Romero et al. International Journal of Educational Technology in Higher Education (2017) 14:42
DOI 10.1186/s41239-017-0080-z

International Journal of Educational Technology in Higher Education

RESEARCH ARTICLE Open Access

Computational thinking development through creative programming in higher education

Margarida Romero[1*], Alexandre Lepage[2] and Benjamin Lille[2]

* Correspondence: Margarida.Romero@unice.fr
[1]Laboratoire d'Innovation et Numérique pour l'Education, Université Nice Sophia Antipolis, Nice, France
Full list of author information is available at the end of the article

Abstract

Creative and problem-solving competencies are part of the so-called twenty-first century skills. The creative use of digital technologies to solve problems is also related to computational thinking as a set of cognitive and metacognitive strategies in which the learner is engaged in an active design and creation process and mobilized computational concepts and methods. At different educational levels, computational thinking can be developed and assessed through solving ill-defined problems. This paper introduces computational thinking in the context of Higher Education creative programming activities. In this study, we engage undergraduate students in a creative programming activity using Scratch. Then, we analyze the computational thinking scores of an automatic analysis tool and the human assessment of the creative programming projects. Results suggested the need for a human assessment of creative programming while pointing the limits of an automated analytical tool, which does not reflect the creative diversity of the Scratch projects and overrates algorithmic complexity.

Keywords: Computational thinking, Problem-solving, Creativity, Assessment

Creativity as a context-related process

Creativity is a key competency within different frameworks for twenty-first century education (Dede, 2010; Voogt & Roblin, 2012) and is considered a competency-enabling way to succeed in an increasingly complex world (Rogers, 1954; Wang, Schneider, & Valacich, 2015). Creativity is a context-related process in which a solution is individually or collaboratively developed and considered as original, valuable, and useful by a reference group (McGuinness & O'Hare, 2012). Creativity is also considered under the principle of parsimony, which occurs when one prefers the development of a solution using the fewest resources possible. In computer science creative parsimony has been described as a representation or design that requires fewer resources (Hoffman & Moncet, 2008). The importance or the usefulness of the ideas or acts that are considered as creative is highlighted by Franken (2007). These authors consider creativity as "the tendency to generate or recognize ideas, alternatives, or possibilities that may be useful in solving problems, communicating with others, and entertaining ourselves and others" (p. 348). In this sense, creativity is no longer considered a mysterious breakthrough, but a process happening in a certain context which can be fostered both by the activity

orchestration and enhanced creative education activities (Birkinshaw & Mol, 2006). Teachers should develop their capacities to integrate technologies in a reflective and innovative way (Hepp, Fernández, & García, 2015; Maor, 2017), in order to develop the creative use of technologies (Brennan, Balch, & Chung, 2014; McCormack & d'Inverno, 2014), including the creative use of programming.

From code writing to creative programming

Programming is not only about writing code but also about the capacity to analyze a situation, identify its key components, model the data and processes, and create or refine a program through an agile design-thinking approach. Because of its complexity, programming is often performed as a team-based task in professional settings. Moreover, professionals engaged in programming tasks are often specialized in specific aspects of the process, such as the analysis, the data modelling or even the quality test. In educational settings, programming could be used as a knowledge building and modeling tool for engaging participants in creative problem-solving activities. When learners engage in a creative programming activity, they are able to develop a modelling activity in the sense of Jonassen and Strobel (2006), who define modelling as "using technology-based environments to build representational models of the phenomena that are being studied" (p.3). The interactive nature of the computer programs created by the learners allows them to test their models, while supporting a prototype-oriented approach (Ke, 2014). Despite its pedagogical potential, programming activities must be pedagogically integrated in the classroom. Programming should be considered as a pedagogical strategy, and not only as a technical tool or as a set of coding techniques to be learnt. While some uses of technologies engage the learner in a passive or interactive situation where there is little room for knowledge creation, other uses engage the learner in a creative knowledge-building process in which the technology aims at enhancing the co-creative learning process (Romero, Laferrière & Power, 2016). As shown in the figure below, we distinguish five levels of creative engagement in computer programming education based on the creative learner engagement in the learning-to-program activity: (1) passive exposure to teacher-centered explanations, videos or tutorials on programming; (2) procedural step-by-step programming activities in which there is no creativity potential for the learner; creating original content through individual programming (3) or team-based programming (4), and finally, (5) participatory co-creation of knowledge through programming Fig. 1.

Creative programming engages the learner in the process of designing and developing an original work through coding. In this approach, learners are encouraged to use the programming tool as a knowledge co-constructing tool. For example, they can (co-)create the history of their city at a given historical period or transpose a traditional story in a visual programming tool like Scratch (http://scratch.mit.edu//). In such activities, learners must use skills and knowledge in mathematics (measurement, geometry and Cartesian plane to locate and move their characters, objects and scenery), Science and Technology (universe of hardware, transformations, etc.), Language Arts (narrative patterns, etc.) and Social Sciences (organization in time and space, companies and territories).

