J. Comput. Educ. (2016) 3(4):377–394
DOI 10.1007/s40692-016-0076-z

A framework of curriculum design for computational thinking development in K-12 education

Siu-Cheung Kong[1]

Received: 13 November 2016 / Revised: 21 November 2016 / Accepted: 21 November 2016 /
Published online: 28 November 2016

Abstract To respond to the growing integration of digital technologies across all sectors of society, a curriculum should be developed to nurture the next generation as creative problem solvers in order to see the world through a computational lens. One way to achieve this goal is to design a curriculum in K-12 to promote computational thinking (CT) through programming. In order to facilitate the design of the CT curriculum, the expected learning outcomes of the curriculum are proposed in this study. The CT learning outcomes of this study compose of CT knowledge, practices, and perspectives. Based on the proposed CT learning outcomes and interest-driven creator theory, this article aims to propose a seven-principle framework for guiding the design of K-12 CT curriculum. The first three principles ensure CT skills and perspectives are delivered in the curriculum through a programming environment that fosters CT knowledge acquisition. The other four principles are the design strategies for CT development: provide incrementally complex computational tasks across all levels of the curriculum to develop CT skills; review each level of the curriculum by producing final project samples to ensure a comprehensive coverage of CT knowledge; design the computational tasks that are of interest to the target learners to nurture interest-driven creator; and establish appropriate assessment criteria for the final projects and showcase their productions to enhance learners' creativity. The future work is to design, implement, and evaluate CT curriculum underpinned by these seven principles in K-12.

Keywords Computational thinking (CT) · Interest-driven creator (IDC) theory · Programming education · Curriculum design · K-12 education

✉ Siu-Cheung Kong
sckong@eduhk.hk

[1] The Education University of Hong Kong, 10 Lo Ping Road, Tai Po, New Territories, Hong Kong

Introduction

The digital world has a growing need for creator to solve problems in all walks of life. They have to be able to adapt existing disciplinary practices with digital technologies. In a bid to imbue future generations with creativity and problem-solving skills that can be seamlessly integrated with digital technologies, there is increasing necessity to develop computational thinking (CT) in young people. Over the past three decades, CT has gained extensive attention and become accepted as one of the skills required by those growing up in the digital era. In the digital world, thinking like a computer scientist is fundamental and beneficial for innovation and advancement (Wing 2006). This creates a need to facilitate young peoples' building of CT. It is believed that K-12 learners should be provided with opportunities to develop CT through programming (Barr and Stephenson 2011; Grover and Pea 2013; Lye and Koh 2014; Seiter and Foreman 2013). However, in the existing literature, the studies on CT development through programming in K-12 education is far from sufficient (Grover and Pea 2013; Lye and Koh 2014) because the importance of CT development has mostly been emphasized in tertiary education, but seldom in primary and secondary levels, or K-12. The review by Lye and Koh (2014) reflects that more research should have been conducted on the implementation of a proper curriculum in K-12 for CT development. Calls for the integration of CT into K-12 education give rise to the need for theory-based, tested, and successful approaches to curriculum design. This article discusses one approach to constructing curricula that help foster CT development in K-12 education with an example of a 3-year curriculum.

Background

Interest-driven creator theory

In the fast-changing 21st century where creativity is highly valued as an essential competence, the next generation should best be nurtured as creator to deal with intricate problems in every walk of life. In order to sustainably develop an innovative mindset in young people, interest is the ideal motivation as it "leads to meaningful learning, promotes long-term storage of knowledge, and provides motivation for further learning" (Schiefele 1991). The interest-driven creator (IDC) theory postulates that learners, given the technological support, are capable of becoming lifelong creator through the immersion in the interest-driven learning process (Chan et al. 2015). It proposes a holistic developmental framework "to guide learners in fostering their learning *interests*, capabilities in *creation*, and learning *habits*" (Wong et al. 2015). These are the "anchor concepts" of the IDC theory and each of them forms a three-component loop.

The interest loop is composed of triggering interest, immersing interest, and extending interest (Wong et al. 2015). To trigger learners' initial interests in learning about a topic, their curiosity has to be aroused by interesting activities. To maintain interest for the long run, immersing interest helps learners emotionally

engage in the learning process. Finally, learners' domain interest can be extended by encouraging them to immerse themselves in the content and to re-engage in similar activities in the future. The creation loop involves imitating, combining, and staging (Chan et al. 2015). Learners first mimic others' behaviors through observation before developing new perspectives through this combination of others' ideas and works. Later, they can create and stage their performance by sharing their new ideas and creations with other people. In the long term, learners are expected to develop a habit of creating, which, in the IDC theory, is the start of the habit loop. It includes three components, namely a cueing environment, routine, and satisfaction (Chen et al. 2015). The whole process starts from the cue, which can trigger a person's behavior, and will gradually become a habit through repetition. The newly formed habit will give the person a sense of satisfaction and achievement, which may help "the repetition of the new behavior in the future" (Chen et al. 2015). Together, the three concepts of the IDC theory, i.e., interests, creation, and habit, provide a theoretical ground for guiding the design of the curriculum to facilitate the interest-driven CT development through programming in K-12 schools.

