Computers in Human Behavior 41 (2014) 51–61

Contents lists available at ScienceDirect

Computers in Human Behavior

journal homepage: www.elsevier.com/locate/comphumbeh

Review

Review on teaching and learning of computational thinking through programming: What is next for K-12?

Sze Yee Lye *, Joyce Hwee Ling Koh

National Institute of Education, Nanyang Technological University, Singapore, 1 Nanyang Walk, Singapore 637616, Singapore

ARTICLE INFO

Article history:
Available online 30 September 2014

Keywords:
Programming
Scratch
Computer science education
K-12
Computational thinking

ABSTRACT

Programming is more than just coding, for, it exposes students to computational thinking which involves problem-solving using computer science concepts like abstraction and decomposition. Even for non-computing majors, computational thinking is applicable and useful in their daily lives. The three dimensions of computational thinking are computational concepts, computational practices and computational perspectives. In recent years, the availability of free and user-friendly programming languages has fuelled the interest of researchers and educators to explore how computational thinking can be introduced in K-12 contexts. Through an analysis of 27 available intervention studies, this paper presents the current trends of empirical research in the development of computational thinking through programming and suggests possible research and instructional implications. From the review, we propose that more K-12 intervention studies centering on computational practices and computational perspectives could be conducted in the regular classroom. To better examine these two dimensions, students could be asked to verbalize their thought process using think aloud protocol while programming and their on-screen programming activity could be captured and analyzed. Predetermined categories based on both past and recent programming studies could be used to guide the analysis of the qualitative data. As for the instructional implication, it is proposed that a constructionism-based problem-solving learning environment, with information processing, scaffolding and reflection activities, could be designed to foster computational practices and computational perspectives.

Contents

* Corresponding author.
E-mail addresses: lye.szeyee@gmail.com (S.Y. Lye), joyce.koh@nie.edu.sg (J.H.L. Koh).

http://dx.doi.org/10.1016/j.chb.2014.09.012
0747-5632/

1. Introduction

Programming for K-12 can be traced to the 1960s when Logo programming was first introduced as a potential framework for teaching mathematics (Feurzeig & Papert, 2011). In Logo, the students moves the turtle (arrow) on the screen by issuing commands like FD 100 (forward 100). In his seminal book "Mindstorms: Children, computers and powerful ideas", Papert (1980) advocated the use of the discovery constructionist mode for learning Logo. Nevertheless, Logo did not catch on in mainstream schools in the 1980s, possibly because of the incompatibility between its discovery-enabled approach and the more conventional behaviourist school culture back then (Agalianos, Noss, & Whitty, 2001). Papert (1980) claimed that the Logo programming experience could develop powerful intellectual thinking skills among children. Contrary to his claim, empirical studies of Logo programming did not find conclusive evidence of it improving the thinking skills of children (Kurland, Pea, Clement, & Mawby, 1986; Pea, 1983).

After Logo, the use of programming to teach thinking skills in K-12 was not extensively reported. However, in the recent years, there has been renewed interest in introducing programming to K-12 students (Grover & Pea, 2013; Kafai & Burke, 2013). This is fuelled by the availability of easy-to-use visual programming languages such as Scratch (Burke, 2012; Lee, 2010), Toontalk (Kahn, Sendova, Sacristán, & Noss, 2011), Stagecast Creator (Denner, Werner, & Ortiz, 2012) and Alice (Graczyńska, 2010). Many of these new programming languages such as Scratch and Alice have been modelled after aspects of Logo (Utting, Cooper, Kölling, Maloney, & Resnick, 2010).

During programming, students are exposed to computational thinking, a term popularized by Wing (2006). It involves the use of computer science concepts such as abstraction, debugging, remixing and iteration to solve problems (Brennan & Resnick, 2012; Ioannidou, Bennett, Repenning, Koh, & Basawapatna, 2011; Wing, 2008). This form of thinking can be considered to be fundamental for K-12 students because it requires "thinking at multiple abstractions" (Wing, 2006, p. 35). More importantly, computational thinking is in line with many aspects of 21st century competencies such as creativity, critical thinking, and problem- solving (Ananiadou & Claro, 2009; Binkley et al., 2012). Thus, it is not surprising that many educators assert that programming is important for K-12 students in this era (Kafai & Burke, 2013; Margolis, Goode, & Bernier, 2011; Resnick et al., 2009). This revived interest in programming for K-12 settings suggests the need to consider how it can be better related to the kinds of educational outcomes that it can potentially foster. Some of the outcomes suggested by researchers are the ability to think more systematically (Kafai & Burke, 2013) and the development of mathematical and scientific expertise (Sengupta, Kinnebrew, Basu, Biswas, & Clark, 2013). Yet, in the current literature, there is a dearth of papers that explore computational thinking through programming in K-12 contexts (Grover & Pea, 2013) as these programming studies are more often examined for tertiary students undertaking computer science courses (e.g., Katai & Toth, 2010; Moreno, 2012). Therefore, in this paper, we attempt to examine published empirical studies involving students in both K-12 and higher education contexts so as to derive insights on computational thinking through programming for K-12 curriculum.

2. Computational thinking

2.1. Definition

The term computational thinking is made popular by Wing (2006). In her seminal article on computational thinking, she argued that computational thinking "represents a universally applicable attitude and skill set everyone, not just computer scientists, would be eager to learn and use" (p. 33). Since then, computational thinking has gained traction in the K-12 context in the USA. However, the definition of computational thinking still remains contested as no dominant discourse reigns (Barr & Stephenson, 2011; Brennan & Resnick, 2012; Grover & Pea, 2013). For example, the International Society for Technology in Education (ISTE) views computational thinking as algorithmic thinking with automation tools and data representation with the use of simulation. On the other hand, the National Research Council (NRC) recommends mathematics and computational thinking to be one of the eight essential practices for the scientific and engineering dimension outlined in the "Framework for K-12 Science Education" (NRC, 2012). In this framework, mathematics and computational thinking involves the use of computer tools to represent physical variables and the relationships among them.

For both ISTE and NRC, students may be considered to be exhibiting computational thinking even though they are not creating with technology tools. Conversely, programming involves students exhibiting computational thinking through the construction of artifacts (Kafai & Burke, 2013; Resnick et al., 2009). Thus, the general definitions on computational thinking suggested by ISTE and NRC may not be suited for programming. Hence, in this review on computational thinking through programming for K-12 students, we are using the framework proposed for Scratch by Brennan and Resnick (2012). Scratch is a popular programming language used in K-12 settings (e.g., Baytak & Land, 2011; Kafai, Fields, & Burke, 2010; Tangney, Oldham, Conneely, Barrett, & Lawlor, 2010; Theodorou & Kordaki, 2010). With respect to Scratch, Brennan and Resnick (2012) proposed three dimensions of computational thinking: computational concepts, computational practices, and computational perspectives. Table 1 summarizes the key ideas on these three dimensions. These dimensions are appropriate for understanding how K-12 students approach programming as they are also in line with the Logo programming language knowledge proposed by Mayer (1992). This includes the syntactic, semantic, schematic knowledge (computational