Computers & Education 95 (2016) 202–215

Contents lists available at ScienceDirect

Computers & Education

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

Exploring students' computational practice, design and performance of problem-solving through a visual programming environment

Po-Yao Chao [a, b, *]

[a] Department of Information Communication, Yuan Ze University, Taiwan
[b] Innovation Center for Big Data and Digital Convergence, Yuan Ze University, Taiwan

ARTICLE INFO

Article history:
Received 7 September 2015
Received in revised form 24 January 2016
Accepted 25 January 2016
Available online 28 January 2016

Keywords:
Computer programming
Visual problem solving
Students programming patterns

ABSTRACT

This study aims to advocate that a visual programming environment offering graphical items and states of a computational problem could be helpful in supporting programming learning with computational problem-solving. A visual problem-solving environment for programming learning was developed, and 158 college students were conducted in a computational problem-solving activity. The students' activities of designing, composing, and testing solutions were recorded by log data for later analysis. To initially unveil the students' practice and strategies exhibited in the visual problem-solving environment, this study proposed several indicators to quantitatively represent students' computational practice (*Sequence*, *Selection*, *Simple iteration*, *Nested iteration*, and *Testing*), computational design (*Problem decomposition*, *Abutment composition*, and *Nesting composition*), and computational performance (*Goal attainment* and *Program size*). By the method of cluster analysis, some empirical patterns regarding the students' programming learning with computational problem-solving were identified. Furthermore, comparisons of computational design and computational performance among the different patterns of computational practice were conducted. Considering the relations of students' computational practice to computational design and performance, evidence-based suggestions on the design of supportive programming environments for novice programmers are discussed.

1. Introduction

Programming has been recognized as one of the important competencies that require students to use computational tools to address real-world problems in the 21st century (Einhorn, 2011; Grover & Pea, 2013; Yen, Wu, & Lin, 2012). Learning programming is not only a prerequisite for becoming a computer scientist, but it is also necessary for the practice of solving problems and designing systems (Palumbo, 1990; Robins, Rountree, & Rountree, 2003). Programming requires programmers to plan solutions to problems, precisely transform the plans into syntactically correct instructions for execution, and assess the consequent results of executing those instructions (Brookshear, 2003; Deek, 1999; Ismal, Ngah, & Umar, 2010). However, research revealed that at the conclusion of introductory programming courses, most students have difficulties in decomposing problems, developing plans and implementing their plans with programming languages to solve programming

* Department of Information Communication, Yuan Ze University, 135 Yuan-Tung Road, Chung-Li 32003, Taiwan.
E-mail address: poyaochao@saturn.yzu.edu.tw.

http://dx.doi.org/10.1016/j.compedu.2016.01.010
0360-1315/

problems (Lister et al., 2004; McCracken et al., 2001; de Raadt, 2007; Robins et al., 2003). Some of them lack adequate understanding of fundamental programming constructs, and most of them lack strategies for transforming programming problems into workable plans and algorithms (Deek, 1999; Kessler & Anderson, 1986; Li & Watson, 2011; de Raadt, 2007). This may be because the formal instruction in programming mostly focuses on students' mastery of a general-purpose programming language and adopts a programming tool that is intentionally designed for professional programmers (Deek, 1999; Ismal et al., 2010; Linn & Clancy, 1992; Robins et al., 2003; Xinogalos, 2012). The employment of the general-purpose programming language and the professional programming tool often drives the teachers and students to invest their efforts more on mastering programming language features than on developing design strategies for solving programming problems (Brusilovsky, Calabrese, Hvorecky, Kouchnirenko, & Miller, 1997; Deek, 1999; Linn, 1985; Pears et al., 2007).

Numerous studies have been devoted to research on instructional and environmental assistance for programming learning (Kelleher & Pausch, 2005; Winslow, 1996). Among the studies aiming to devise potential means for enhancing programming, an alternative approach to engaging students in solving computational problems (Edmonds, 2008) has been recognized as an effective way of cultivating students' programming constructs and skills (Liu, Cheng, & Huang, 2011; Ring, Giordan, & Ransbottom, 2008). This method often provides students with computational problems, which are specially designed to foster specific programming concepts or skills. In a scenario requiring students to solve a computational problem by exercising various programming knowledge and strategies, the students are expected to learn by formulating computer programs and systematically evaluating the consequent results (Deek, 1999). Many studies have also proposed alternative approaches to the students' difficulties in programming by the use of visual programming environments, such as Scratch and Alice (Cooper, Dann, & Pausch, 2000; Maloney, Resnick, Rusk, Silverman, & Eastmond, 2010), LighBot and PlayLOGO 3D (Gouws, Bradshaw, & Wentworth, 2013; Paliokas, Arapidis, & Mpimpitsos, 2011), or objectKarel and Jeroo (Sanders & Dorn, 2003; Xinogalos, 2012). These environments often adopt different visual programming elements that help novice programmers construct their programs or understand the process of program execution and the state of a problem (Green & Petre, 1996; Kelleher & Pausch, 2005; Navarro-Prieto & Canas, 2001). Research has revealed that visual programming environments could enhance novice programmers' engagement in programming tasks and help them demonstrate programming skills and problem solving strategies during the course of creating digital artifacts or solving programming problems (Cooper et al., 2000; Lye & Koh, 2014). Although visual programming environments are becoming important and have demonstrated their particular benefits to assist learning programming and problem solving (Lye & Koh, 2014), little is known about how novice programmers use a visual programming environment to learn to solve computational problems. Moreover, because constructing a computer program to solve a computational problem in a visual programming environment requires novice programmers to manipulate visual programming elements (e.g., control-flow blocks) to formulate and test a design solution to the problem (e.g., Gouws et al., 2013; Maloney et al., 2010), the programmers' behavior and strategies of solving computational problems in a visual programming environment may affect their performance of problem solving. Therefore, there is a need to further explore the novice programmers' behavioral patterns in a visual programming environment and investigate the difference in their strategies and performance of solving computational problems among different behavioral patterns.

Based on the aforementioned rationale, the purpose of this study is twofold. The first is to develop a visual problem-solving environment for programming learning and explore how novice programmers use it to learn to solve computational problems through interacting with the provided visual programming elements. To understand how novice programmers interact with the proposed environment to solve computational problems, the second purpose is to investigate novice programmers' visual programming behavior and strategies of computational problem-solving enacted in the visual problem-solving environment, as well as to examine the performance of computational problem solving among different patterns of visual programming behavior in computational problem solving activities. To this end, this study aims to answer the following research questions:

- What are the novice programmers' patterns of visual programming behavior exhibited in a visual programming environment to solve computational problems?
- Do the novice programmers' computational design and performance of solving computational problems differ in different patterns of visual programming behavior?

The results of this study may be of interest to interface designers attempting to design a specific programming learning environment for novice programmers. The results may also particularly interest teachers or educators who design formal instruction for students to foster their programming strategies (de Raadt, Watson, & Toleman, 2009) or computational problem solving skills.

2. Related works

2.1. Programming learning in visual programming environments

Many visual programming environments have been developed to provide novice programmers with visual supports in constructing programs and understanding programming constructs. For example, ToonTalk enables a programming environment in which users interact with visual objects, such as birds or cars in a city, to construct programs (Kahn, 1996). These