This is the second of the two special issues on the Unified Modeling Language (UML); the first appeared in the January-March 2008 issue of the Journal of Database Management (Batra, 2008), and included four papers on the topic. The first issue covered UML application topics: the use of UML in practice (Dobing & Parsons, 2008), the organizational inadequacies of UML (Smolander & Rossi, 2008), the need to maintain seamless traceability of business rules in UML (Loucopoulos & Kadir, 2008), and assessment of the effectiveness of domain models as aids to application models developed using UML (Reinhartz-Berger & Sturm, 2008). In the current issue, the three papers address cognitive and ontological concerns related to UML.

Assuming the novice perspective, VanderMeer and Dutta (2009) assert that UML is complex and difficult to learn. They evaluate the UML sequence diagram, which may be viewed as a bridge between a use case and its class diagram. By employing the cognitive complexity theory, they explore the difficulty in learning the sequence diagram. They then employ the principles of learner-centered design to develop recommendations for the student analyst engaged in learning and developing the sequence diagram.

Gemino and Parker (2009) investigate the usability of use cases and use case diagrams. Using the Cognitive Theory of Multimedia Learning (Mayer, 2001), they examine whether use case diagrams improve effectiveness of novice users working with use cases, which are text-based. They evaluate effectiveness by measuring comprehension, retention, and problem-solving performance. The results indicate that participants viewing the
use cases with the supporting diagram perform significantly better in retention and on problem-solving tasks. The study implies that use cases need to be complemented by use case diagrams to improve participant understanding.

Evermann and Wand (2009) employ an ontological perspective to examine behavioral aspects of conceptual modeling, focusing on constructs that describe change and interaction. They argue that conceptual models based on a strong ontological foundation should be used instead of software models to represent an application domain. While languages for software design, such as UML, are available and widely used, no generally accepted language exists for conceptual modeling. The authors provide rules for mapping ontologically-based conceptual models to software models. This approach can lead to ontologically-grounded software models represented in a popular language such as UML.

The seven papers in the two special issues cover a variety of UML topics. Some of the papers tend to suggest that UML suffers from a litany of shortcomings. Nevertheless, UML continues to enjoy significant adoption (Selic, Ramackers, & Kobryn, 2002), which indicates that the practitioners find UML useful and reasonably easy to use. The editorial preface in the previous issue discussed the possibilities for future UML research in general; this preface will attempt to reconcile some of the apparent contradictions between its usability problems and adoption while suggesting additional cognitive topics for future research.

Object-oriented technology has often been promoted as a silver bullet, capable of solving many of the longstanding ills facing the software industry (Grossman, Aronson, & McCarthy, 2004). While it is true that UML has achieved tremendous popularity and is rapidly becoming the standard for object-oriented systems development, there are many who feel that it is too difficult to use and is not fulfilling its promise. Commonly heard complaints about UML are that it is too big and too complex (Siau & Cao, 2001); semantically imprecise (Evermann & Wand, 2006); difficult to learn (Siau & Loo, 2006); and has poorly-defined syntax, semantics, and spatial layout, and limited customizability (Tilley & Huang, 2003). Additionally, some feels that it offers inadequate support for component-based development, and does not allow for the easy interchange among model diagrams (Kobryn, 2002).

Recent surveys however, indicate that some of these problems are not considered particularly serious by practitioners. For example, the complexity of UML is frequently cited as an impediment to its ease of use. Grossman et al’s (2004) questionnaire-based survey findings indicate that the respondents find UML to be fairly understandable. Further, the respondents rate UML as flexible and accurate. Agarwal and Sinha (2003) used ease-of-use measures derived from Davis (1989) and found that subjects perceive use case, class, and state diagrams as reasonably easy to use. In the Dobing and Parsons (2008) survey, respondents did not find
serious problems with UML usage although there was a strong concern about the lack of UML support for user interfaces, and some concern about the lack of support for security and database design. Lang and Fitzgerald (2006) note high adoption rates for some UML constructs such as use case diagrams/scenarios (72%), class diagrams (62%), and state diagrams (50%). Similar adoption results are reported by Dobing and Parsons (2006), although Grossman et al. (2004) report a somewhat lower (27.5%) overall adoption rate for UML.

