Unified Modeling Language: Theoretical and Practical Complexity

John Erickson
University of Nebraska-Lincoln

Keng Siau
University of Nebraska-Lincoln

UNIFIED MODELING LANGUAGE: THEORETICAL AND PRACTICAL COMPLEXITY

John Erickson
University of Nebraska–Lincoln
jeericks@unlserve.unl.edu

Keng Siau
University of Nebraska–Lincoln
ksiau@unl.edu

Abstract

Systems development efforts are gradually moving to encompass object-oriented (OO) analysis and design methodologies. The Unified Modeling Language (UML) was adopted by the Object Management Group as the standard visual modeling language for OO systems development. However, UML has been criticized in the literature for its complexity, inconsistent semantics, and ambiguous constructs. A set of complexity indices for UML in aggregate, and the nine diagramming techniques individually, was compiled recently. Since this set of metrics is based on inclusion of all possible constructs in UML, it thus provides an indication of the theoretical (maximum) complexity of the modeling methods. This research aims to differentiate (for UML) the theoretical complexity from a more practical complexity, and also to define and develop measures for practical complexity, by examining three distinct software system types.

Keywords: Systems analysis and design, OO systems development, unified modeling language, complexity

Introduction

Systems development efforts are gradually moving to encompass object-oriented (OO) analysis and design methodologies, largely as a response to the emerging dominance of OO programming languages. As a tool in OO Analysis and Design efforts, the Unified Modeling Language (UML) was adopted by the Object Management Group (a standards body in the industry) as the standard visual modeling language for OO systems development. UML has become more important to systems development efforts in the past five years, and is one tool that agile systems developers have come to depend on as crucial to their efforts. However, UML has also been criticized for its complexity, inconsistent semantics, and ambiguous constructs (Dobing and Parsons, 2000; Dori, 2002; Duddy, 2002; Kobryn, 2002; Zendler, Pfeiffer, Eicks, and Lehner, 2001).

Constructing models of systems is an approach that developers have devised to help manage the task of processing some of the cognitively complex tasks involved in systems development. As a popular modeling method, and in terms of complexity, a set of complexity indices for UML in aggregate, and the nine diagramming techniques individually, was compiled recently (Siau and Cao, 2001). Since this set of metrics is based on inclusion of all possible constructs in UML, it thus provides an indication of the theoretical (or maximum) complexity of the modeling methods.

In reality, however, it can reasonably be supposed that not all of the constructs are used all of the time when building systems. This phenomenon is known in software development as the (heuristic) 80/20 rule, which states that 80% of software programs are written using only 20% of the language constructs. Developers often refer to this 20% as the kernel, or core of a language. Kobryn (2002) proposed that the situation might be analogous to and applicable to UML as well. He proposed that 80% of UML models developed might be created by using just 20% of the (UML) language. This relates to practical or use-based complexity that may be markedly less than theoretical complexity. To accomplish the purposes of this research, three distinct software system types will be examined as part of this study are Enterprise systems, Web-based systems, and Real-time systems.

This research aims to (1) differentiate theoretical complexity as defined by Rossi and Brinkkemper (1996) from a more practical complexity, as well as (2) to define and measure practical complexity, specifically as related to UML.

Literature Review

UML, as a modeling language, is really all about creating models of software systems. Models are an abstraction of reality, meaning that we cannot, and really do not care to model in total reality settings, simply because of the complexity that such models would entail. Without abstraction, models would consume far more resources than any benefit gained from their construction. Thus, a model constitutes a view into the system. UML originally proposed a set of nine distinct modeling techniques representing nine different models of the system. The techniques can be separated into structural (static) and behavioral (dynamic) views of the system. UML 1.5 is the latest version of the modeling language.

For developers the existence and utility of the nine different diagramming techniques should bring the complexity issue clearly into focus. From a development perspective including comprehension and cognition, a valid question is, "How can anyone (developer or end-user) possibly understand all of these techniques well enough to successfully use them in developing information systems?" The apparent complexity of UML as a modeling tool has been the subject and topic of much research in the area (Dobing and Parsons, 2000; Kim, Hahn, and Hahn, 2000; Rossi and Brinkkemper, 1996; Siau and Cao, 2001).

Theoretical Foundation

The study of comprehension is a relatively new area of research. J. R. Anderson is a cognitive psychology researcher and theorist whose models have often been used in Management Information Systems (MIS) and Human-Computer Interaction (HCI) research. Anderson's Atomic Components of Thought (ACT) (1998) attempts to divide cognition into its component parts.

ACT breaks knowledge into two parts, declarative knowledge and procedural knowledge. Declarative knowledge is similar to that knowledge captured in an encyclopedia or dictionary – it is a list of what we know. Procedural knowledge, on the other hand is knowledge about how things work. Procedural knowledge depends on declarative knowledge as a starting point, but uses that knowledge to help solve problems (Anderson, 1998). Declarative knowledge is produced in chunks, and is constrained by our cognitive limits (Miller, 1956). Procedural knowledge is used to create production rules that describe productions, or specific steps we use to solve common problems (Anderson, 1998).