Expected learning outcomes of computational thinking curriculum in K-12

In order to provide a guide to the design of K-12 CT curriculum, the expected learning outcomes of the CT curriculum have to be defined. Popularized by Wing (2006), the term computational thinking refers to "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" (Wing 2011). In other words, CT relates to problem solving, system design, and the comprehension of human behavior by means of drawing on the fundamental knowledge and skills of computer science. Since there is not any commonly agreed definition of CT, developing a suitable one for K-12 can be a challenging task (Barr and Stephenson 2011; Brennan and Resnick 2012; Grover and Pea 2013; Lye and Koh 2014), thus resulting in CT frameworks with different focus. In the review by Lye and Koh (2014), most of the previous studies on CT framework in K-12 have put their emphasis on CT concepts, while some other works focus on the development of CT practices (e.g., Curzon et al. 2014; Grover and Pea 2013). The CT framework constructed by Brennan and Resnick (2012) covers CT concepts, practices, and perspectives and is adopted in this article as the basis for defining the expected learning outcomes of K-12 CT curriculum due to its wide coverage of CT (see Table 1).

Since the CT curriculum in this study is to nurture creative problem solvers for them to interpret the world with a computational mindset, the learning outcomes of this curriculum have to be enriched by emphasizing the importance of problem solving in the real world in the three dimensions of the CT framework by Brennan and Resnick (2012). Figure 1 shows the expected learning outcomes in the dimensions of CT knowledge, CT practices, and CT perspectives in this study. In the dimension of CT knowledge, while Brennan and Resnick (2012) place their emphasis on the concepts in a programming language context, this study broadens this dimension by including both fundamental programming concepts as well as

Table 1 The CT framework developed by Brennan and Resnick (2012)

Dimensions	Description	Examples
Computational concepts	The concepts applied in programming	Events Operators Parallelism Sequences Conditionals Loops Data
Computational practices	The problem-solving practices arising in programming	Being incremental and iterative Testing and debugging Reusing and remixing Abstracting and modularizing
Computational perspectives	The views that programmers hold about themselves, their relationships to others, and the technological world around them	Expressing Connecting Questioning

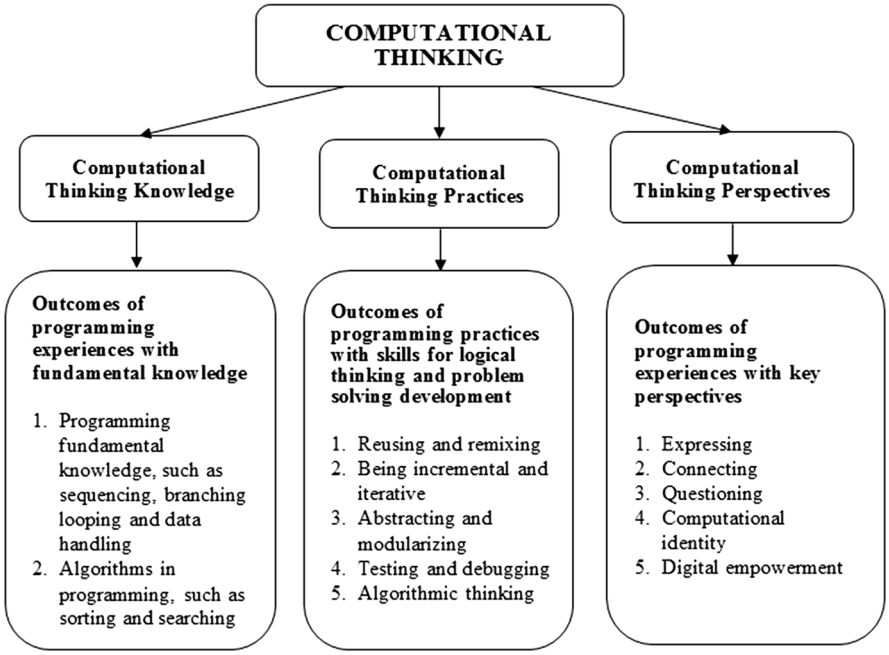

Fig. 1 Expected learning outcomes of computational thinking curriculum in K-12

simple algorithms in programming. It is because elementary algorithms such as sorting and searching are indispensable in solving real-world computational problems. With this regard, the term CT knowledge is used in this study to replace

"computational concepts" since the meaning of "knowledge" is more embracive that covers facts, concepts, and procedures in the programming context in this study (Clark and Mayer 2007). In the dimension of CT practices, the proposed outcomes are the practices involved in the process of dealing with computational problems. The list of CT practices in this study not only adopts those in Brennan and Resnick's work (2012) but also includes an additional skill—algorithmic thinking. It is a problem-solving competence composed of various intellectual abilities, such as specifying problems, analyzing problems, finding the fundamental actions for the problems (Futschek 2006). Also, it has been perceived as one of the essential components in CT (Aho 2012; Barr et al. 2011; Grover and Pea 2013; Selby and Woollard 2013; Wing 2011), which implies the necessity of including algorithmic thinking as one of the CT skills in the learning outcomes of this study. In the dimension of CT perspectives, extended from the original three perspectives in Brennan and Resnick's work (2012), computational identity and digital empowerment are the proposed learning outcomes in CT perspectives in this study as they are the important attributes for achieving the goal of nurturing creative problem solvers in the digital world. Every individual, especially the young generation, is expected to be digitally comfortable and competent so as to maintain their competitive power, and also to be willing to contribute to social enhancement by solving problems creatively with digital technologies. It is therefore important for learners to experience expressing, connecting, and questioning in the process of programming. Through these experiences, they are expected to establish a computational identity and achieve digital empowerment. Computational identity emphasizes the individual and collective perspectives of self-representations through networking and knowledge–experience sharing in computational activities with other in-group members. The acquirement of digital empowerment through education is an inevitable means to equip learners to become influential members in the digital community. These learning outcomes serve as an important guide to orient the curriculum to CT development in K-12 education.