Computational thinking in the context of creative programming

We now expand on cognitive and metacognitive strategies potentially used by learners when engaged in creating programming activities: procedural and creative

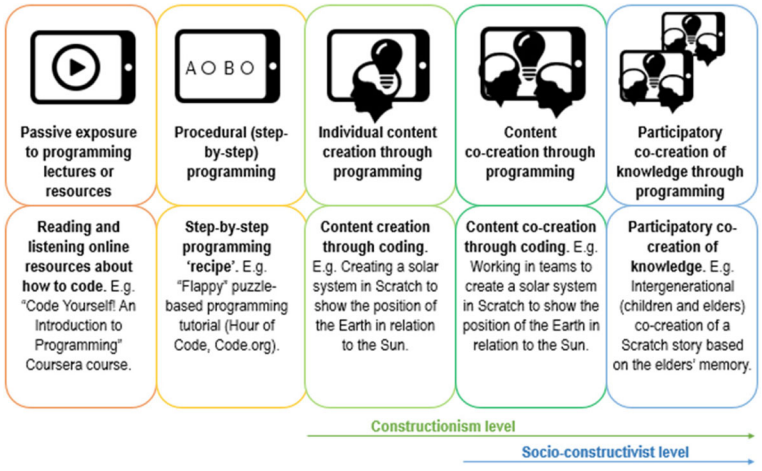

Fig. 1 Five levels of creative engagement in educational programming activities

programming. In puzzle-based coding activities, both the learning path and outcomes have been predefined to ensure that each of the learners is able to successfully develop the same activity. These step-by-step learning to code activities do not solicit the level of thinking and cognitive and metacognitive strategies required by ill-defined co-creative programming activities. The ill-defined situations embed a certain level of complexity and uncertainty. In ill-defined co-creative programming activities, the learner should understand the ill-defined situation, empathize (Bjögvinsson, Ehn, & Hillgren, 2012), model, structure, develop, and refine a creative program that responds in an original, useful, and valuable way to the ill-defined task. These sets of cognitive and metacognitive strategies could be considered under the umbrella of the computational thinking (CT) concept initially proposed by Wing (2006) as a fundamental skill that draws on computer science. She defines it as "an approach to solving problems, designing systems and understanding human behavior that draws on concepts fundamental to computing" (Wing, 2008, p. 3717). Later, she refined the CT concept 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" (Cuny, Snyder, & Wing, 2010). Open or semi-open tasks in which the process and the outcome are not decided can address more dimensions of CT than closed tasks like step-by-step tutorials (Zhong, Wang, Chen, & Li, 2016).

Ongoing discussion about computational thinking

The boundaries of computational thinking vary among authors. This poses an important barrier when it comes to operationalizing CT in concrete activities (Chen et al., 2017). Although some associate it strictly with the understanding of algorithms, others insist to integrate problem solving, cooperative work, and attitudes in the concept of CT. The identification of the core components of computational thinking is also discussed by Chen et al. (2017). Selby and Woollard (2013) addressed that problem and made a review of literature to propose a definition based on elements that are widely accepted: abstraction, decomposition,

evaluations, generalization, and algorithmic thinking. On the one hand, these authors' definition deliberately rejected problem solving, logical thinking, systems design, automation, computer science content, and modelling. These elements were rejected because they were not widely accepted by the community. On the other hand, other authors such de Araujo, Andrade, and Guerrero (2016, p.8) stress, through their literature review on the CT concept and components, that 96% of selected papers considered problem solving as a CT component. Therefore, we claim that the previously named components are relevant to the core of computational thinking and should be recognized as part of it.

Roots of the computational thinking concept

Following Wing (2006, 2008), Duschl, Schweingruber, Shouse, and others (2007) have described CT as a general analytic approach to problem solving, designing systems, and understanding human behaviors. Based on a socio-constructivist (Nizet & Laferrière, 2005), constructionist (Kafai & Resnick, 1996) and design thinking approach (Bjögvinsson et al., 2012), we consider learning as a collaborative design and knowledge creation process that occurs in a non-linear way. In that, we partially agree with Wing (2008), who considers the process of abstraction as the core of computational thinking. Abstraction is part of computational thinking, but Papert (1980, 1992) pointed out that programming solicits both concrete and abstract thinking skills and the line between these skills is not easy to trace. Papert (1980) suggests that an exposure to computer science concepts may give concrete meaning to what may be considered at first glance as abstract. He gives the example of using loops in programming, which may lose its abstract meaning after repeated use. If we expand that example by applying it to a widely-accepted definition of abstraction from the APA dictionary (i.e. "such a concept, especially a wholly intangible one, such as "goodness" or "beauty"", VandenBoss, 2006), we could envision a loop as something tangible in that it may be seen as such in the environment. The core of CT might be the capacity to transpose abstract meaning into concrete meaning. This makes CT a way to reify an abstract concept into something concrete like a computer program or algorithm. In this sense programming is a process by which, after a phase of analysis and entities identification and structuration, there is a reification of the abstract model derived from the analysis into a set of concrete instructions. CT is a set of cognitive and metacognitive strategies paired with processes and methods of computer science (analysis, abstraction, modelling). It may be related to computer science the same way as algorithmic thinking is related to mathematics.

> "Algorithmic thinking is a method of thinking and guiding thought processes that uses step-by-step procedures, requires inputs and produces outputs, requires decisions about the quality and appropriateness of information coming and information going out, and monitors the thought processes as a means of controlling and directing the thinking process. In essence, algorithmic thinking is simultaneously a method of thinking and a means for thinking about one's thinking." (Mingus & Grassl, 1998, p. 34).

Algorithmic thinking has to deal with the same problem as computational thinking: its limits are under discussion. To some authors it is limited to mathematics.