For now, UML appears to be here to stay. However, the apparent contradictions between its usability problems and adoption do raise some issues. Is UML merely having its “15 minutes of fame” and we can expect it to fade away with time, or will it go on to dominate the software modeling arena? Is UML a complex language laden with rarely used constructs, or is it a flexible language with rich features that can easily be adapted to various contexts? Is UML complexity an artifact of the usability research focus on the novice designer? Is the widespread adoption of UML among practitioners an artifact of the need for standardization in practice? How will UML fare in the increasingly popular agile development environment? Future research can separate the myths from the facts, and provide answers to what may appear to be contradictory findings.

An obvious shortcoming of UML is its lack of grounding in systems theory (Kast & Rosenzweig, 1972). The unit for analysis in UML is the use case, but it does not lend itself easily to decomposition. If we assume that the use case represents an aspect of an information system and provides some functionality, we should be able to decompose it into smaller use cases. There is no reason a use case should not be considered a sub-system; after all, it has parts that interact with each other as in any system or sub-system. In fact, use cases have been considered at very levels of abstractions.

Cockburn (2000) lists five levels of abstractions from highly summary to highly detailed for use cases: cloud, kite, sea level, fish, and clam, but he does not discuss how UML can provide transitions from one level to another. Perhaps, it is not the responsibility of the language itself to facilitate the transition. If so, we need a modeling approach that can provide the mapping from one abstraction level to another. Surprisingly, this fundamental systems issue has not attracted much attention, and we continue to see textbooks and practitioner literature in which the customary method is to show an information system as a small number of interacting use cases in a use case diagram with a single-level description of each use case.

In contrast, the process-based structured systems analysis (Gane & Sarson, 1977), along with its data flow diagram, has a solid grounding in systems theory. Functional decomposition is common in the structured approach, and one can discuss a system at any level of abstraction. Although the data
flow diagram is not appropriate for object-oriented development and it is unlikely to supplant UML, it does provide a lesson in systems decomposition, and in combining data, processes, and actors. After all, it is a basic premise of object-oriented development that data and processes be presented in an integrated fashion (George, Batra, Valacich, & Hoffer, 2007).

Some studies have concluded that it is difficult to model a correct and consistent application using UML, and to understand such a specification (Peleg & Dori, 2000). Dori (2002) claims that UML requires several models to completely specify a system, and is, therefore, low in usability. Dori (2001) has proposed the Object-Process Methodology (OPM) to provide a simpler methodology that offers a single, integrated graphic model. It achieves model integration by incorporating the three major system aspects—function, structure, and behavior—into a single model in which both data and processes are adequately represented without either one suppressing the other.

There is implicit support for the tenets of the OPM approach in the study by Masri, Gemino, and Parker (2008), which was conducted to evaluate the effectiveness of combining elements of UML diagrams when presenting information. They employed the notion of intrinsic, extraneous, and germane cognitive load to suggest ways to reduce the overall cognitive load by combining appropriate pieces of diagrams. By adding the element of interaction with the combining of diagrams, they found that subject performance increased significantly. One may thus infer that implicitly, there is a case for data flow diagram (DFD), which also combines function, structure, and behavior. Should we then compare, say, UML and DFD, to assess which one is better? It is possible that such a study would reveal some interesting findings. However, the DFD, by itself, is not amenable to object-oriented development, so the practical value of such a comparative study is questionable. Almost all new information systems are developed based on the object-oriented approach.

Interestingly, there is a methodology that employs DFDs to come up with class diagrams. In an attempt to tie the structural and behavioral diagrams together, Shoval and Kabeli (2001) have proposed a methodology called Functional and Object-Oriented Analysis and Design Methodology (FOOM), which tries to blend the DFD, the entity relationship diagram (ERD), and object-oriented (OO) constructs. A laboratory experiment that compares FOOM, OPM, and Masri et al.'s (2008) combination diagram approach can likely provide findings that can improve the design and effectiveness of UML. Since there are subtle differences in the constructs underlying the three approaches, such a study will invariably encounter the usual internal validity issues, especially with regard to the equivalence of the treatments (Gemino & Wand, 2001). Nevertheless, as long as some threshold level of equivalence is attained, we should obtain useful results that could potentially resolve the criti-
cism that UML is weak in presenting a combined view of data and processes.