A framework of curriculum design for computational thinking development

This study proposes an outcome-based curricular design framework for CT development in K-12 education. It is a design framework that provides curriculum designers with a set of principles to focus on and organize their design. The outcome-based curricular design framework emphasizes the goals for learners to achieve at the end of the learning path (Spady 1994). Therefore, the curriculum design should begin with the end in mind, which means it should start from the final outcomes to be obtained by learners. Other fundamental elements of the curriculum, such as content, teaching strategies, and assessments, can then be designed based on these desired outcomes (Spady and Marshall 1991). In this study, the objective of the programming curriculum is to develop learners' CT skills and perspectives in the process of learning CT knowledge, which constitute the first 3 design principles. Based on these CT components, four strategies are employed to structure the

curriculum for fostering the CT development. First, complex computational problems should be designed for learners to tackle by a top–down strategy. Second, a comprehensive coverage of CT knowledge is essential in each level for producing meaningful final projects with a reasonable scope of the CT knowledge learnt at the level. Third, interest-driven learning activities should be embedded in the curriculum for nurturing creativity. Finally, the appropriate assessment criteria of the final projects and a showcase of the final projects are necessary to motivate learners' creativity. Since the acquirement of CT is a stage-by-stage process that needs a long period of time, this study illustrates the framework with an example of a 3-year curriculum, such as grade 4–6 in primary education. The details of the framework for curriculum design are discussed in the following sections.

Use computational thinking knowledge as the basis of the design

There is a set of fundamental CT knowledge that are essential for all learners of computer programming at each level of the curriculum: events, sequences, repetition, conditional, parallelism, variables, operators, manipulation of data, elementary data structures, and simple algorithms like sorting and searching. Such CT knowledge needs to be introduced according to their difficulty levels. Those simpler ones, like sequences, events, repetitions, and conditions, can be taught earlier, while the rest should be introduced later. This CT knowledge will be repeatedly used throughout the activities in all levels of the curriculum.

It is critical that learners' understanding of CT knowledge is enhanced by using appropriate tasks and tools in their programming activities. The learning tasks and tools listed in Table 2 are sample content designed for building mobile Apps in block programming environments such as App Inventor in the sample curriculum. These tasks and tools will form the basis of the design of the learning activities across all levels in the 3-year curriculum.

Two main criteria are suggested for curriculum designers to select the suitable tasks and tools for teaching CT knowledge. First, the complexity of the tasks and tools at each level should be appropriate to the learners' competence. At level 1, the basic tasks and tools should be selected for programming beginners. The

Table 2 Examples of learning tasks and tools in App Inventor for covering at the three levels of the computational thinking curriculum

Level 1 curriculum	Level 2 curriculum	Level 3 curriculum
Music	Connecting to the web	Google chart API
Drawing	Maps and location aware programs	Communication and the web
Animation	Accelerometer	Bluetooth
Camera/video	Orientation sensor	Firebase
TextToSpeech	TinyDB	SpeechToText
Translation	Bar code scanner	Concepts of internet of things (IoT)

basic tasks involve fewer blocks and steps for writing Apps. The tools are easy for learners to use with a few constraints in using tools such as music and drawing. The basic CT knowledge learnt at this initial level provides a fundamental understanding of programming for learners to move forward to the advanced levels. At level 2, the tasks and tools should be more complex in the sense that it requires learners to combine what they previously learnt with the new knowledge from level 2. For example, connecting to the web and use of sensors are the possible choices for the advanced level. At level 3, the tasks and tools should involve communication—interactions with one another through mobile apps. It is a critical element in the activity design because the value of developing apps with mobile technologies lies in the fostering of communication. To fulfill the objective at this level, the apps that learners are required to design should encourage interactions with their peers. For instance, Firebase can be used in designing apps since it enables two or more people to simultaneously draw the same picture using their individual mobile devices. To summarize, curriculum designers are expected to arrange the content across levels in a way that there is increasing complexity in the tasks and tools that are considered appropriate to learners at each level. These learning tasks and tools will pave the way for learners to experience CT practices with increasing difficulty level. Thus, learners will be able to develop CT perspectives with increasing maturity.

Second, the learning tasks and tools at each level should be able to assist learners in producing interesting and meaningful Apps. As the ultimate goal of the curriculum is to nurture interest-driven creator, the learning activities have to be made interesting and meaningful so as to trigger and extend their interest (Wong et al. 2015). The selected tasks and tools in Table 2 are interesting to K-12 learners as they can produce various types of games with music, drawing, and animation, for example, a maze game. They are also of practical use, like maps and location aware programs and Bluetooth, which can create useful mobile apps for a map tour and communication.

Putting computational thinking skills as the core of the design

Learning the above CT knowledge is not the only goal of the programming activities in the CT curriculum. Rather, they serve as a channel for learners to embark on learning CT skills which are the core of developing problem-solving abilities. Lye and Koh (2014) suggested designing a constructionism-based problem-solving learning environment in which the learning activities should be designed for intentionally acquiring problem-solving skills. Therefore, each activity in the curriculum should guide learners to develop CT skills with problem-solving situations.