Drawing once again on the systems analysis domain, we can see how critical it is that those who develop the modeling methods (such as UML) do not make the methods too complex. The reasoning works as follows. As domain experts, the methods engineers have their expertise to draw upon, and will have participated in many development efforts. So, these experts have at their disposal highly developed and specialized stores (productions) of domain knowledge, and will be able to quickly recall from LTWM (Long-Term Working Memory) far more of that domain knowledge than the typical end-user likely knows about modeling methods (Ericsson and Kintsch, 1995). That means that what may be glaringly apparent and simple to the methods engineers, may be quite complex and simply beyond the processing capacity of the users using the methods.

For the purposes of this research, complexity will be approached from two separate but closely related perspectives; cognitive complexity as related to human perception, and structural complexity, as related to the structural properties of the diagramming techniques found in the UML modeling approaches. In this context cognitive complexity can be defined as the mental burden people face as they work with systems development constructs.

The research proposes to adopt the ideas on the definition of structural complexity as proposed by Briand, Wüst, and Lounis (1999), in which the physical (structural) complexity of diagrams affects the cognitive complexity faced by the humans using the diagrams as aids to understand and develop systems. (See Figure 1 below)

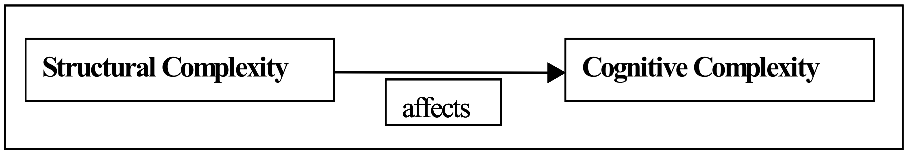

Figure 1 (Adapted from Briand, Wüst, and Lounis, 1999)

Since cognitive complexity, as defined for this research, is difficult and even impossible to measure, structural complexity will be used to explain cognitive complexity. Structural complexity can be defined as a function of the number of distinct elements (or constructs) that constitute a given diagramming technique. Rossi and Brinkkemper (1996) formulated seventeen distinct definitions relating to the structural complexity of each diagramming technique. Using all available constructs (the metamodel component); these definitions form an estimate of the total structural complexity of the diagramming technique, which this research terms theoretical complexity.

Structural complexity as a part of the structural characteristics of the information or modeling system, refers to the elements, or constructs that comprise a given diagramming technique. These constructs would include meta-construct types such as objects (classes, and interfaces), properties (class names, attributes, methods, and roles), and relationships and associations (aggregations generalizations, specializations).

Kobryn (2002) proposed that 80% of systems are developed by using only 20% of the language constructs. This can be tied to a practical or use-based complexity that may be markedly less than the theoretical complexity.

The conjecture here is that UML's practical complexity is measurable, and is less than the existing metrical results developed and verified by others (Rossi and Brinkkemper, 1996; and Siau and Cao, 2001). Thus, in addition to theoretical complexity, a set of metrics for estimating practical complexity can be developed based on the most commonly used constructs rather than all constructs. This argument is supported by Siau *et al.*'s (2002) results, which indicate that theoretical complexity is not a good predictor of the complexity faced by users. This provides motivation for an extension of that work, with the goal being to determine the practical complexity of UML.

In summary, we propose that comprehension is at least roughly inversely proportional to complexity. Then, especially for non-experts, it would be beneficial to reduce the complexity of models, and modeling methods where possible. Since we have highlighted the relationship and differences between structural and cognitive complexity, we propose that a measure of structural complexity via the method(s) proposed will also provide an implication for understanding cognitive complexity as well.

Research Methodology

This research extends the study of Siau *et al.* (2002) by attempting development of practical complexity measures. As indicated earlier, the aim is to define and measure practical complexity, as related to UML. Specifically we propose that this be accomplished by defining the 20% of the constructs that are used to construct 80% of the systems developed (i.e., the kernel of UML), as a way to get at practical complexity.

Research Procedure and Subjects

A series of questionnaires will be sent to 30–50 OO development practitioners in a Delphi-type study, asking them to reach a consensus and determine the UML constructs that are most important to them, and that they most commonly use. The subjects need to have at least two years of working experience on UML. The Delphi study will be done to attempt to identify the core UML constructs, and also the core constructs for three separate types of systems. A Delphi approach was chosen because it attempts to elicit a consensus of experts in a given area.

UML Domains

UML 2.0 is currently under development, with five Request for Proposals (RFPs) forming the basis of the revision effort. A common theme of the five proposals is the idea of extensibility to provide functionality for different types of systems, and different languages, therefore this research proposes an examination of three distinct system types.

