Chapter 49
Designing for Computational Expression:
Four Principles for the Design of Learning Environments towards Computational Literacy

David Weintrop
Northwestern University, USA

Uri Wilensky
Northwestern University, USA

ABSTRACT

In this chapter, framed by Vygotsky's sociocultural theory, Wilensky and Papert's restructuration theory, and Noss and Hoyles' theoretical construct of webbing, the authors explore four practical design principles facilitating the creation of learning environments that can overcome the challenge of introducing learners to computational expression in meaningful contexts and can start learners down the path towards computational literacy. The four design principles discussed are (1) low-threshold interfaces, (2) task-specific tools, (3) visual feedback, and (4) in-context examples. The heart of this chapter presents these features and their design rationales in the context of a qualitative study examining participants' use of RoboBuilder, a blocks-based, program-to-play game.

INTRODUCTION

The ability to express ideas in a computationally meaningful way is becoming an increasingly important skill (diSessa, 2000; National Research Council, 2010, 2011; Papert, 1980, 1993; Wilensky, 2001; Wing, 2006). As computational devices become cheaper, more powerful, and increasingly ubiquitous, the ability to take advantage of this computational power in a personally meaningful way has become more valuable. The idea that this skill has far reaching benefits is not new. Papert (1980), in his seminal book *Mindstorms*, argued that computers have the ability to fundamentally change how people think and learn. Twenty years later, diSessa (2000) reasserts that the ability to

DOI: 10.4018/978-1-4666-6042-7.ch049

express ideas in a computationally meaningful way can serve as the foundation of a powerful new literacy that will have widespread positive effects on society. Central to this new literacy is the ability to read, share, and express ideas in a medium that a computational device can interpret and execute. Traditionally, the knowledge and skills associated with this activity have been a part of the domain of computer science, but they are beneficial in a wide range of disciplines (Wing, 2006). Indeed, computer science educators have long championed the importance of computer science principles in benefiting the larger population, arguing that these skills deserve a place alongside reading, writing, and arithmetic as a core 21st century skill (Guzdial & Soloway, 2003).

Part of the challenge of introducing learners to these skills is designing learning environments that support the act of computational expression in a way that enables them to have early successes in a meaningful context. In this chapter, we discuss four design principles, framed by theory and supported by data from a qualitative study of *RoboBuilder* (Weintrop & Wilensky, 2012), a program-to-play game of our own design. This chapter begins by providing a background of prior research that informed these design principles. Next, we introduce RoboBuilder, outline the methods of the qualitative study, and articulate our theoretical framework. We then discuss the four principles. To conclude, we discuss implications of this work, limitations of the approach advocated in this chapter, and contributions to the goal of creating a computationally literate society.

BACKGROUND

The skills associated with computational literacy have many applications, but relatively few students are given the opportunity to learn them. These skills include the ability to read as well as compose sets of instructions using a representational medium that a computer can interpret and execute, along with knowing how and when to use standard computing constructs like conditional logic, iterative structures and recursion to achieve a computational goal. While these skills are taught in computer science courses, restricting them to such contexts greatly limits the audience for this important skill set as few students take computer science courses as part of their formal schooling. Additionally, computer science courses are often designed for students who hope to pursue careers in the field of computer science. As such, they do not emphasize developing basic skills for immediate use, but instead seek to lay the foundations for future computer science studies (Guzdial & Soloway, 2003). In response to the growing recognition that, while not all students will pursue computer science, all students can nevertheless benefit from learning to express ideas in a computationally meaningful way, there have been several efforts to design so-called *low-threshold* computer languages that are easier to learn but still permit significant expressivity.

Beginning with the *Constructionist Logo* project (Feurzeig, Papert, Bloom, Grant, & Solomon, 1970; Harel & Papert, 1991; Papert, 1980), there have been increasingly more efforts to bring learning environments for supporting computational expression to a wide audience of learners. Wilensky and colleagues have responded to this need by creating low-threshold, text-based programming languages designed for more specific contexts such as computer-modeling in NetLogo (Wilensky, 1999a; Wilensky & Reisman, 2006; Wilensky & Resnick, 1999; Wilensky, 1995, 2001). Repenning and colleagues created Agentsheets and Agentcubes (Ioannidou, Repenning, & Webb, 2009; Repenning, Ioannidou, & Zola, 2000; Repenning & Ioannidou, 2004) which provide a graphical programming language that facilitates creating games and simulations. Another approach is the use of blocks-based programming languages (Begel, 1996; Maloney, Peppler, Kafai, Resnick, & Rusk, 2008; Resnick et al., 2009) that enable elementary-school aged children to