Five sets of CT skills should be introduced to learners through learning activities. The first one is the skills of reusing and remixing codes. Learners are able to recall and apply what they have learnt from previous activities, and build on their own or others' work to create something new and more complex. Curriculum designers should carefully consider how to interconnect the learning activities to enable learners to reuse and remix the codes. The second one is to be incremental and iterative. It means a repeated cycle of imagining and developing. Learners build

solutions step-by-step, try them out, and then further develop (Brennan and Resnick 2012). The activities in the CT curriculum should be increasingly complex, which means the learning activities should not be only interrelated but also built upon one another. This can provide room for learners to develop solutions progressively and repeatedly by making use of what they have previously learnt. The third one is abstraction and modularization. Abstraction is applied in defining patterns, generalization from instances, and parameterization, and is essential to deal with complexity and scale (Wing 2011). Modularization is decomposition of complex problems, which helps structure large-grain program (Parnas 1972). To enhance learners' problem-solving skills in dealing with the complex computational problems, it is essential to design activities that help them to manage complexity by introducing abstraction and modularization. The fourth one is testing and debugging. Learners should be given opportunities to develop tactics for predicting and tackling problems through "trial and error, transfer from other activities, or support from knowledgeable others" (Brennan and Resnick 2012). They should also learn from other's mistakes by proofreading their codes. Learners should be provided with erroneous programs to practice the skill of debugging so that they may understand the value of debugging as a skill to help solve computational problems. Additionally built on Brennan and Resnick's framework (2012), algorithmic thinking is an important problem-solving skill which assists learners in learning how to formulate computational problems and systematically process information and gain a deeper understanding of symbol systems and representations, algorithmic notions of flow of control, and conditional logic. All these CT skills should be considered as the core of CT curriculum activity design. In the process of building the outcomes of these basic skills from CT practices, learners have opportunities to develop problem-solving skills through problem formulation, decomposition, generalization, abstraction, and finding a feasible computational solution while engaged in testing and debugging activities.

In order to develop learners' computational thinking, some design and teaching strategies of CT skills can be employed. Concerning curriculum design strategy, the use of the procedures should be introduced in the learning tasks at the lowest level of the curriculum. The learning activities at the higher levels should be complex enough for learners to apply CT skills that they consider relevant and necessary in the tasks for solving problems. In the programming lessons, teachers' explicit guidance of learners' development of CT practices is essential. According to Grover and Pea (2013), it is important to guide learners' cognitive aspects of CT practices in order to enable them to actively reflect on their programming experience. Instead of only programming in the trial-and-error mode, learners should learn to be thinking–doing. For example, at the initial stage of teaching programming, teachers need to clearly explain why the procedures are introduced to learners and how they relate to abstraction and modularization. Teachers should reinforce learners' understanding of the reasons and the relations by occasionally asking them because reflection enables young learners to focus on their own thinking and goals (Davis 2003). Also, teachers need to teach learners how the procedures can be used in various circumstances for the purpose of understanding what abstraction and modularization are. With teachers' instruction, learners can learn to debug

erroneous programs so as to reflect on the value of debugging—an important skill to help tackle computational problems.

Develop computational thinking perspectives through experiences in the programming process

In the process of teaching CT knowledge and skills, the CT curriculum should motivate learners to reflect on their relationship with the digital world. This CT framework adopts expressing, connecting, and questioning from Brennan and Resnick's work (2012), and proposes two more perspectives—computational identity and digital empowerment. First, expressing means learners perceive computation as a medium, and become able to self-express through creating with programming and computation. Second, connecting is the process of learners creating with and for others by programming. Creating with others means that learners may realize their cooperation with others can result in more accomplishments by collaborations, discussions, and remixing of codes. Creating for others implies that learners understand the value of authentic audience when their creations are appreciated by others in various ways like entertaining, engaging, equipping, or educating others. Third, questioning is the ability to "interrogate the taken-for-granted, and, in some cases, responding to that interrogation through design" (Brennan and Resnick 2012), which means learners will become empowered to ask questions about and with technology.

With sufficient experiences of expressing, connecting, and questioning in the computational process, learners may be able to establish computational identity and achieve digital empowerment. Computational identity refers to the individual and collective perspectives of self-perceptions through computational activities with other members in the computational community. When learners express their own ideas in the process of creating with programming and computation (Brennan and Resnick 2012), they will possibly create a sense of computational identity with which the learners feel more related to be part of and become more engaged in the computational group. The development of learners' ability to connect and question may result in digital empowerment which refers to a person's perception of his or her ability to effectively use digital technologies to form life skills and reinforce their ability to take control of their own lives and influence in the technological world (Akkoyunlu et al. 2010; Makinen 2006). In the process of connecting and questioning, learners are able to modify and create new things with computation and programming for the digital world (Brennan and Resnick 2012) and hence become digitally empowered.

To assist learners' development of computational identity and digital empowerment, the design of some learning tasks should enable learners to contribute to others and gain a sense of satisfaction. For instance, learners of higher grades can create a simple addition mathematics game app for the lower grade learners. Their effort in facilitating others' learning by producing educational apps will result in a sense of satisfaction and empowerment. Such reward will not only reinforce their sense of belonging in computational community but also strengthen their self-

perception of competence in using digital technologies to contribute to the community and even to the society.

Adopt a top–down strategy in activity design to create opportunities for solving complex computational problems

In order to achieve the goal of the CT curriculum—to develop learners' CT skills and perspectives, a central strategy is to create opportunities for learners to solve complex computational problems. Complex tasks facilitate the learning of collecting information for the tasks and actively utilizing the learnt knowledge to further explore, create, and construct their own new knowledge (Resnick et al. 2009; Wang et al. 2016). This can be achieved when curriculum planners design activities from the higher to the lower level. Based on the content of the tasks decided at the top, designers can then look back to the activity design at lower levels to see how simpler tasks can be embedded to prepare learners to use this knowledge in solving complex tasks at higher levels (see Fig. 2). This design strategy prepares learners to first build foundational programming knowledge and CT skills, and enables them to reuse and remix code and apply other CT skills when dealing with complex tasks at higher levels. This strategy also provides opportunities for learners to practice their CT skills across levels. In order to have an overall picture of the whole design, a map of the designed activities in the three levels is suggested to be made in the process of constructing the curriculum. The map can reveal the vertical relationship between activities in different levels in terms of CT knowledge and skills.