But definitions such as that from Mingus and Grassl (1998) make the concept go beyond mathematics (Modeste, 2012). Viewing algorithmic thinking as the form of thinking associated to computer science at large instead of part of mathematics allows a more adequate understanding of its nature (Modeste, 2012). We can consider that algorithmic thinking is an important aspect of computational thinking. However, when considering computational thinking as a creative prototype-based approach we should not only consider the design thinking components (exploration, empathy, definition, ideation, prototyping, and creation) (Brown, 2009) but also the hardware dimension of computational thinking solutions (i.e. use of robotic components to execute a program). Within a design thinking perspective, different solutions are created and tested in the attempts to advance towards a solution. From this perspective we conceptualize CT as a set of cognitive and metacognitive strategies related to problem finding, problem framing, code literacy, and creative programming (Brennan & Resnick, 2013). It is a way to develop new thinking strategies to analyze, identify, and organize relatively complex and ill-defined tasks (Rourke & Sweller, 2009) and as creative problem solving activity (Brennan et al., 2014). We now elaborate on how computational thinking can be assessed in an ill-defined creative programming activity.

Assessment of computational thinking

There is a diversity of approaches for assessing CT. We analyze three approaches in this section: Computer Science Teachers Association's (CSTA) curriculum in the USA (Reed & Nelson, 2016; Seehorn et al., 2011), Barefoot's computational thinking model in the UK (Curzon, Dorling, Ng, Selby, & Woollard, 2014), and the analytical tool Dr. Scratch (Moreno-León & Robles, 2015).

CSTA's curriculum includes expectation in terms of levels to reach at every school grade. It comprises five strands: (1) Collaboration, (2) Computational Thinking, (3) Computing Practice and Programming, (4) Computers and Communication Devices, and (5) Community, Global, and Ethical Impacts. Thus, CSTA's model considers computational thinking as part of a wider computer science field. The progression between levels appears to be based on the transition between low-level programming and object-oriented programming (i.e. computer programs as step-by-step sequences at level 1, and parallelism at level 3). CSTA standards for K12 suggest that programming activities in K12 should "be designed with a focus on active learning, creativity, and exploration and will often be embedded within other curricular areas such as social science, language arts, mathematics, and science". However, CSTA standards do not give creativity a particular status in their models; moreover, the evaluation of creativity in programming activities seems to be ultimately up to the evaluator. From our perspective, and because of the creative nature of CT, we need to consider creativity in an explicit way and provide educators with guidelines for assessing it.

The Barefoot CT framework is defined through five components: logic, algorithms, decomposition, patterns, abstraction, and evaluation. We agree on their relevance in computational thinking. Barefoot CT framework provides concrete examples of how each of the components may be observed in children of different ages. However, relying only on that model may result in assessing abilities instead of competency. These concepts represent a set of abilities more than an entire competency (Hoffmann, 1999).

Dr. Scratch is a code-analyzer that outputs a score for elements such as abstraction, logic, and flow control (Moreno-León & Robles, 2015). Scores are computed automatically from any Scratch program. It also provides instant feedback and acts as a tutorial about how one can improve his program, which makes it especially adequate for self-assessment. Hoover et al. (2016) believe that automated assessment of CT can potentially encourage CT development. However, Dr. Scratch only considers the complexity of programs, not their meaning. This tool is suitable to evaluate the level of technical mastery of Scratch that a user has, but it cannot be used to evaluate every component of a CT competency as we defined it (i.e. the program does not give evidence of thought processes, and does not consider the task demanded). Finally, it would be hard for an automated process to measure or teach creativity since that behaviour is an act of intelligence (Chomsky, 2008) which should be analyzed considering the originality, value, and usefulness for a given problem-situation. In other terms, there is a need to evaluate the appropriateness of the creative solution according to the context and avoid over-complex solutions, which use unnecessary or inappropriate code for a given situation. In automatic code-analyzers tools, it is impossible to rate creativity, parsimony and appropriateness of a program considering that ill-defined problem-situations could lead to different solutions. We therefore now elaborate on how computational thinking can be assessed while considering that CT is intertwined with other twenty-first century competencies such as creativity and problem-solving.

Computational thinking components within the #5c21

We consider CT as a coherent set of cognitive and metacognitive strategies engaged in (complex) systems identification, representation, programming, and evaluation. After identifying and analyzing a problem or a user need, programming is a creative problem-solving activity. The programming activity aims to design, write, test, debug, and maintain a set of information and instructions expressed through code, using a particular programming language, to produce a concrete computer program which aims to meet the problem or users' needs. Programming is not a linear predefined activity, but rather a prototype-oriented approach in which intermediate solutions are considered before releasing a solution which is considered good enough to solve the situation problem. Within this approach of programming, which is not only focused on the techniques to code a program, we should consider different components which are related to the creative problem-solving process. In this sense, we identify six components of the CT competency in the #5c21 model: two related to code and technologies literacies and four related to the four phases of Collaborative Problem Solving (CPS) of PISA 2015. Firstly, component 1 (COMP1) is related to the ability to identify the components of a situation and their structure (analysis/representation), which certain authors refer to as problem identification. Component 2 (COMP2) is the ability to organize and model the situation efficiently (organize/model). Component 3 (COMP3) is the code literacy. Component 4 (COMP4) is the (technological) systems literacy (software/hardware). Component 5 (COMP5) focuses on the capacity to create a computer program (programming). Finally, component 6 (COMP6) is the ability to engage in the evaluation and iterative process of improving a computer program Fig. 2.

