artefact using any product, in any context, but within the activity reviewed this was not the case. This indicates an issue with how an instrument to measure the constructionist aspect of activity might scope the boundaries of an activity.

Despite the limitations of our quantitative work, a number of interesting data results have been revealed. There are no differences between activities with each of the CT and CS concepts across the scales of each of our problem, design and code dimensions of our constructionism matrix. But there were differences for other aspects, including the activity type, such as lesson plan compared to onscreen student activity and also the artefact created such as a physical artefact compared to an onscreen activity or an unplugged activity.

A resulting suggestion might be that CT & CS concepts are merely the content that is being delivered by a constructionism approach, the success of this approach is less impacted by the underlying material being delivered but more by the techniques applied to deliver it, such as through human (teacher) mediated activity as opposed to a symbolic (system) mediated (Kozulin, 2003) event, or through the creation of a physical artefact compared to an onscreen or unplugged artefact.

In summary, the contribution of this paper is the presented mapping tool incorporating a new framework, called the constructionism matrix, for reviewing activities which are used in the teaching and learning of computing in terms of constructionism and computational thinking. We have identified limitations with our framework and with our trial, but from its development and using it with a small sample of lesson plan and online student activities (n=21) targeted at K-5 learners, we have been able to report the trial results of its first use, discuss problems encountered and suggest opportunities for improvement of both the mapping tool and the matrix.

For further work, we suggest to further refine the constructionism matrix, by adding % of use of each scale and more closely defining the scope of the activities reviewed, it should then be used to survey a larger population of resources. Similarly, more work is needed to reflect on the depth of learning of CT & CS concepts perhaps in a similar scale as the student autonomy using Solo taxonomy or other more finely grained frameworks of a degree of learning. Our mapping tool will then have a combined framework, which might provide more insight into the potential relationship between constructionism and computational thinking.

References

Aho, A. (2011). Computation and computational thinking. *Ubiquity*, (January), Article No. 1.

Ackermann, E., 2001. Piaget's constructivism, Papert's constructionism: What's the difference. Future of learning group publication, 5(3)

Armoni, M. (2013). On teaching abstraction in computer science to novices. *Journal of Computers in Mathematics and Science Teaching*, 32(3), 265–284.

CSTA & ISTE (2011). *Operational Definition of Computational Thinking for K–12 Education*. Retrieved from `http://csta.acm.org/Curriculum/sub/CurrFiles/CompThinkingFlyer.pdf`

Berry, M., Woollard, J., Hughes, P., Chippendal, J., Ross, Z., Waite, J. (2015). *Barefoot Computing Resources*. Last accessed 8th August 2018. Retrieved from `http://barefootcas.org.uk/`

Biggs, J., Collis, K. (1982). Origin and description of the SOLO taxonomy. In: *Evaluating the Quality of Learning: The SOLO Taxonomy.* New York: Academic Press Inc, 17–30.

Bloom, B.S. (1956). *Taxonomy of Educational Objectives: The Classification of Educational Goals, Handbook 1 Cognitive Domain*. New York: David McKay Co. Inc.

Bloom, B., Anderson, L., Sosniak, L. (1994). *Bloom 's Taxonomy, a Forty Year Retrospective*. Chicago, IL: University of Chicago.

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

Csizmadia, A., Curzon, P., Dorling, M., Humphreys, S., Ng, T., Selby, C., Woollard, J. (2015). Computational thinking: A guide for teachers. *Computing at Schools*.

Dagienė, V., Futschek, G., Stupurienė, G. (2016). Teachers' constructionist and deconstructionist learning by creating Bebras tasks. In: *Constructionism in Action. Conference Proceedings*.

Dagienė, V., Sentance, S., Stupurienė, G. (2017). Developing a two-dimensional categorization system for educational tasks in informatics. *Informatica*, 28(1), 23–44.

Cohen, L, Manion, L., Morrison, K. (2011). *Research Methods in Education*. Vol. 7th Edition. Routledge.

Cutts *et al*., (2012). The abstraction transition taxonomy: developing desired learning outcomes through the lens of situated cognition. In: *Proceedings of the Ninth Annual International Conference on International Computing Education Research*. ACM, 63–70.

Dagienė, V., Sentance, S. (2016). It's computational thinking! Bebras tasks in the curriculum. In: *International Conference on Informatics in Schools: Situation, Evolution, and Perspectives*. Springer, Cham, 28–39.

Falkner, K., Vivian, R. (2015). A review of computer science resources for learning and teaching with K-12 computing curricula: An Australian case study. *Computer Science Education*, 25(4), 390–429.

Giordano, D., Maiorana, F., Csizmadia, A.P., Marsden, S., Riedesel, C., Mishra, S., Vinikienė, L. (2015). New horizons in the assessment of computer science at school and beyond: Leveraging on the viva platform. In: *Proceedings of the 2015 ITiCSE on Working Group Reports*. 117–147.

Grover, S., Pea, R. (2013). Computational thinking in K12 – A review of the state of the field. *Educational Researcher*, 42(1), 38–43.