Use the final project samples as a strategy to review the coverage of CT knowledge across the level

Another strategy to develop CT skills and perspectives is to ensure the CT knowledge in the activities at each level is sufficiently covered. In order to review

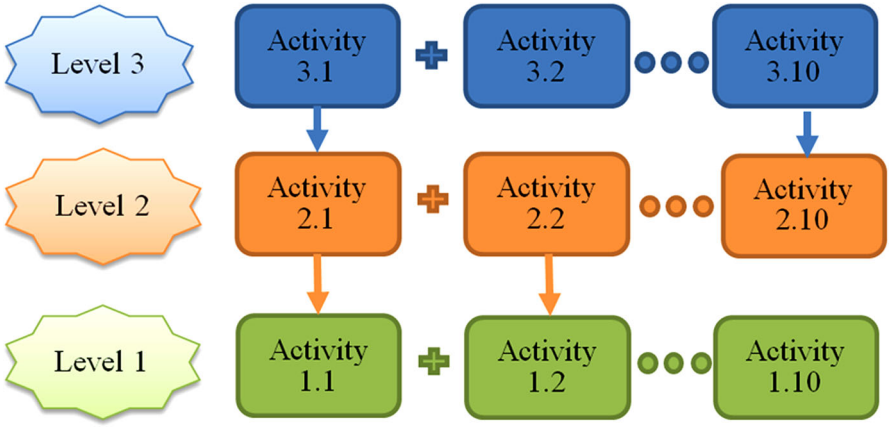

Fig. 2 Adopt a top–down approach of activity design to enable learners to solve complex computational problems

the coverage of CT knowledge at each level, designers may require the teachers who are responsible for teaching each level to produce samples of the final projects. These samples enable curriculum designers to check if any CT knowledge is omitted in the curriculum (see Fig. 3). It should be noted that these sample final projects are not only used for review but also for teachers to understand the scope of the curriculum at a given level. The sample final projects are not expected to be shown to learners as they may hinder learners' creativity to produce their own final projects.

Use interest-driven activity design as a strategy to incubate interest-driven creator

The third strategy to develop learners' CT skills and perspectives is to maintain learners' interest in programming through interest-driven activities. According to the IDC theory, learners can become lifelong creator if they engage in sustained interest-driven learning activities (Chan et al. 2015). Since the ultimate goal of CT development is to nurture interest-driven creator for the digital world, all activities designed in the CT curriculum should be interest-driven. Therefore, interesting and meaningful activities are necessary in the curriculum. There are four possible ways to arouse and maintain learners' interest. First, the learning activities should be kept up-to-date and regarded as modern by learners. Second, the content of the activities have to be relevant to learners' daily lives since learning problem solving requires an authentic problem that matters to and relates to learners, for which they may possibly become more engaged in the learning activities (Jonassen 2011). Third, there is a need to collect the learners' opinions on what is considered as meaningful and interesting, and incorporate them in the CT curriculum design. Lastly, the learning environment should be relaxing for both learners and teachers, which means trying not to rush the curriculum or push too hard on the evaluation of learners' performance. These conditions help build an interest-driven learning environment in which the interest of learners can be activated, maintained, and extended, and their creativity can be nurtured.

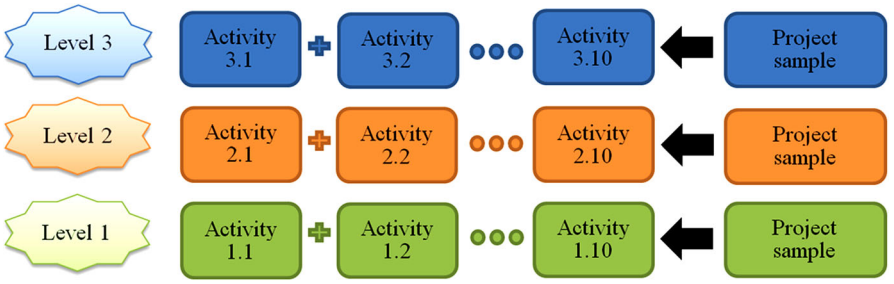

Fig. 3 Use the final project samples to review the coverage of CT knowledge across the level in the CT curriculum

Use the assessment criteria and staging of the final projects to nurture creativity

The fourth strategy is to nurture learners' creativity by designing appropriate assessment criteria and offering opportunities for learners to present their final projects at the end of each level in the curriculum. The assessment criteria should be designed based on the overarching main goal of the curriculum—to nurture creator in the digital world to rethink existing practices across all fields in society with digital technologies. This goal implies that the assessment criteria of the final projects should focus on whether the learners can demonstrate their ability to solve computational problems with digital technologies in a creative way. Creativity is generally believed that a creative person tends to have "the capacity to combine or synthesize existing ideas, images, or expertise in original ways and the experience of thinking, reacting, and working in an imaginative way characterized by a high degree of innovation, divergent thinking, and risk taking" (AAC&U—Association of American Colleges and Universities 2010). Based on this definition, certain personality traits can be drawn to depict what creative people are like, including sensitivity, flexibility, innovative thinking, and the ability to connect, synthesize, and transform ideas and solutions (AAC&U—Association of American Colleges and Universities 2010). Sensitivity refers to the awareness of the possibility of changes, influences, and incompleteness. Flexibility means the ability to "adjust his or her problem solving as task demands are modified" (Krems 1995). Innovative thinking is the ability to extend novel ideas to create new knowledge or knowledge that crosses boundaries (AAC&U—Association of American Colleges and Universities 2010). The above features can serve as the criteria of determining whether learners are creative when producing the final projects. Learners' creativity can be evaluated by how sensitive they can be in identifying computational problems associated with their daily lives and their community, how flexible they can be in finding innovative solutions using digital technologies, and how innovative their solutions can be. These criteria of creativity can guide and motivate learners to become creative problem solvers with digital technologies.

