ABSTRACT

The Unified Modeling Language has been shown to be complex and difficult to learn. The difficulty of learning to build the individual diagrams in the UML, however, has received scant attention. In this article, we consider the case of the UML sequence diagram. Despite the fact that these diagrams are among the most frequently used in practice, they are difficult to learn to build. In this article, we consider the question of why these diagrams remain so difficult to learn to build. Specifically, we analyze the process of learning to build sequence diagrams in the context of cognitive complexity theory. Based on this analysis, and drawing on the theory of learner-centered design, we develop a set of recommendations for presenting the sequence diagram building task to the student analyst to reduce the complexity of learning how to build them.

Keywords: cognitive complexity; learner-centered design; sequence diagrams; systems analysis and design, unified modeling language (UML)

INTRODUCTION

Object-oriented analysis and design (OOAD) is the dominant software design method among practitioners. The Unified Modeling Language (UML) is an ISO standard graphical modeling language used in OOAD (International Organization for Standardization, 2005). The UML consists of a set of diagrams and associated notations, where each diagram represents a different view of a software analysis and design model—structure, interaction, or state (Blaha &
In the OOAD process, analysts develop a set of objects, where each object has a set of attributes (data that the object knows about) and behaviors (operations that the object can perform). Objects interact by invoking one another’s operations to fulfill the high-level behaviors required of the system, representing the business logic of the to-be software system. The analyst captures this logic in a set of UML sequence diagrams (SDs). Figure 1 shows an example of such a diagram (all diagrams in this article were developed using Visual Paradigm, a software tool from Visual Paradigm International).

SDs are crucial building blocks in system design. Analyst errors in building them lead to significant rework efforts or serious software defects at implementation time, which in turn leads to increased costs. Thus, it is critical that analysts have the skills to produce high-quality SDs. While sequence diagrams are among the most widely used diagrams in the UML in practice (Dobing & Parsons, 2006, 2008; Fowler, 1997) and are critical to the design process, they are difficult to learn to build (Bolloju & Leung, 2006; Siau & Loo, 2006).

The OOAD process consists of five basic steps: (1) developing high-level requirements, (2) use case analysis, (3) domain data modeling, and (4) building sequence diagrams (George, Batra, Valacich, & Hoffer, 2006), and (5) building a class diagram. In steps (1) and (2), the analyst develops a description of what the system should do. In step (3), the analyst develops the structure of the system from a data perspective and, in step (4), defines how the system should achieve the behaviors required. In step (5), the analyst refines the structure of the system to include both data and behavior. Good, clear techniques and heuristics are available for steps (1), (2), (3), and (5) through many textbooks and other sources. Textbooks covering OOAD (George et al., 2006), as well as other sources (Fowler, 1997), offer a sizeable set of advice on step (4). Even with so many useful guidelines and heuristics, however, learning to build these diagrams remains difficult.

In this article, we consider the task of learning to build SDs. Although much thinking has been done on the problem of developing guidelines and heuristics for building a good SD, and much work has been done on the difficulties of learning OOAD, there has not been much discussion of the process of learning to build SDs in particular. This is our focus in this article. Specifically, we concentrate on two questions:

- What are the complexity factors inherent in the process of learning to build SDs?
- How can these factors be mitigated to minimize the cognitive load of learning to build SDs?

In this article, we apply Reeves’ Learner-Centered Design (LCD) framework (Reeves, 1999) to the task of learning to build SDs. We chose
Is Extreme Programming Just Old Wine in New Bottles: A Comparison of Two Cases
www.igi-global.com/article/extreme-programming-just-old-wine/3341?camid=4v1a

An Overview of Learning Object Repositories
www.igi-global.com/chapter/overview-learning-object-repositories/28547?camid=4v1a