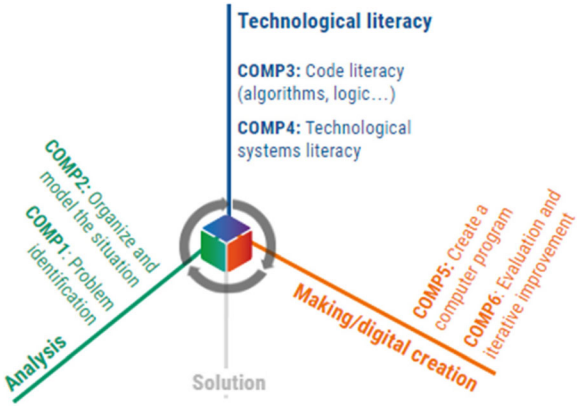

Fig. 2 Six components of the CT competency within the #5c21 framework

When we relate computational thinking components to the four phases of Collaborative Problem Solving (CPS) of PISA 2015, we can link the analysis/abstraction component (COMP1) to collaborative problem solving (CPS-A) Exploring and Understanding phase. model component (COMP2) is related to (CPS-B) representing/model component (COMP2) is related to (CPS-B) representing and formulating. The capacity to plan and create a computer program (COMP5) is linked to (CPS-C) planning and executing but also to (CPS-D) monitoring and reflecting (COMP6) Fig 3.

Code literacy (COMP3) and (technological) systems literacy (COMP4) are programming and system concepts and processes that will help to better operationalize the other components. They are also important in CT because knowing about computer programming concepts and processes can help develop CT strategies (Brennan & Resnick, 2013) and at the same time, CT strategies can be enriched by code-independent cognitive and metacognitive strategies of thinking represented by CPS related components (COMP1, 2, 5 and 6). Like in the egg-hen paradox, knowing about the concepts and process (COMP3 and 4) could enrich the problem-solving process (COMP1, 2, 5 and 6) and vice versa. The ability to be creative when analyzing, organizing/modeling, programming, and evaluating a computer program is a meta-capacity that shows that the participant had to think of different alternatives and imagine a novel, original and valuable process, concept, or solution to the situation.

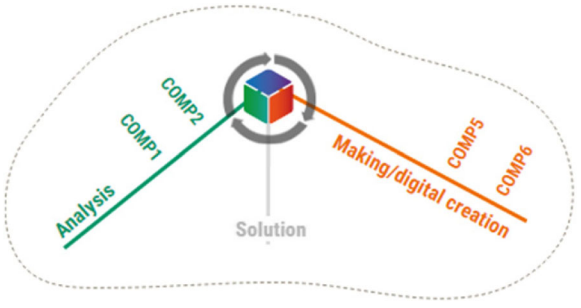

Fig. 3 Four components of the CT competency related to CPS of PISA 2015

Advancing creative programming assessment through the #5c21 model

After revising three models of CT assessment (CSTA, Barefoot, Dr. Scratch), we describe in this section our proposal to evaluate CT in the context of creative programming activities. We named our creative programming assessment the #5c21 model, because of the importance of the five key competencies in twenty-first century education: CT, creativity, collaboration, problem solving, and critical thinking. First, we discuss the opportunity of learning the object-oriented programming (OOP) paradigm from the early steps of CT learning activities. Second, we examine the opportunity to develop CT in an interdisciplinary way without creating a new CT curriculum in one specific discipline such as mathematics. Third, we discuss the opportunity of developing CT at different levels of education from primary education to lifelong learning activities.

In certain computer sciences curricula, low-level programming is introduced before OOP, which is considered as a higher-level of programming. Nevertheless, following Kölling (1999) if the OOP paradigm is to be learnt, it should not be avoided in the early stages of the learning activities to avoid difficulties due to paradigmatic changes. For that reason, our model does not restrict programming to step-by-step at early stages of development and embraces the OOP paradigm from its early stages. Moreover, we should consider the potential of non-programmers to understand OOP concepts without knowing how to operationalize it through a programming language. For instance, we may partially understand the concept of heritage through the concept of family without knowing the heritage concepts in computer science. Our model of a CT competency recognizes the possibility for certain components to develop at different rhythms, or for an individual with no prior programming experience to master some components (i.e. abstraction). For that reason, we did not integrate age-associated expectations. These should be built upon the context and should be task-specific. While CSTA considers concept mapping as a level 1 skill (K-3), our model would consider that this skill may be evaluated with different degrees of complexity according to the context and prior pupil's experience. Our view is that computational thinking encompasses many particular skills related to abstraction.

Our model pays attention to the integration of CT into existing curricula. We recognize the identification of CT related skills in the CSTA's model, and we agree to its relevance in computer science courses. However, our CT model is intended for use in any subject. Thus, it carefully tries not to give over relative importance to subjects such as mathematics and science. In that, we are working to define computational thinking as a transferable skill that does not only belong to the field of computer science. We also made it to be reusable in different tasks and to measure abilities as well as interactions between them (i.e. "algorithm creation based on the data modelling").

Our model of computational thinking is intended for both elementary and high school pupils. In that it differs from CSTA, which expects nothing in term of computational thinking for children under grade 4. Though CSTA expects K-3 pupils to "use technology resources […] to solve age-appropriate problems", some statements suggest that they should be passive in problem-solving (i.e.: "Describe how a simulation can be used to solve a problem" instead of creating a simulation, "gather information" instead of produce information, and "recognize that software is created to control computer operations" instead of actually controlling something like a robot).