Apart from establishing assessment criteria to induce creativity, presenting the final projects can also motivate learners to be creative. According to the IDC theory, staging is the final step in the creation loop, aiming to provide opportunities for learners to receive feedback and encourage refinement. Making the final projects available for other users to use enables others to provide feedback for designers to make their design more refined if it is not yet well received. They can then continue to refine and extend their designs. Through this process, learners are re-enforced to find innovative ways to develop solutions to make their applications popular. The process of continuous modifications is expected to further nurture creativity. In the long term, it is anticipated that learners will become creator in the digital world with the potential of integrating the existing practices across all fields of society with digital technology.

Conclusion on proposing the framework of curriculum design for CT development and future research work

Figure 4 shows an overview of the framework of curriculum design for CT development in K-12 education. Based on the goals and the principles for the design of CT curriculum, this study proposes seven principles for curriculum designers to consider. The first three principles ensure CT skills and perspectives are delivered in the curriculum through a programming environment that fosters CT knowledge acquisition. The other four principles are strategies formulated to ensure the effective and comprehensive construction of a programming curriculum for CT development, which includes designing complex computational tasks to develop CT skills, reviewing each level of the curriculum to ensure a comprehensive coverage of CT knowledge, designing computational tasks for nurturing learners as interest-driven creator, and establishing appropriate assessment criteria for the final projects and staging learners' final projects for motivating their creativity. The IDC theory plays an important role in guiding the design of the curriculum to stimulate learners' interest and creativity.

The application of the interest-driven creator theory to the curriculum design

The ultimate goal of this curriculum framework is to nurture creative problem solvers in the digital world. As discussed in the sixth principle, the interest-driven activity design in the curriculum is the key to nurture learners' creativity. Therefore, linking up the interest and the creation loops in the IDC theory is a critical aspect of this curriculum design. In the 3-year curriculum, with three levels of curriculum content, learners' interest is the focal point of the activity design. It should be fostered at the initial stage (i.e., level one in the curriculum), and should be maintained and extended throughout the other two levels in the curriculum (Wong et al. 2015).

After learners' interest is triggered, it can be maintained through the fostering of learners' "flow" state, which refers to "an experience of intense emotional involvement and being completely engaged in the activity for its own sake"

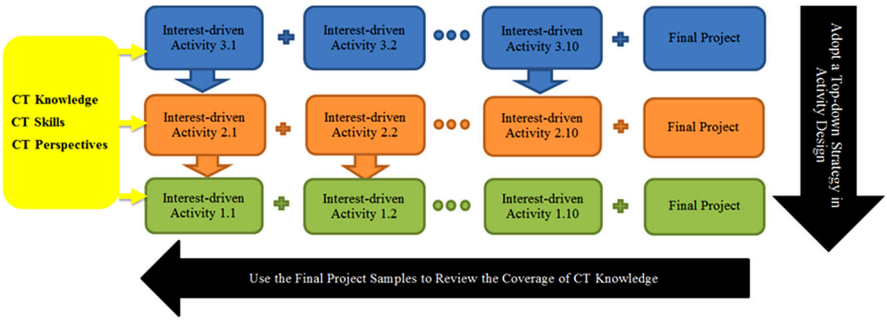

Fig. 4 An overview of the framework for curriculum design for CT development in school education

(Csikszentmihalyi and Rathunde 1993). This state allows learners to fully engage in learning and thus they have greater potential to achieve effective learning (Chan et al. 2015). Such deep emotional engagement in the learning activities can be built in the process of solving complex computational tasks that are of interest to learners or completing a meaningful final project at the end of each level. This experience of "flow" requires learners to apply their CT knowledge and skills to produce programming outcomes such as mobile Apps. In order to empower learners to accomplish meaningful projects, such as mobile applications, at the end of each level, learners must first be exposed to a spectrum of CT knowledge for them to reuse and remix in the learning activities. The curriculum planners are reminded not only to design activities that introduce CT tasks and tools but also to provide opportunities for learners to connect their knowledge. This implies that the activities at the later stages are not just embedded with new CT knowledge but also interwoven with previous knowledge, such that learners can reuse and remix the CT knowledge that they have previously learnt towards producing more complex Apps. This is a versatile means to empower the learners to become interest-driven creator. The ladder of producing a final project at each level provides a platform for learners to extend their interest across levels.

Learners' deep engagement in learning programming through the final project production serves as a bridge linking the interest loop and the creation loop. Creation usually starts with imitation (Chan et al. 2015). Learners assimilate and accommodate CT knowledge by imitating programming examples from their teachers and peers to create their own programs, such as mobile Apps. This is the second stage of the creation loop—combining. Combining is one essential part of the creation process (Chan et al. 2015). In their attempts to create something new, learners will first analyze and evaluate their knowledge, and decide what should be used (Anderson and Krathwohl 2000). They will then combine their knowledge with new ideas to develop novel programming outcomes such as innovative mobile Apps. The integration of the prior CT knowledge and the new ideas help learners solve complex computational problems, for which the top–down strategy is used in the design of the activities in the curriculum. While the opportunity of combining the prior CT knowledge and novel ideas tends to arise more in the higher-level learning activities, it may appear at lower levels, but less frequently due to the limited CT knowledge acquired by learners. The final stage in creation loop is staging. Staging means that learners are able to show their CT knowledge and skills by sharing their programming outcomes with others. Staging is a significant process not only because it is a showcase of the learners' computational and creative competence, but it also serves as a kind of assessment and learning opportunity when they receive feedback from the users. Learners can then continuously improve their products based on users' constructive feedback. This will drive learners to be innovative if they want to produce outcomes that are popular among users. This learning process may lead learners to become lifelong creator in the digital world. A programming environment like App Inventor allows learners to produce mobile Apps and use them on mobile devices, which serves as a perfect stage to show their learning outcomes in programming education.

