Teaching in Visual Programming Environments

Wilfred W. F. Lau
The University of Hong Kong, China

Allan H. K. Yuen
The University of Hong Kong, China

INTRODUCTION

This article is an extension of the authors' previous work "*Toward a Framework of Programming Pedagogy*" (Lau & Yuen, 2009). It aims to highlight the recent development of programming pedagogy, in particular those related to visual programming environments.

The persistent problems of high attritions among students and under-representation of females in the computing academia have been widely documented. For example, Sloan and Troy (2008) found that the annual attrition rate among freshmen and sophomores who majored in computer science (CS) in the US has been reported to average 19% and at some schools to be 66%. According to Lasserre and Szostak (2011), the dropout rate in CS courses was between 30% to 50%. While females earned 57% of all the undergraduate degrees in the US in 2009, only 18% of all the computer and information sciences graduates were females (National Center for Women & Information Technology, 2010).

To address the problems more fundamentally, it is argued that more effort should be made to transform computing education in high school. Cuny (2012) described an ambitious project called CS 10K that "aims to develop effective new high school computing curricula and get it into 10,000 high schools taught by 10,000 well-prepared teachers by 2016" (p. 35) in the US. Internationally, many other countries also seize the opportunity to revamp their high school CS curricula. The *Shut Down or Restart? The Way Forward for Computing in the UK Schools* (*SDoR?*) report supported the replacement of information and communication technology (ICT) with three inter-related concepts, namely digital literacy, information technology, and CS in the national curriculum (Snyder, 2012) among other recommendations.

All these initiatives were aimed to attract and retain students in CS programmes. Programming pedagogy has played a significant role in the teaching and learning of computer programming, which often includes the use of a variety of programming tools (Pears et al., 2007). These tools aid novice programmers to develop programs through program visualisation and algorithm animation (Yuen, 1999, 2006). In our previous review on programming pedagogy (Lau & Yuen, 2009), we identified seven approaches of teaching computer programming based on an initial framework that conceptualised pedagogy along the two dimensions, namely programming knowledge and programming representation. We also predicted that with a trend towards user friendliness, there will be a gradual shift from concept-textual pedagogy to concept-visual pedagogy, which is realised through visual programming. As such, this article examines the recent use of visual programming environments in classrooms and reports its effectiveness. It first gives a synopsis of the seven programming pedagogies. Next it focuses on the discussion of three popular integrated development environments (IDEs) that support visual programming: Alice, Greenfoot, and Scratch. Finally, it envisages the future of visual programming and concludes with suggestions for further developments.

BACKGROUND

Drawing on the pertinent literature of computer programming from our previous review (Lau & Yuen, 2009), we identified seven programming pedagogies. The following paragraphs provide a brief description of each approach.

DOI: 10.4018/978-1-4666-5888-2.ch253

Copyright © 2015, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

The structured programming approach was intended to "support the production of correct, understandable programs which are easy to modify and maintain" (Freiburghouse & Liskov, 1973, p. 5). Control structures of sequence, selection, and repetition were allowed while the use of the GOTO statement was not encouraged (Dijkstra, 1968). A top-down design method was adopted to facilitate the development of a structured program. Programs were improved successively through stepwise refinement. The problem solving approach consisted of four steps, namely understanding, designing, writing, and reviewing in developing a program (Barnes, Fincher, & Thompson, 1997). Thompson (1997) claimed that using the approach, "a novice can make substantial progress in completing a programming task before beginning to write any program code" (p. 324). Gries (1974) argued that it was pedagogically unsound to assume that students should have learnt programming after providing them with tools and examples. He suggested the four-phase process of problem solving by Polya (1957) in order to address this problem.