Methodology for assessing CT in creative programming activities

In order to assess the CT from the theoretical framework and its operationalization as components described in the prior section, we have developed an assessment protocol and a tool (#5c21) to evaluate CT in creative programming activities. Before the assessment, the teacher defines the specific observables to be evaluated through the use of the tool. Once the observables are identified, four levels of achievement for each observable are described in the tool. The #5c21 tool allows a pre-test, post-test or just-in time teacher-based assessment or learner self-assessment which aims at collecting the level of achievement in each observable for the activity. At the end of a certain period of time (e.g., session and academic year) the teacher can generate reports showing the evolution in learners' CT assessment.

A distinctive characteristic of the #5c21 approach to assess CT is the consideration of ill-defined problem-situations. The creative potential of these activities engages the participants in the analysis, modelling and creation of artifacts, which may provide the teacher with evidence of an original, valuable, relevant, and parsimonious solution to a given problem-situation.

Participants

A total of 120 undergraduate students at Université Laval in Canada ($N = 120$) were engaged in a *story2code* creative challenge. All of them were undergraduate students of a bachelor's degree in elementary school education. They were in the third year of a four-year program and had no former educational technology courses. At the second week of the semester, they were asked to perform a programming task using Scratch. Scratch is a block-based programming language intended for children from 7 years of age. Participants were only presented two features of the language: creation of a new sprite (object) and the possibility to drag and drop blocks in each sprite's program. They were also advised about the use of the green flag to start the program.

Procedure

The ill-defined problem proposed to the students is rooted in the narrative frame of a children' book introducing basic concepts of programming and robotics, *Vibot the robot* (Romero & Loufane, 2016). The story introduces a robot, which has to be programmed for play. The Scratch Cat is the mascot of the visual programming tool Scratch and the default sprite appearing in each new project. Vibot is a fictional robot character, which waits for instructions to act in its environment. Based on these two characters, the *story2code* are short text-based stories, which engage learners in analyzing, modeling, and creating a Scratch project representing the story. In our study, participants were given a *story2code* and were asked to represent it in Scratch. The situation invited participants to create a Scratch project featuring a dialog between two characters: Scratch and Vibot. The students were given a text-based script for creating the dialog including 9 quotations in which Scratch and *Vibot the robot* introduce themselves. After the dialog between the two characters, Scratch asks Vibot to draw a blue line. The scenario of this *story2code* could be solved with a certain degree of diversity in the Scratch visual programming software. Participants were asked to remix a Scratch project containing the two characters' sprites. Remixing is a feature of Scratch that allows users to duplicate existing projects and edit them. Participants were required to share their projects in

order to develop a double assessment: the code-analyzer Dr. Scratch and the #5c21 assessment by an external evaluator.

Assessment of computational thinking

Computational thinking was assessed from a Scratch project developed by each undergraduate student using two different tools: Dr. Scratch and the #5c21 CT competency model. Firstly, all the Scratch projects were passed through the Dr. Scratch analytical tool. Then, they were evaluated by an evaluator following the #5c21 CT competency model. Dr. Scratch is an automated tool and has been selected in order to highlight the need of a competency-based approach in the assessment of CT.

Results

All participants had to submit a Scratch program by providing its URL. They were required to share it in order to make it accessible to an evaluator. In this section we highlight the results obtained from the Dr. Scratch analytical tool and those obtained using the #5c21 CT competency model.

CT assessment using Dr. scratch

Dr. Scratch results are computed from seven criteria: abstraction, parallelism, logic, synchronization, flow control, user interactivity, and data representation. Each of them may be given a maximum of three points, for a possible total score of 21. Thirteen projects have not been passed through Dr. Scratch due to technical problems (i.e. URL not provided), so we have Dr. Scratch's results for 107 projects ($n = 107$, $M = 0.27\%$; $ET = 0.06\%$) Fig. 4.

Ninety-one participants out of 107 got a total score of 6 at Dr. Scratch. The highest score was 10 and was reached by two participants. Instead of organizing the dialogs using timers (wait instruction), those two participants used broadcasting. Broadcasting is a feature of Scratch that allows the user to trigger events and program the events' handler (or listener) or each of the sprites (objects). Event handlers or listeners are callback subroutines which are able to react to certain inputs. Using the broadcasting feature caused Dr. Scratch to attribute additional points of parallelism and synchronization. One of these two highest-score projects received points for the use of a single event's handler or listener "when backdrop switches to …" in both parallelism and synchronization. Typical projects (those who scored 6) all worked about the same way: there was only one event's handler for each sprite (when green flag is clicked), and dialogs were synchronized using timers (*wait* instruction).

Assessment using #5c21 CT model

Results from the #5c21 computational thinking model are determined through some observables derived from the model, its components, and their subcomponents. These

Fig. 4 Automatic CT analysis by Dr. Scratch (0.251 ± 0.0184)

subcomponents were selected because of their relevance to the situation and the possibility to observe them in programs handed. Each of these subcomponents is converted into an observable that is task-specific. For instance, "Identification of entities" is a subcomponent of "Analysis/abstraction" (COMP1). It has been converted to an observable item specific to the task demanded: "Dialogs are well-integrated and the blue line is traced". When applicable, these observables were rated twice: one time for the level of execution and a second time for the level of creativity. These four subcomponents were converted into observables: "Identification of entities" (plus creativity assessment), "Identification of events" (no creativity assessment), "Identifying the function (or code block) for a certain objective" (with creativity assessment), and "Analysis of errors leading to improvement of the computer program" (no creativity assessment). All of them were rated on a 4-point scale. When applicable, creativity was also assessed on a 4-point scale. That makes a possible total score of 24 points (4 points for each of the 4 observables, plus 8 points of creativity). Thirty-three participants have no score using the #5c21 CT competency model due to technical problems (i.e. the Scratch project was not shared), so we have results for 87 participants. The two highest-score projects using the #5c21 CT competency model are not the same as the two from Dr. Scratch's results. Their Dr. Scratch's scores are 8 and 6, and their CT competency score are respectively 22 and 23 ($n = 87$, $M = 0.64\%$, $ET = 0.3\%$) Fig. 5.