Nurturing interest-driven creator is a lengthy process and requires developing the habit of being an interest-driven creator. In the design of a CT skill-focused programming curriculum, one lesson per week serves well as a cue to trigger learners' behavior of learning programming regularly. The three levels in the 3-year curriculum enable learners to learn programming as a routine, while progressively developing their habit of programming. Such routines can only be sustained when learners feel satisfied or rewarded in the learning process. Therefore, the activities need to be interesting and meaningful to learners and enable them to have a sense of satisfaction. It will act as "a reinforcer of cue-response association" (Chen et al. 2015) which in turn encourages them to keep on learning and creating with digital technologies in the future.

A review of the four strategies in the curriculum design

Following the discussion of the application of the IDC theory to the curriculum design, the role of the last four principles can be reviewed. The roles of the strategies in the last four principles are to facilitate the progressive structuring of the curriculum with the focus on motivating learners to be engaged in the learning activities, and to put emphasis on CT development as well as the activity design to foster learners' creativity in solving computational problems. Therefore, the most fundamental strategy is the interest-driven activity design. Interest is the key factor of effective learning (Schiefele 1991), and has a positive relationship with learners' motivation (Frymier et al. 1996). It is believed that CT development can be optimized by the inducement of learners' interest. The significance of interest in nurturing creativity is advocated by the IDC theory which proposes a continuing engagement in the interest-driven learning activities which is required for nurturing creator (Looi et al. 2015). Since nurturing creative problem solvers is the ultimate target of CT development, the design of the interest-driven learning activities should be placed at the top of the agenda in the curriculum design. Without this strategy, learners may hardly be motivated to even start to learn programing, especially when programming is generally perceived as a technical and specialized activity which only suits a small population (Resnick et al. 2009). Another important strategy is the assessment criteria of the final projects. Since creativity is the essential learning outcomes of the curriculum and is expected to be shown by learners through the final projects, the establishment of the assessment criteria for creativity becomes critical in the curriculum design. The criteria of creativity design of the final projects would help delineate what creativity components are considered as desirable in the project design which would provide a strong guidance for learners to design their projects creatively, and can as well serve as an extrinsic motivation to encourage them to think outside the box in the project design.

With a thorough consideration of learning motivation in the learning activities design as well as the goal of fostering creativity of this programming curriculum, designers can put effort to facilitate CT development in the curriculum. Perceived as the core of the CT development, CT skills play an important role in equipping learners with problem-solving ability. Learners can acquire and practice these skills through solving complex computational problems which should be best designed by

deploying a top–down strategy in the curriculum—embedding prior knowledge in previous level of curriculum activities. Without such design, learners may be unable to progressively acquire the skills, and incrementally apply what they have learnt in tackling the complex computational tasks. Lastly, the strategy of producing project samples serves well in reviewing the curriculum coverage which ensures learners can produce a meaningful final project at the end of each level with what they have learnt in that level.

Future research work

There are a number of future research tasks being considered in the agenda of this study. First, design and implement a K-12 programming curriculum constructed based on these seven principles. Second, design instruments to assess CT knowledge, skills, and perspectives of learners in the programming curriculum. Third, evaluate whether the first three principles are effective by evaluating the progression of learning outcomes of CT knowledge, skills, and perspectives which includes computational identity and digital empowerment. Fourth, evaluate whether the other four principles are effective in CT development; for example, carrying out surveys to study whether the learners are interested in the curriculum units, and conducting a series of interviews to continually collect teachers' feedback on whether the learners can incrementally acquire CT knowledge and skills across this three-level programming curriculum. Finally, investigate whether creative problem solvers can be nurtured in this three-level programming curriculum by reviewing projects produced by learners, conducting focus group meetings with learners and interviewing teachers across this three-level programming curriculum about the learners' creative and problem-solving ability.

References

AAC&U—Association of American Colleges and Universities. (2010). *Creative Thinking VALUE Rubric*. Retrieved Nov 11, 2016 from https://www.aacu.org/value/rubrics/creative-thinking.

Aho, A. V. (2012). Computation and computational thinking. *The Computer Journal, 55*(7), 832–835.

Akkoyunlu, B., Yilmaz Soylu, M., & Çağlar, M. (2010). A study on developing "digital empowerment scale" for university learners. *Hacettepe University Journal of Education, 39*(39), 10–19.

Anderson, L. W., & Krathwohl, D. R. (2000). *A taxonomy for learning, teaching, and assessing—A revision of Bloom's taxonomy of educational objectives*. New York: Longman.

Barr, D., Harrison, J., & Conery, L. (2011). Computational thinking: A digital age skill for everyone. *ISTE Learning and Leading, 38*(6), 20–22.

Barr, V., & Stephenson, C. (2011). Bringing computational thinking to K-12: What is involved and what is the role of the computer science education community? *ACM Inroads, 2*(1), 48–54.

Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. In A. F. Ball & C. A. Tyson (Eds.), *Proceedings of the 2012 annual meeting of the American educational research association* (pp. 1–25). Vancouver: American Educational Research Association.

Chan, T. W., Looi, C. K., & Chang, B. (2015). The IDC theory: Creation and the creation loop. In T. Kojiri, T. Supnithi, Y. Wang, Y.-T. Wu, H. Ogata, W. Chen, S. C. Kong, & F. Qiu (Eds.), *Workshop*

proceedings of the 23rd international conference on computers in education (pp. 814–820). Hangzhou: Asia-Pacific Society for Computers in Education.