Lee, I. *et al*. (2011). Computational thinking for youth in practice. *ACM Inroads*, 2(1), 32–37.

Grover, S., Pea, R. (2018). Computational thinking: A competency whose time has come. In: S. Sentence, E. Barendsen, C. Schulte (Eds.) *Computer Science Education: Perspectives on teaching and learning in school*. London: Bloomsbury Academic, 19–37.

Kafai, Y.B. (1994). *Minds in Play: Computer Game Design as a Context for Children's Learning*.

Kozulin, A. (2003). Psychological tools and mediated learning. In: A. Kozulin, B. Gindis, S. Ageyev, Vladimir, and M. Miller, Suzanne (eds.), *Vygotsky's Educational Theory in Cultural Context*. Cambridge University Press, 15–38.

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

Passey, D. (2017). Computer Science (CS) in compulsory education curriculum: Implications for future research. *Education and Information Technologies*, 22(2), 421–443.

Piaget, J. (1970). *Genetic Epistemology*. New York: Columbia University Press.

Royal Society. (2017). After the reboot: computing education in UK schools. The Royal Society, 6–9 Carlton Terrace, London, SW1Y 5AG.

Rich, K.M., Strickland, C., Binkowski, T.A., Moran, C., Franklin, D. (2017). K-8 learning trajectories derived from research literature: Sequence, repetition, conditionals. In*: Proceedings of the 2017 ACM Conference on International Computing Education Research*. ACM, 182–190.

Rich, K., Strickland, C., Franklin, D. (2017). A literature review through the lens of computer science learning

goals theorized and explored in research. In: *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*. ACM, 495–500.
Teague, D., Lister, R. (2014.) Programming: reading, writing and reversing. In: *Proceedings of the 2014 Conference on Innovation and Technology in Computer Science Education*. 285–290.
Selby, C.C., Woollard, J. (2013). Computational thinking: the developing definition, 5–8.
Settle, A., Perkovic, L. (2010). *Computational Thinking across the Curriculum: A Conceptual Framework. Technical Reports*, Paper 13.
Standl, B. (2017). Solving everyday challenges in a computational way of thinking. In: *International Conference on Informatics in Schools: Situation, Evolution, and Perspectives*. Springer, Cham, 180–191.
Tedre, M., Denning, P. (2016). The long quest for computational thinking. In: *Proceedings of the 16th Koli Calling International Conference on Computing Education Research*. New York, NY: ACM, 120–129.
UCLA: Statistical Consulting Group. (2019). How can I calculate a Kappa Statistic for variables with unequal score Ranges? SPSS FAQ. Last accessed 8th April 2019. Available at `https://stats.idre.ucla.edu/spss/faq/how-can-i-calculate-a-kappa-statistic-for-variables-with-unequal-score-ranges/`
Voogt, J., Fisser, P., Good, J., Mishra, P., Yadav, A. (2015). Computational thinking in compulsory education: Towards an agenda for research and practice. *Education and Information Technologies*, 20(4), 715–728.
Waite, J., Curzon, P., Marsh, D., Sentence, S., Hawden-Bennett, A. (2018). Abstraction in action: K-5 teachers' uses of levels of abstraction, particularly the design level, in teaching programming. *International Journal Of Computer Science Education In Schools* (2018).
Waite, J. *et al*. (2016). Abstraction and common classroom activities. In: *Proceedings of the 11th Workshop in Primary and Secondary Computing Education*. ACM, 112–113.
Waite, J. *et al*. (2018). Abstraction in action: K-5 teachers' uses of levels of abstraction, particularly the design level, in teaching programming. *International Journal of Computer Science Education in Schools*, 2(1), 14–40.
Wing, J.M. (2006). Computational thinking. *Communications of the ACM*, 49(3), 33–35.
Wing, J.M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 366(1881), 3717–3725.
Yadav, A., Gretter, S., Good, J., & McLean, T. (2017). Computational thinking in teacher education. In: *Emerging Research*.
Yadav, A., Zhou, N., Mayfield, C., Hambrusch, S., Korb, J.T. (2011). *Introducing Computational Thinking in Education.*
Yang, S., Park, S. (2014). Teaching some informatics concepts using formal system. *Informatics in Education*, 13(2), 323–332.

A. Csizmadia, has an active interest in computer science education, and promotes this agenda at a local, regional and national level through projects, presenting and writing. His research interests include computational thinking, computational modelling and programming pedagogy, Andrew has published on these topics and has presented at both national and international conferences.

B. Standl is Junior professor for Computer Science Education at the Karlsruhe University of Education in Germany. His research interests are in the conceptualisation of teaching-learning scenarios and in problem-solving strategies based on computational thinking concepts.

J. Waite is undertaking a PhD with Queen Mary University of London looking at the teaching of design in primary programming. She is a qualified primary teacher, having ten years' experience in education and twenty years' experience in the IT industry. She also writes for cs4fn, Primary Computing and Cambridge International, delivers the Scratch taught element of the BCS Certificate in Computer Science Teaching, presents at national and local conferences and provides CPD on a wide range of computing topics.