Only 11 projects where given a score higher than 1 (in a 4-point scale) in creativity for "Identification of entities". The two aforementioned projects scored 3 points out of 4 and 2 points out of 4 in creativity for that subcomponent. In the first project, creativity was assessed from the untaught use of sounds (instruction "play sound") and the relevance of sounds chosen (i.e. the cat says "meow" and the robot makes "laser sounds"). Using irrelevant sounds would not have been considered an evidence of creativity because of the principle of parsimony, which aims to value the use of the fewer resources possible when solving a given situation through a creative solution. Creativity in the second project was assessed through the use of a loop to make the cat walk (using a change in costumes and delays). That was considered a higher level of modelling by the expert evaluators, since the walking is more realistic than a translation. Because participants had no prior experience with Scratch, using untaught blocks, such as broadcasting, was considered by the evaluator as an evidence of creativity.

Dr. scratch and #5C21 CT assessment differences

The analysis of CT based on the automatic Dr. Scratch analysis and the human expert #5c21 CT model lead to important differences. While the automatic analysis of the Scratch projects leads to similar scores in terms of algorithmic complexity (with a low standard deviation, ET = 0.0184), the expert analysis shows a high diversity in the creative programming performance (ET = 0.0551) Fig. 6.

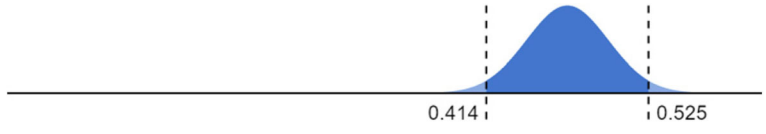

Fig. 5 #5c21 CT expert analysis (0.469 ± 0.0551)

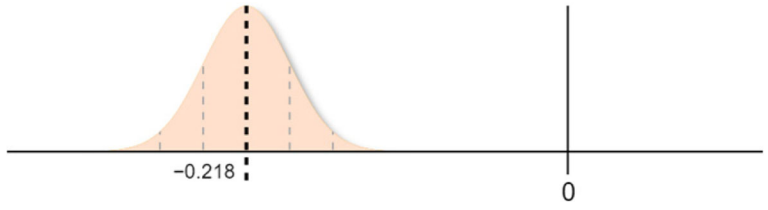

Fig. 6 Difference of means (d = −0.218, SE = 0.0294, $p < 0.001$)

Discussion on creative diversity assessment

Even considering the simplicity of the task demanded when solving the *story2code*, each Scratch project submitted by the students was different. Each of the 120 projects created by the undergraduate students was original. Despite the simplicity of the task demanded (to create a Scratch project featuring a dialog between two characters), none of the projects was identical to another. For instance, the project differences might come from the choice of blocks, the order in which they are placed, the duration of timers, the type of events' handlers and their operationalization, or the use of optional features such as backdrops switches. However, results from an automated analytical tool like Dr. Scratch do not reflect that wide creative diversity. The model we proposed is based on both computer science and problem-solving as defined by PISA 2015. It is an attempt to define criteria that may be suitable to evaluate ill-defined tasks involving CT. Results from Dr. Scratch and from the #5c21 CT competency model are not to be compared on any basis as they do not evaluate the same components. Dr. Scratch evaluates the algorithmic complexity of a Scratch project based on a unique model of CT assessment which is generic and does not consider the program in relation to the situation problem. In that, it is not intended for use when the aim is to evaluate CT as a creative problem-solving competency. However, the use of Dr. Scratch is a useful tool for allowing the learners to reflect on the algorithmic implementation and could provide useful tips for improving the quality of the program. Dr. Scratch is able to identify missing names of instances, repetitions, and some coding practices that could be improved. Also, the automatic analysis allows a generalized use without requiring a human activity of evaluation. The #5C21 CT model is based on humans with a certain knowledge of CT to carry on an assessment which is focused not only on the algorithmic properties of the program, but considers also the creative process by which the learner has developed a valuable, original and parsimonious solution to a specific situation.

Contribution of the #5c21 model for creative programming

The #5c21 CT model is not language-specific and could serve to evaluate different types of creative programming activities that can be developed in different computer languages but also within unplugged activities. Compared to other CT conceptualizations, we explicitly integrate a hardware component (COMP 4), which could be part of the creative solution in a creative programming task. The results on the evaluation of the *story2code* task suggest the pertinence of combining automatic code analysis tools and human expertise assessment on the creative aspects of programming. By developing this study, we intended to advance the CT competency conceptualization and assessment; from a creative programming perspective, we should critically consider the possibility to assess human creativity using automated tools such Dr. Scratch.