One may argue that the UML sequence diagram already blends function, structure, and behavior. After all, a sequence diagram is the principal behavior model for representing system functionality given that it is based on a use case. Also, it can only be developed accurately if the structure of entity objects is known, perhaps from an entity-relationship (ER), or a domain model. At present, this argument is weak, however, because the entity objects are shown simply in a linear fashion without revealing the underlying relationships. Yet, the sequence diagram can be improvised, perhaps by depicting a domain (data) model alongside. Additionally, the sequence diagram can be further improved by an interface model, and a logic model (e.g., a decision table or a decision tree) placed alongside the sequence diagram or available on demand using a software tool.

Another concern regarding UML is its expansion of an already inflated specification: the number of diagrams in UML 2.0 has been augmented to 13, from the nine found in UML 1.0. Most of the diagrams added in UML 2.0 depict higher level views of the system under development, for the purpose of enhancing component-based development (Kobryn, 2002). The ideas behind the original UML diagrams have not changed much and are expected to remain the same for the foreseeable future (Dori, 2002; Miller, 2002; Selic et al., 2002). However, the number of diagrams is clearly large and will continue to raise usability concerns.

Yet, UML has seen remarkable adoption. The two findings—UML’s complexity and its popularity—reveal a contradiction that needs to be resolved. One possibility is that the practical complexity of UML is limited (Siau, Erickson, & Lee, 2005), and that UML is, after all, a flexible language; analysts and designers can limit their cognitive complexity by adopting only what it is the most useful to them. This hypothesis may well explain the seemingly contradictory points of view regarding UML, but this claim needs to be strengthened by empirical validation.

This leads us to the next research question: do we really know how UML is used in practice? Questionnaire-based studies have provided some clues. For example, we know that the use case (and the use case diagram), the class diagram, and the sequence diagram are the most often used UML construct (e.g., Dobing & Parsons, 2006). Multiple case-studies, protocol-based studies, and action research may reveal how these constructs are used. Naturally, many research questions arise: are the UML constructs used formally with the support of case tools? Is the sequence diagram used to bridge the use case with the class diagram, or is it used to clarify the coding logic to the developer? Is UML primarily a way of communicating (i.e., informal), or a means to design software (i.e., formal), or both? Does it provide cognitive support to developers? Does it aid comprehension? Does it reduce cognitive load? Do programmers perform better at writing code when aided by UML documentation? Do programmers perform better at
software maintenance when aided by UML documentation?

A few recent studies provide preliminary answers to such questions (Costain, 2008; Costain & Srinivasan, 2008; Lange, Chaudron, & Muskens, 2006). Costain (2008) found that UML documentation does support programmers in writing code, in creating internal representations of the problems, in aiding comprehension, and in offloading notations from working memory to an external representation. Still, programming performance is largely dependent on experience. Industry-experienced developers use UML documentation more than novices do. Designers customize UML notations when offloading diagrams, say, on to paper. Overall, there is some evidence that UML does provide cognitive support.

Lange et al (2006) conducted a study on actual UML usage and found several significant problems suggesting that UML could be used somewhat loosely. For instance, they frequently found methods in class diagrams that are not called in sequence diagrams. Methods that are not called might not be used in interactions; alternatively, they might not be sufficiently described in terms of their interactive functionality. Conversely, they found messages in sequence diagrams that do not correspond to any methods in class diagrams. Each message that an object in a sequence diagram receives must correspond to a method in the object’s class interface; otherwise the meaning of the message is unclear. The researchers also found classes without methods, which violates the object-oriented paradigm, particularly the concept of encapsulation. A class without methods cannot interact with other classes and is, therefore, incomplete.