Chen, W., Chan, T. W., Liao, C. C. Y., Cheng, H. N. H., So, H., & Gu, X. (2015). The IDC theory: Habit and the habit loop. In T. Kojiri, T. Supnithi, Y. Wang, Y.-T. Wu, H. Ogata, W. Chen, S. C. Kong, & F. Qiu (Eds.), *Workshop proceedings of the 23rd international conference on computers in education* (pp. 821–828). Hangzhou: Asia-Pacific Society for Computers in Education.

Clark, R. C., & Mayer, R. E. (2007). *E-learning and the science of instruction: Proven guidelines for consumers and designers of multimedia learning*. San Francisco: Pfeiffer.

Csikszentmihalyi, M., & Rathunde, K. (1993). The measurement of flow in everyday life: Toward a theory of emergent motivation. *Nebraska Symposium on Motivation, 40*, 57–97.

Curzon, P., Dorling, M., Ng, T., Selby, C., & Woollard, J. (2014). *Developing computational thinking in the classroom: A framework.* Retrieved Nov 11, 2016 from https://academy.bcs.org/sites/academy.bcs.org/files/DevelopingComputationalThinkingInTheClassroomaFramework.pdf.

Davis, E. A. (2003). Prompting middle school science students for productive reflection: Generic and directed prompts. *Journal of the Learning Sciences, 12*(1), 91–142.

Frymier, A., Shulman, G., & Houser, M. (1996). The development of a learner empowerment measure. *Communication Education, 45*(3), 181–199.

Futschek, G. (2006). Algorithmic thinking: The key for understanding computer science. *Lecture Notes in Computer Science, 4226*, 159–168.

Grover, S., & Pea, R. (2013). Computational thinking in K-12: A review of the state of the field. *Educational Researcher, 42*(1), 38–43.

Jonassen, D. (2011). *Learning to solve problems: A handbook for designing problem-solving learning environments*. New York: Routledge.

Krems, J. F. (1995). Cognitive flexibility and complex problem solving. In P. A. Frensch & J. Funke (Eds.), *Complex problem solving: The European perspective* (pp. 201–218). Hillsdale: Lawrence Erlbaum Associates.

Looi, C. K., Chan, T.-W., Wu, L., & Chang, B. (2015). In T. Kojiri, T. Supnithi, Y. Wang, Y.-T. Wu, H. Ogata, W. Chen, S. C. Kong, & F. Qiu (Eds.), *Workshop proceedings of the 23rd international conference on computers in education*. Hangzhou: Asia-Pacific Society for Computers in Education.

Lye, S. Y., & Koh, J. H. L. (2014). Review on teaching and learning of computational thinking through programming: What is next for K-12? *Computers in Human Behavior, 41*, 51–61.

Makinen, M. (2006). Digital empowerment as a process for enhancing citizens' participation. *E-Learning, 3*(3), 381–395.

Parnas, D. L. (1972). On the criteria to be used in decomposing systems into modules. *Communications of the ACM, 15*(12), 1053–1058.

Resnick, M., Maloney, J., Monroy-Hernandez, A., Eastmond, E., Brennan, K., Millner, A., et al. (2009). Scratch: Programming for all. *Communications of the ACM, 52*(11), 60–67.

Schiefele, U. (1991). Interest, learning, and motivation. *Educational Psychologist, 26*(3&4), 299–323.

Seiter, L., & Foreman, B. (2013). Modeling the learning progressions of computational thinking of primary grade students. In B. Simon, A. Clear, & Q. Cutts (Eds.), *Proceedings of the ninth annual international ACM conference on international computing education research* (pp. 59–66). La Jolla: Association for Computing Machinery.

Selby, C., & Woollard, J. (2013). Computational thinking: The developing definition. In J. Carter, I. Utting, & A. Clear (Eds.), *Proceedings of 18th annual conference on innovation and technology in computer science education* (p. 6). Canterbury: University of Southampton.

Spady, W. (1994). *Outcome-based education: Critical issues and answers*. Arlington: American Association of School Administrators.

Spady, W., & Marshall, K. (1991). Beyond traditional outcome-based education. *Educational Leadership, 49*(2), 67–72.

Wang, H.-Y., Huang, I., & Hwang, G.-J. (2016). Comparison of the effects of project-based computer programming activities between mathematics-gifted students and average students. *Journal of Computers in Education, 3*(1), 33–45.

Wing, J. M. (2006). Computational thinking. *Communications of the ACM, 49*(3), 33–35.

Wing, J. M. (2011). *Computational thinking: What and why?*. Retrieved Nov 11, 2016 from https://www.cs.cmu.edu/~CompThink/resources/TheLinkWing.pdf.

Wong, L. H., Chan, T. W., Chen, Z. H., King, R. B., & Wong, S. L. (2015). The IDC theory: Interest and the interest loop. In T. Kojiri, T. Supnithi, Y. Wang, Y.-T. Wu, H. Ogata, W. Chen, S. C. Kong, & F.

Qiu (Eds.), *Workshop proceedings of the 23rd international conference on computers in education* (pp. 804–813). Hangzhou: Asia-Pacific Society for Computers in Education.

Siu-Cheung Kong is Professor of the Department of Mathematics and Information Technology and Director of Centre for Learning, Teaching and Technology at The Education University of Hong Kong. He was the Convener of Theory and Practice of Pedagogical Design for Learning in Digital Classrooms International Research Network (IRN) under World Educational Research Association (WERA) from December 2012 to December 2015. His research interests include pedagogy in the digital classroom, policy on technology-transformed education, and the professional development of teacher for learner-centered learning in seamless learning environments, IT in mathematics education, coding for computational thinking development, and computational thinking education.