Creative programming activities through the lens of the zone of proximal development

The wide diversity of projects collected brings us to think about the possible application of the Vygotskian concept of Zone of Proximal Development (Vygotsky, 1978). To place an individual in a situation of competency, the situation proposed to the learner must offer an appropriate degree of newness, a certain ambiguity or ill-definition and the creativity potential to engage the learner in a creative process where there is not only a single process or solution to accomplish, but a creative scope of processes and solutions. In this sense, creative programming activities should engage learners in problem situations where the process and the solution are not known in advance and could be very diverse in order to allow the learners to develop their own creative process and solution. We should, at the same time, recognize the need to design programming activities with an adequate potential for their creative activity. In that way, the ill-defined problem-situation could be analyzed while allowing learners to create and implement a solution When activities are too externally guided or structured, there is no room for creativity, while too much ambiguity in the ill-defined situation could lead to uncertainty and confusion. Meanwhile, there is the Zone of Proximal Creativity (ZPC). The ZPC describes an appropriate level of creative potential to be developed by the learner when engaged in an activity that allows an appropriate potential of creativity during the development of a creative solution, which is original, valuable, useful, and parsimonious for a given situation and context. From the observations of this study, we highlight the value not only of developing the CT competency by considering creative-enough programming activities within the ZPC, but also of encouraging ambiguity tolerance among learners in order to embrace ill-defined situations as an opportunity to express their creativity.

Limits of the study and future research directions

While the results and insights of this study contributes in offering a better understanding on creative and context-related implementation of programming in education, we also want to point out that this study was focused on a *story2code* task based on a dialogue between two characters followed by an instruction to draw a line. This *story2code* task offers different degrees of creative potential to be solved while being simple to achieve. The simplicity of this task could have had an influence on the creative expression of the students and we should develop further studies in which more complex tasks are analysed in relation to the creative expression in order to identify the influence of the degree of complexity of the task on creative programming. The present study is also limited to a very specific task involving undergraduate students with no prior experience in programming. Future research should therefore analyze CT skills in more complex and open activities in order to deepen our understanding on how CT skills are deployed in an ill-defined creative programming task. We advocate the need for research with a wider range of learners in order to better understand how CT components may show or develop across a lifespan and through different creative programming activities including not only Scratch but also other technological supports from mobile based programming to educational robotics devices aiming to engage the learner in creative programming.

Acknowledgements
We acknowledge the contribution of John Teye for his advice during the linguistic revision.

Funding
This project has been funded by the Fonds de recherche du Québec – Société et culture (FRQSC).

Authors' contributions
All persons who meet authorship criteria are listed as authors (RLL), and all authors (RLL), certify that we have participated sufficiently in the work to take public responsibility for the content, including participation in the concept, design, analysis, writing, or revision of the manuscript. All authors read and approved the final manuscript.

Competing interests
All authors (Romero, Lepage, Lille) declare that we have no competing financial, professional or personal interests that might have influenced the performance or presentation of the work described in this manuscript.

Publisher's Note

Author details
[1]Laboratoire d'Innovation et Numérique pour l'Education, Université Nice Sophia Antipolis, Nice, France. [2]Université Laval, Québec, Canada.

Received: 14 June 2017 Accepted: 20 November 2017
Published online: 12 December 2017

References

Birkinshaw, J. M., & Mol, M. J. (2006). How management innovation happens. *MIT Sloan Management Review*, *47*(4), 81–88.
Bjögvinsson, E., Ehn, P., & Hillgren, P.-A. (2012). Design things and design thinking: Contemporary participatory design challenges. *Design Issues*, *28*(3), 101–116.
Brennan, K., Balch, C., & Chung, M. (2014). *Creative computing*. Harvard University Press: Cambridge. Retrived from http://scratched.gse.harvard.edu/guide/
Brennan, K., & Resnick, M. (2013). Imagining, creating, playing, sharing, reflecting: How online community supports young people as designers of interactive media. In *Emerging technologies for the classroom* (p. 253–268). New York: Springer.
Brown, T. (2009). *Change by design. How design thinking transforms organizations and inspires innovation*. New York, NY, USA: Harper Collins.
Chen, G., Shen, J., Barth-Cohen, L., Jiang, S., Huang, X., & Eltoukhy, M. (2017). Assessing elementary students' computational thinking in everyday reasoning and robotics programming. *Computers & Education*, *109*, 162–175.
Chomsky, N. (2008). *Language and mind*, (3rd ed.,). Cambridge: Cambrdige University Press.
Cuny, J., Snyder, L., & Wing, J. M. (2010). Demystifying computational thinking for non-computer scientists. Unpublished Manuscript in Progress, Referenced in https://www.cs.cmu.edu/link/research-notebook-computational-thinking-what-and-why.
Curzon, P., Dorling, M., Ng, T., Selby, C., & Woollard, J. (2014). Developing computational thinking in the classroom: A framework.
de Araujo, A. L. S. O., Andrade, W. L., & Guerrero, D. D. S. (2016). A systematic mapping study on assessing computational thinking abilities. In *Frontiers in education conference (FIE), 2016 IEEE*, (pp. 1–9). IEEE.
Dede, C. (2010). Comparing frameworks for 21st century skills. In J. A. Bellanca, & R. S. Brandt (Eds.), *21st century skills: Rethinking how students learn*, (vol. 20, pp. 51–76). Bloomington, IN: Solution Tree Press.
Duschl, R. A., Schweingruber, H. A., & Shouse, A. W. (2007). *Taking science to school: Learning and teaching science in grades K-8. National Academies report*. Washington, DC: National Academies Press.
Franken, R. E. (2007). *Human motivation* (6th ed). Belmont, CA: Thomson/Wadsworth.
Hepp, P., Fernández, M. À. P., & García, J. H. (2015). Teacher training: Technology helping to develop an innovative and reflective professional profile. *International Journal of Educational Technology in Higher Education*, *12*(2), 30–43.
Hoffman, R. N., & Moncet, J.-L. (2008). All Data are Useful, but not All Data are Used! What'S Going on Here? In *Geoscience and Remote Sensing Symposium, IGARSS 2008* (p. II-1-II-4). Boston, MA: IEEE. https://doi.org/10.1109/IGARSS.2008.4778912.
Hoffmann, T. (1999). The meanings of competency. *Journal of European Industrial Training*, *23*(6), 275–286.
Hoover, A. K., Barnes, J., Fatehi, B., Moreno-León, J., Puttick, G., Tucker-Raymond, E., & Harteveld, C. (2016). Assessing computational thinking in students' game designs. In *Proceedings of the 2016 annual symposium on computer-human interaction in play companion extended abstracts*, (pp. 173–179). ACM.
Jonassen, D., & Strobel, J. (2006). Modeling for meaningful learning. In *Engaged learning with emerging technologies* (p. 1–27). Dordrecht: Springer.
Kafai, Y. B., & Resnick, M. (1996). *Constructionism in practice: Designing, thinking, and learning in a digital world*. (Vol. 1). New York, NY: Routledge.
Ke, F. (2014). An implementation of design-based learning through creating educational computer games: A case study on mathematics learning during design and computing. *Computers & Education*, *73*, 26–39.
Kölling, M. (1999). The problem of teaching object-oriented programming. *Journal of Object Oriented Programming*, *11*(8), 8–15.
Maor, D. (2017). Using TPACK to develop digital pedagogues: a higher education experience. *Journal of Computers in Education*, *4*(1), 71–86.
McCormack, J., & d'Inverno, M. (2014). On the future of computers and creativity. In *AISB 2014 Symposium on Computational Creativity, London*.
McGuinness, C., & O'Hare, L. (2012). Introduction to the special issue: New perspectives on developing and assessing thinking: Selected papers from the 15th international conference on thinking (ICOT2011). *Thinking Skills and Creativity*, *7*(2), 75–77 https://doi.org/10.1016/j.tsc.2012.04.004.
Mingus, T. T. Y., & Grassl, R. M. (1998). Algorithmic and recursive thinking - current beliefs and their implications for the future. In L. Morrow, & M. J. Kenney (Eds.), *The teaching and learning of algorithm in school mathematics*, (pp. 32–43).

