Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 224 (2009) 57–65
www.elsevier.com/locate/entcs

Rationale Behind the Design of the EduVisor Software Visualization Component

Jan Moons[1] and Carlos De Backer[2]

Department of Management Information Systems, Faculty of Applied Economics
Universiteit Antwerpen
Antwerpen, Belgium

Abstract

The EduVisor software visualization component is a new pedagogical tool specifically developed to address some wide-spread problems in teaching object-oriented technology to novice programmers. The visualization tool is integrated in a world-class IDE, and shows the students the structure of their own creations at runtime. EduVisor is based on a solid grounding in literature and over 25 years of combined experience in teaching a CS1 course. With this component we have set the goal of helping our students progress faster through the most difficult initial stages of programming.

Keywords: Program visualization, CS1, Java programming

1 Introduction

Over the past decades software design has often been described as a *wicked* or difficult problem [10][11][12][2]. Dalbey and Linn [4] note that the average student does not make much progress in an introductory programming course. More recently, there are many reports corroborating this position. For instance, in the infamous McCracken Report [14] the authors noted that the average score on a programming test was only 22.89 out of 110 points for a sample of 216 students. As difficult as it is for students to acquire programming and software design skills, just as difficult is it for teachers to teach those skills.

This paper is concerned with a novel visualization tool that can be used as a teaching aid in CS1 courses. The tool is called EDUcational VISual Object Runtime or EduVisor, and seeks to incorporate a lot of the acquired knowledge from previous visualization projects. The goal of EduVisor is threefold. First, we want to improve students' comprehension of the concepts introduced during the CS1 course. Second,

[1] Email: jan.moons@ua.ac.be
[2] Email: carlos.debacker@ua.ac.be

doi:10.1016/j.entcs.2008.12.049

we want them to be able to debug their programs faster. Third, we want to increase the enthusiasm of students by visualizing (and thus reducing the abstraction level) their own efforts at the push of a button. The design of the tool is based on several decades of combined CS1 teaching experience and on a thorough grounding in relevant literature.

In section 2 we describe the driving forces behind the design of EduVisor. Section 3 describes the most important runtime issues one encounters during a CS1 course, which will be used as input to the design of EduVisor. Section 4 shows a small sample of the graphical representation used in the EduVisor component based on a simple use case. Section 5 provides an overview of the resulting properties of the component. Section 6 discusses the similarities and differences between EduVisor and related work. Finally, in section 7 we present our conclusions and provide an outlook on the future development of the EduVisor component.

2 Rationale of the EduVisor software visualization component

As so many educational institutions, the University of Antwerp has migrated from Pascal to C, later to C++ and finally to Java over the past two decades as the language of choice in our CS1 course. The switch to Java was made seven years ago. During our course we have noticed the same basic errors appear again and again, causing students to loose valuable time and generating frustration and disappointment.

On the highest level, these errors can be divided in compile-time errors and runtime errors. The code editor can help with some of the compile-time errors (although the compiler messages are very cryptic to novice programmers), but does nothing to aid in understanding runtime behavior. Thus, over the past five years, we have designed a visual language to illustrate the runtime behavior of a program. The language is, as we tend to say, as simple as possible and as complicated as necessary. We use this visual language when explaining programs at the whiteboard, and students' comprehension of these specific programs has improved markedly. However, when it is time to start programming their own exercises, the same errors tend happen all over again.

This is caused by several issues. First, the nature of their programming efforts is very much trial and error - which is actually a well known fact [1]. Second, the students do not go through the effort of drawing out their solutions in the way we do at the whiteboard. This is not *that* surprising - creating the visual representations for a running program takes quite some time. Encouraging however is that, when we force them to draw their programs on a sheet of paper, most of the time they are able to pinpoint the problems themselves.

Therefore we concluded that an automated software component based on our language could help students in recognizing and correcting their problems sooner. We did an extensive review of visualization components that address some of these issues, but none were found to be completely satisfactory. Section 6 talks in more