The software development approach integrated both the problem solving skills and the programming skills needed into a single process, and thus gave a framework for beginning students (Deek, 1999). Deek (1999) noted three types of difficulties faced by students when learning to program: (1) deficiencies in problem solving strategies and tactical knowledge; (2) ineffective pedagogy of programming instruction; and (3) misconceptions about syntax, semantics, and pragmatics. It was shown that this approach helped to address all the three types of difficulties. The small programming approach reduced cognitive load on novices by programming in a "small" scale. Programming by Number enabled students to start programming in a step-by-step manner with the flexibility in the design of the solution to a problem (Glaser, Hartel, & Garratt, 2000). Brusilovsky, Kouchnirenko, Miller, and Tomek (1994) proposed three approaches of teaching introductory programming, namely the incremental approach, the mini-language approach, and the sub-language approach. These three approaches offered simple and small language subsets and a visually attractive metaphor embedded in a context-rich environment to facilitate novices learning to program. The language teaching approach argued that research in the learner strategies in second language pedagogy may reveal insights into programming pedagogy (Baldwin & Macredie, 1999). Deek and Friedman (2001) noted that the common element that exists in both domains (problem solving and program development) suggested "new ways for students to transfer skills between domains" (p. 9).

The learning theory approach emphasised the importance of learning theory in programming education. In the assessment of programming performance, Lister and Leaney (2003) recommended a criterion-referencing approach to grading, in which explicit and clear criteria were set for each grade with reference to the Bloom's Taxonomy of Educational Objectives (Bloom, 1956). Macfarlane and Mynatt (1988) investigated the effectiveness of advance organizer in teaching the syntax of arrays. While the experimental and control groups did not differ on syntactic knowledge (near transfer), the experimental group performed best in semantic knowledge (far transfer).

TEACHING WITH VISUAL PROGRAMMING

Jimenez-Peris, Pareja-Flores, Patino-Martinez, and Velazquez-Iturbide (2000) made the prediction of a day in a 2020 university that "current lab programming environments are truly educational, and they are highly visual and intuitive. They detect and accurately diagnose most errors…" (p. 132). As we enter the second decade of the millennium, it is increasingly apparent that visual programming with the support of IDEs has become a trend, and has brought new impetus and learning experiences to students, specifically for school children who are easily allured by the media-rich culture.

Davies, Polack-Wahl, and Anewalt (2011) surveyed the environments (command-line environment, an IDE, or both) that students used for programming in the US, and found that while 15% of the students used command-line environment for both CS1 and CS2 courses, 47% and 40% used graphical IDE for CS1 and CS2 courses respectively. Mason, Cooper, and de Raadt (2012) reported that with regard to the environments/tools being used for introductory programming courses, there has been "the movement away from using text editors and command line compilers only -from approximately 45% of these instructors using no IDE/

7 more pages are available in the full version of this document, which may be purchased using the "Add to Cart" button on the publisher's webpage:
www.igi-global.com/chapter/teaching-in-visual-programming-environments/112676

Related Content

A Rough Set Theory Approach for Rule Generation and Validation Using RSES
Hemant Rana and Manohar Lal (2016). *International Journal of Rough Sets and Data Analysis (pp. 55-70).*
www.irma-international.org/article/a-rough-set-theory-approach-for-rule-generation-and-validation-using-rses/144706

Complexity Analysis of Vedic Mathematics Algorithms for Multicore Environment
Urmila Shrawankar and Krutika Jayant Sapkal (2017). *International Journal of Rough Sets and Data Analysis (pp. 31-47).*
www.irma-international.org/article/complexity-analysis-of-vedic-mathematics-algorithms-for-multicore-environment/186857

Reversible Data Hiding Scheme for ECG Signal
Naghma Tabassum and Muhammed Izharuddin (2018). *International Journal of Rough Sets and Data Analysis (pp. 42-54).*
www.irma-international.org/article/reversible-data-hiding-scheme-for-ecg-signal/206876

Design of a Migrating Crawler Based on a Novel URL Scheduling Mechanism using AHP
Deepika Punj and Ashutosh Dixit (2017). *International Journal of Rough Sets and Data Analysis (pp. 95-110).*
www.irma-international.org/article/design-of-a-migrating-crawler-based-on-a-novel-url-scheduling-mechanism-using-ahp/169176

Validating IS Positivist Instrumentation: 1997-2001
Marie-Claude Boudreau, Thilini Ariyachandra, David Gefen and Detmar W. Straub (2004). *The Handbook of Information Systems Research (pp. 15-26).*
www.irma-international.org/chapter/validating-positivist-instrumentation/30340