Lange et al (2006) also conducted a controlled experiment with 110 students and 48 practitioners to evaluate the usage of UML. The results showed that defects often remain undetected, even if the model is read thoroughly. For example, 61 percent of readers did not detect a sequence diagram message that lacked a corresponding method in the class diagram. Although several studies have suggested that UML has been widely adopted, the Lange et al (2006) study suggests that the nature of adoption may be in a perfunctory, loose fashion.

With the growing popularity of agile development with its informal practices and the need to simplify modeling, one wonders if UML will be used in the most formal manner (Ambler, 2004). Since the focus of agile development is on working software, it is likely that the syntactic inconsistencies will be ignored and semantic ambiguity will be tolerated. A “light” version of UML may address the complexity of UML. In a questionnaire-based survey of 180 students, Wycza and Marcinkowski (2007) found that more than 90 percent of the respondents prefer a light version of UML 2.0. Four diagrams were selected as the components of a light version: use case, class, activity, and sequence diagrams.

It seems that for most tasks, a “light” or limited version of UML will
be employed to manage the practical complexity of UML (Dobing & Parsons, 2006; Siau et al., 2005). The use case, class, activity, and sequence diagrams may continue to be found the most useful as new additions to UML are proposed for specific applications. Developers will pick and choose and improvise based on their context, methodology, and culture.

How will we use UML or a similar language in future? Systems analysis methods have always been evolving and this trend can be expected to continue. In the beginning, there were text descriptions. Diagrams quickly followed. These diagrams were soon managed using CASE tools. However, the CASE tools presented models as if they were on paper. For some reason, we have never seriously crossed the “paper-like” threshold of using the computer interface for presenting analysis information. So what awaits us if we do? We need to consider presentation options such as animation, narration, dynamic highlighting, diagram combination, video, and many others. These presentation options are already available, but we have provided little guidance on how to use these interface elements effectively. Eventually, the goal is to conduct research that takes our well specified modeling techniques and presents the information contained within them in a way that is easily, clearly, and accurately understood.

In this research arena, academics have the opportunity to take the lead over the practitioners because of the need to understand both analysis and design methods, and cognitive theory. For example, cognitive complexity, among other sources, may provide a theoretical basis to study the usability of UML. The theories of cognitive complexity have been applied to software engineering processes from coding or programming (Cant, Jeffery, & Henderson-Sellers, 1995) to OO system design (Rosson & Alpert, 1990), and recently to UML (Sin & Batra, 2007). Cognitive complexity principles are also applied to other areas of research such as the design of user interfaces for information-intensive products (Reeves, 1999) and computer-based learning systems (Norman & Spohrer, 1996; Soloway & Pryor, 1996).

This special issue is the second of two and provides the concluding set of papers on UML topics that resulted from a call for papers made in late 2006. A large number of high-quality manuscripts were received and each submitted manuscript was sent out to three reviewers. Each of the accepted papers in the special issues went through one round of comprehensive revision based on reviewer comments. I thank the reviewers, most of whom are members of the AIS Special Interest Group on Systems Analysis and Design (SIG-SAND), for their meticulous reading of the papers, and for their substantive comments, which improved the quality of the papers. I thank Andrew Gemino for his feedback on some of the ideas presented in this preface. I am grateful to the editor-in-chief, Keng Siau, who counseled me on various matters. I am confident that the readers of the journal
will find the papers of high quality, and I hope the ideas and issues discussed herein will lead to future research in this area of considerable contemporary interest.

REFERENCES


Dinesh Batra is a Knight-Ridder research professor at the Department of Decision Sciences and Information Systems in the College of Business Administration at the Florida International University. Dr. Batra has published articles in journals such as Management Science, Communication of the ACM, Journal of MIS, International Journal of Human Computer Studies, Data Base, European Journal of Information Systems, Journal of Database Management, Communications of the AIS, Decision Support Systems, Requirements Engineering, Computers and OR, Information Systems Management, and Information & Management. His research interests focus on systems analysis and design methods, usability issues in systems and databases, and distributed development. He is currently an associate editor in the Journal of Database Management, Communications of the AIS, and Information Systems Management. He is a co-author of the book Object-Oriented Systems Analysis and Design. He has served as a president of the AIS Special Interest Group on Systems Analysis & Design (SIGSAND).