Improving Sequence Diagram Modeling Performance: A Technique Based on Chunking, Ordering, and Patterning

Thant Syn, School of Business Administration, University of Miami, Coral Gables, FL, USA
Dinesh Batra, College of Business Administration, Florida International University, Miami, FL, USA

ABSTRACT

The Unified Modeling Language (UML) has become the de facto standard for object-oriented software development. It has been widely adopted in both training and practice. However, UML has often been criticized for being overly complex and difficult to learn for novice analysts. Although some research studies have identified specific novice difficulties in learning UML, there is little research proposing viable techniques for addressing these difficulties. In particular, there is a lack of research evaluating the usability of the sequence diagram (SD), which models the interactions among objects of a software application. This paper reports a research study that proposes a technique called “CHOP” (CHunking, Ordering, Patterning), which is designed to improve novice analyst performance in modeling an SD. The CHOP technique is based on the Cognitive Load Theory (CLT) and was developed by addressing the three types of cognitive load encountered by novices. An experimental study testing the efficacy of the CHOP technique in comparison to the worked-example approach indicated that the CHOP technique significantly improves novice analyst’s ability to model interactions among objects; however, the worked-example technique was the more efficient during training. The study also found that subjects using the CHOP technique rated its perceived usefulness significantly higher than subjects using the worked-example approach.

Keywords: Cognitive Complexity, Cognitive Load Theory, Novice Analyst, Sequence Diagram, Unified Modeling Language (UML)

INTRODUCTION

Contemporary practices in systems development indicate a shift from traditional software development to object-oriented (OO) development (Armstrong & Hardgrave, 2007). It is widely acknowledged that the Unified Modeling Language (UML) has become the de facto standard for modeling OO software systems in traditional development (Kobryn, 1999; Pender, 2003) as well as agile development (Erickson, Lyytinen, & Siau, 2005). UML is primarily used in specifying, visualizing, constructing, and documenting the artifacts of software systems (Booch, 2000).
While UML increases in popularity, concerns about its usability and complexity have been raised (Erickson, 2008; Siau, 2000; Siau, Long, & Ling, 2010). Bolloju and Leung (2006) conducted an experiment to understand the usability of UML diagrams and found multiple errors in solutions developed by novice analysts. They attributed these errors to novice analysts’ inexperience in problem-solving techniques such as decomposition; indeed, Burton-Jones and Meso (2006) found that analysis diagrams that manifest better decompositions increase analysts’ understanding of a domain. Siau and Loo (2006) reported that UML is difficult to learn for novice analysts and recommended that training should focus on core UML diagrams. However, with some exceptions, modeling techniques that may ease the difficulties of learning UML have, in general, yet to be proposed and evaluated.

Sien (2011) found severe difficulties in modeling sequence diagram (SD) and class diagram. Based on an experimental study, the author concluded: “Essentially, students do not know how to produce appropriate UML class and sequence diagrams.” In the Sien (2011) study, 67% of the subjects had significant errors and misunderstandings. About 70% of the subjects perceived producing sequence diagrams as difficult or very difficult. Most subjects had difficulty relating sequence diagrams with class diagrams.

Surveys have found that the class, the use case, and the sequence diagrams/models are the three UML constructs most commonly used by software developers (Dobing & Parsons, 2006). Given that a software system eventually implements classes, the class diagram is considered the most important UML diagram (Larman, 2005). The use case diagram depicts system behavior from an overview level, while the corresponding written use case details user requirements. The SD (or the collaboration/communication diagram) can be employed to translate use case activities into object interactions, which can then be used to develop class diagrams (Ambler, 2004; George, Batra, Valacich, & Hoffer, 2007; Selonen, Koskimies, & Sakkinen, 2003). Larman (2005) notes that novice UML modelers don’t pay enough attention to interaction diagrams (such as SD) even though they are incredibly valuable.

In most OO methodologies, the use case serves as the starting point for the analysis and design that lead to the class diagram. There are two major aspects of a class: attributes and operations. The attributes of the class approximately correspond to attributes of the data model, much like in an entity-relationship (ER) model. A large number of classes are entity classes because the backend of most transaction-based information systems has been traditionally built on relational databases (Siau, Nah, & Cao, 2011). For modeling entity classes, one can extrapolate knowledge gained from research in data modeling (Topi & Ramesh, 2002) to help novice analysts. The operations aspect, however, usually requires the use of SD (Quatrani & Palistrant, 2006; VanderMeer & Dutta, 2009), but there are currently no systematic guidelines for SD modeling.

OOSA&D textbooks and practitioner literature typically discuss how to develop an SD. These sources typically employ the “worked example” approach, which has been found to be a viable approach for many tasks (Cooper & Sweller, 1987). A worked example includes a problem accompanied by a solution, usually given together with some reasoning specific to the problem. Learning is imparted if the novice learner can establish the mapping between the problem and its solution. The simplicity of the worked example approach makes it quite popular. The worked-example approach can reduce cognitive load in some situations and is more effective than the means-ends analysis approach (Cooper & Sweller, 1987; Sweller & Cooper, 1985). Because of its simplicity, the worked example may also be the easier and faster approach for training novices. In more complex tasks such as SD modeling, other approaches may need to be proposed and examined.

This study was guided by the design science framework (Hevner, March, Park, &
Integration of Relational and NoSQL Databases
(2018). *Bridging Relational and NoSQL Databases* (pp. 239-281).
www.igi-global.com/chapter/integration-of-relational-and-nosql-databases/191985?camid=4v1a

Co-creation and Collaboration in a Virtual World: A 3D Visualization Design Project in Second Life
*Cross-Disciplinary Models and Applications of Database Management: Advancing Approaches* (pp. 323-335).
www.igi-global.com/chapter/creation-collaboration-virtual-world/63672?camid=4v1a