Available online at www.sciencedirect.com

ScienceDirect

Computers & Education 48 (2007) 318–328

COMPUTERS & EDUCATION

www.elsevier.com/locate/compedu

Scheduling topics for improved student comprehension of recursion

Michael Zmuda [a,*], Melanie Hatch [b]

[a] *Department of Computer Science and Systems Analysis, Miami University, Oxford, OH 45056, United States*
[b] *School of Business Administration, 2807 N. Glebe Road, Marymount University, Arlington, VA 22207, United States*

Received 1 April 2002; accepted 10 February 2005

Abstract

This paper presents the results of an experiment conducted to assess the affects of teaching recursion in two disjoint, non-consecutive units of instruction. One group of students was taught basic and advanced recursion topics in four consecutive class periods, while a second group was taught recursion in two two-period blocks that were separated by several class periods. It was unknown whether the time period separating the presentation of basic and advanced material would benefit, or hinder, student comprehension. Statistical analysis of empirical data indicates that students learning basic and advanced recursion in a consecutive unit of instruction spend less time solving their problems than the students learning the topic in two separated units, while achieving comparable scores.
© 2005 Elsevier Ltd. All rights reserved.

Keywords: Recursion; Pedagogy; Scheduling

1. Introduction

Recursion is a valuable programming technique that can yield small, elegant, efficient solutions to certain types of problems. Its importance in computer science education is readily apparent since advanced courses in data structures, compilers, algorithms, and artificial intelligence all utilize recursive algorithms. Accreditation criteria – "Computing Curricula 2001" – identify

* Corresponding author. Tel.: +1 513 529 5930; fax: +1 513 529 1524.
E-mail address: zmudam@muohio.edu (M. Zmuda).

0360-1315/$ - see front matter © 2005 Elsevier Ltd. All rights reserved.
doi:10.1016/j.compedu.2005.02.003

recursion as a required component of fundamental programming. Recursion, however, is a difficult topic for students to learn. Students often do not acquire an accurate model of recursion even after receiving instruction. Students can easily confuse recursion with iteration or fail to acquire any consistent model at all (Mayer, 1981; Kahney, 1989; Kessler & Anderson, 1989; Rogalski & Samurcay, 1990; Ginat & Shifroni, 1999). Clearly, it is important to identify techniques that assist students in improving comprehension.

A variety of teaching techniques have been developed to facilitate the learning of recursion. Murnane (1991) suggests duplicating the source code multiple times to help insure the students do not acquire an iterative model of recursion. Er (1995) describes an approach in which students are presented a detailed history of process activation frames, one for each recursive call. The advantage of the activation frames is that they explicitly illustrate the notion of a separate set of local variables and distinct return points, while using only one copy of the source code. Kruse (1982) suggests the use of trees to graphically depict the sequence of function calls and their parameters. The use of trees allows students to identify potentially inefficient recursive procedures by locating large, common subtrees. Give'on (1991) indicates that turtle graphics is a good tool for teaching recursion due to its richness and intrinsically interesting subject matter. Wilcocks and Sanders (1994) use graphics as the tool to teach recursion on non-graphical applications. These graphical tools dynamically display procedure activation frames as recursive function calls are made. The appearance and disappearance of these frames helps students obtain a better understanding of recursion.

The pedagogical experiment conducted in this paper was motivated while constructing a syllabus for a programming course that required students to be introduced to the concept of recursion and then taught to apply recursion to new problems. While trying to balance the other course's other objectives, four class periods (1 1/3 weeks) were allocated to recursion. This material was divided into *introductory recursion* and *advanced recursion* units of instruction. It was initially unclear how to schedule the delivery of this material. One scheduling approach delivers all of the material in four consecutive class periods. The advantage of this approach is that students see an uninterrupted presentation of this naturally cohesive topic. On the other hand, such a tight schedule could result in confusion during the advanced topics if the students did not have time to adequately digest the introductory material. A second approach is to separate the two units of instruction by an unrelated topic. Having extra time to master introductory material may help the students be better prepared for advanced topics, thereby learn the more complex material more easily and thoroughly. One potential disadvantage, however, is that a separation between the two units may disrupt the flow of a naturally cohesive topic and/or students may start to forget the introductory material and be forced to review previous material when the advanced material is covered. Some institutions use this delivery schedule, although at a much larger time scale than addressed here, by addressing introductory material in CS I and advanced recursion in CS II, data structures, or algorithms. Many CS I/CS II textbooks mirror this approach by dedicating a section or two to basic recursive topics; topics involving recursive thinking and recursive problem solving are left to other courses.

The two delivery approaches are referred to as the *consecutive* and *intermittent* approaches. The experiment described in this paper was conducted to help determine if one of these approaches was superior. Either approach is amenable to the teaching techniques appearing in the literature. In fact, the techniques of Er (1995) and Kruse (1982) were extensively used in the in-class presentation.