Lessons from the Design of Three Educational Programming Environments: Blue, BlueJ and Greenfoot

Michael Kölling, University of Kent, Canterbury, UK

ABSTRACT

Educational programming systems are booming. More systems of this kind have been published in the last few years than ever before, and interest in this area is growing. With the rise of programming as a school subject in ever-younger age groups, the importance of dedicated educational systems for programming education is increasing. In the past, professional environments were often used in programming teaching; with the shift to younger age groups, this is no longer tenable. New educational systems are currently being designed by a diverse group of developing teams, in industry, in academia, and by hobbyists. In this paper, the author describes his experiences with the design of three systems—Blue, BlueJ, and Greenfoot—and extract lessons that he hopes may be useful for designers of future systems. He also discusses current developments, and suggests an area of interest where future work might be profitable for many users: the combination of aspects from block-based and text-based programming. The author briefly presents his work in this area—frame-based editing—and suggest possible future development options.

Keywords: Blue, BlueJ, Design, Educational IDE, Frame-based Editing, Greenfoot

INTRODUCTION

In the last ten years or so, educational programming environments have become very popular for the teaching and learning of introductory programming. This was not always the case: while there have been educational systems for a long time, they were considerably fewer early in this century than today, and older systems were considerably simpler, often consisting of compilers or libraries, rather than complete programming environments. Long and heated debates used to rage among educators about the respective benefits of teaching with dedicated educational versus industry-strength tools. These debates usually remained unresolved.

In the last decade, the situation has shifted, due to a combination of reasons which we discuss below, and educational programming environments have taken a much more prominent role. They are more used, more accepted, and simply many more in number, than ever before. As a result, the design of educational environments has become a topic of considerable interest.

DOI: 10.4018/IJPOP.2015010102

In this paper we describe experiences with the design of a sequence of educational environments dating back more than 20 years. These systems are Blue (Kölling, 1999a), a programming language and development environment for teaching and learning object-oriented programming in a single, integrated system; its successor BlueJ (Kölling, Quig, Patterson, & Rosenberg, 2003), a similar environment using the Java Programming Language; and a third pedagogical system called Greenfoot (Kölling, 2010). Blue was relatively short-lived, but is of interest here because it heavily influenced the design of its successor, BlueJ. BlueJ and Greenfoot are both systems with significant user communities over a number of years (and still very much in use today), and have undergone many changes and adaptations since their first publication.

In this paper we present a short history of these systems and discuss the goals and design rationale for each, their respective target groups and how these influenced design decisions, and their scope and application. Most importantly, we discuss lessons learnt from their use with actual users, and how those lessons shaped the design of the later systems, or later versions. We also discuss their relation to other educational programming systems, similarities, possible sequences of use, and future developments. The emphasis is not on providing a complete description of each system, but to identify the trends and goals at the time of their design, and how these have changed over time. Overall, we present some lessons we learnt along the way that we hope may be of use to designers of future systems.

A SHORT HISTORY OF EDUCATIONAL PROGRAMMING TOOLS

Educational software tools are nearly as old as programming as a discipline. Ever since computer scientists started teaching others about programming, they started thinking about tools to support this challenge. In the early days, there was no difference between the tools used by professionals and the ones taught to newcomers. However, pretty soon systems started to be developed that were designed partly or primarily with beginners as users in mind.

We will not give a complete history of educational software here; instead, we mention just a few influential early systems to arrive quickly at our destination: educational development environments for object-oriented programming. This is where we will slow down and start discussion in more detail.

The first pedagogically oriented software tools were programming languages and their associated compilers. Among the early ones, BASIC (1964), Logo (1967), Pascal (1970), and Smalltalk (1972) stood out as the most used and most influential—all aiming at learners as their primary target group. The goal of these languages was partly *simplification*: taking known concepts and avoiding the complications that could arise in other existing languages at the time. BASIC and Pascal were part of this movement, introducing more rigid structure and creating higher abstraction levels in programming in the process. The other part was the introduction or appropriation of concepts and abstractions that might be more accessible to learners: micro-worlds in the case of Logo (Papert, 1980), and the adaptation of object orientation (a reasonably obscure programming paradigm at the time, introduced a few years earlier in the Simula language (Dahl, Myhrhaug, & Nygaard, 1967)) in the case of Smalltalk.

In parallel, a small number of libraries were being developed for similar purposes, aiming at programming education that was (or later became) language independent. In the 70s and 80s, a few of these dominated the educational space. One of the most successful was *Turtle Graphics*, a library first developed by Seymour Papert and others for the Logo language (Papert, 1980), and later re-implemented for countless other educational languages (Caspersen & Christensen, 2000; Python Software Foundation, 2012; Slack, 1990). Turtle graphics introduced the concept