Modeste, S. (2012). La pensée algorithmique : Apports d'un point de vue extérieur aux mathématiques. Presented at the Colloque espace mathématique francophone.
Moreno-León, J., & Robles, G. (2015). Dr. scratch: A web tool to automatically evaluate scratch projects. In *Proceedings of the workshop in primary and secondary computing education*, (pp. 132–133). ACM.
Nizet, I., & Laferrière, T. (2005). Description des modes spontanés de co-construction de connaissances: contributions à un forum électronique axé sur la pratique réflexive. *Recherche et Formation*, *48*, 151–166.
Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. Inc: Basic Books.
Papert, S. (1992). *The Children's machine*. New York: BasicBooks.
Reed, D., & Nelson, M. R. (2016). Current initiatives and future directions of the computer science teachers association (CSTA). In *Proceedings of the 47th ACM technical symposium on computing science education*, (pp. 706–706). ACM.
Rogers, C. R. (1954). Toward a theory of creativity. *ETC: A Review of General Semantics*, *11*, 249–260.
Romero, M., Laferriere, T., & Power, T. M. (2016). The move is on! From the passive multimedia learner to the engaged co-creator. *eLearn*, *2016*(3), 1.
Romero, M., & Loufane (2016). *Vibot the robot*. Québec, QC: Publications du Québec.
Rourke, A., & Sweller, J. (2009). The worked-example effect using ill-defined problems: Learning to recognise designers' styles. *Learning and Instruction*, *19*(2), 185–199.
Seehorn, D., Carey, S., Fuschetto, B., Lee, I., Moix, D., O'Grady-Cunniff, D., … Verno, A. (2011). CSTA K–12 computer science standards: Revised 2011.
Selby, C. C., & Woollard, J. (2013). Computational thinking: The developing definition. In *Presented at the 18th annual conference on innovation and Technology in Computer Science Education, Canterbury*.
VandenBos, G. R. (Ed.). (2006). APA dictionary of psychology. Washington, DC: American Psychological.
Voogt, J., & Roblin, N. P. (2012). A comparative analysis of international frameworks for 21st century competences: Implications for national curriculum policies. *Journal of Curriculum Studies*, *44*(3), 299–321.
Vygotsky, L. S. (1978). *Mind and society: The development of higher mental processes*. Cambridge, MA: Harvard University Press.
Wang, X., Schneider, C., & Valacich, J. S. (2015). Enhancing creativity in group collaboration: How performance targets and feedback shape perceptions and idea generation performance. *Computers in Human Behavior*, *42*, 187–195.
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 of London A: Mathematical, Physical and Engineering Sciences*, *366*(1881), 3717–3725.
Wing, J. (2011). Research notebook: Computational thinking-What and why? The Link Newsletter, *6*, 1–32. Retrieved from http://link.cs.cmu.edu/files/11-399_The_Link_Newsletter-3.pdf.
Zhong, B., Wang, Q., Chen, J., & Li, Y. (2016). An exploration of three-dimensional integrated assessment for computational thinking. *Journal of Educational Computing Research*, *53*(4), 562–590.