The three software system types to be examined as part of this study are Enterprise systems, Web-based systems, and Real-time systems. These specific system types were selected for three reasons. First, these system types were chosen because many organizations have increasingly turned to the use of such systems in many different settings and industries, and second, these particular types of systems require quite different development efforts. Finally, there appears to be movement toward developing special UML extension mechanisms in each area (Douglass, 2000; Conallen, 2000; and Marshall, 2000).

Enterprise Systems

Enterprise systems have become a focal point for systems development efforts in many organizations across industries. Enterprise Resource Planning (ERP) has emerged as one of the dominant corporate strategic initiatives of the past decade (McKeown, 2001). Essentially, ERP aims to combine and integrate all the information resources of an organization into one centralized system (Mello, 2002). The focus of enterprise systems is on the organization as a whole, and development efforts necessarily focus on systems from that perspective.

Web-based Systems

Similarly, and notwithstanding the dot.com crash, Web-based application development has become a primary push for many organizations, again with little regard for industry boundaries. However, Web-based application development focuses generally upon connecting systems with the Internet, and is usually but not exclusively related to sales processes, and the underlying sales databases (Conallen, 2000).

Real-Time Systems

Finally, many organizations have developed and depend upon systems that respond and are used in real-time environments. Real-time applications approach systems development from an entirely different perspective. Real-time systems can often be found embedded in such diverse items as automobiles, watches, and communication systems.

What distinguishes real-time systems is that they must be designed to be extremely robust and resistant to faults and downtime (Douglass, 2000). For example, if the operating system on our home computer freezes for any of the common reasons, we simply shut it down and reboot. However, the navigation computers for the space shuttle or other various aircraft must be far more reliable than our home or office computers, as a general rule. Another distinguishing feature of real-time systems is that many real-time systems the hardware and software are both embedded, and are not necessarily easily accessed once they go on line (Douglass, 2000).

Expected Results and Contribution

The research attempts to differentiate practical complexity from theoretical complexity. The argument is that not all constructs in UML are used all the time. In fact, only a small portion of UML constructs forms the kernel of UML. As such, practical complexity is a more realistic and a better reflection of the complexity of UML faced by users in practice.

The results will be useful in developing a set of complexity metrics that are different, more realistic and hopefully, as well as providing valuable information to developers and those constructing UML release 2.0. The research is on going.

References

Anderson, J., and Lebiere, C., 1998, *The Atomic Components of Thought*, Lawrence Erlbaum Associates.

Briand, L., Wüst, J., and Lounis, H., 1999c. "A Comprehensive Investigation of Quality Factors in Object-Oriented Designs: An Industrial Case Study". 21st International Conference on software Engineering, Los Angeles, CA. pp. 345-354.

Conallen, J., 2000, *Building Web-Applications with UML*, Addison-Wesley.

Dobing, B. and Parsons, J., (2000), Understanding the Role of Use Cases in UML: A Review and Research Agenda, *Journal of Database Management,* Vol. 11, No. 4. pp. 28 – 36.

Dori, D., 2002, "Why Significant UML Change is Unlikely", *Communications of the ACM,* Vol. 45, No. 11, November, pp. 82-85.

Douglass, B., 2000, *Real-Time UML Second Edition: Developing Efficient Objects for Embedded Systems*, Addison-Wesley.

Duddy, K., 2002, "UML2 Must Enable a Family of Languages", Communications of the ACM, Vol. 45, No. 11, November, pp. 73-75.

Ericsson, K. and Kintsch, W., 1995, "Long-Term Working Memory", *Psychological Review*, 102(2) 211-245.

Fowler, M. and Highsmith, J., 2001, "The Agile Manifesto", **http://www.sdmagazine.com/print/documnetID=11649**.

Kim, J., Hahn, J., and Hahn, H., 2000, "How Do We Understand a System with (So) Many diagrams? Cognitive Integration Processes in Diagrammatic Reasoning", *Information Systems Research*, Vol. 11, No. 3, September, pp. 284-303.

Kobryn, C., 2002, "What to Expect from UML 2.0", *SD Times,* accessed 10/22-2002.

Marshall, C., 2000, *Enterprise Modeling with UML Designing Successful Software Through Business Analysis*, Addison-Wesley.

McKeown, P., 2001, *Information Technology and the Networked Economy*, Harcourt College Publishers, Orlando, FL.

Mello, Adrian. (2002). ERP Fundamentals, Enterprise, URL: **http://zdnet.search.com/search?cat=279&tag=st.ne.sr.srch.zdnet&q=%22IDC%22**.

Miller, G., 1956, "The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information", *The Psychological Review*, Vol. 63 No. 2, pp. 81-97.

Rossi, M. and Brinkkemper, S., 1996. "Complexity Metrics for Systems Development Methods and Techniques," *Information Systems,* Vol. 21, No. 2 pp. 209-227.

Siau, K., and Cao, Q., 2001, "*Unified Modeling Language (UML) – A Complexity Analysis*", Journal of Database Management, Vol. 12, No. 1, pp. 26-34

Siau, K., Erickson, J., and Lee, L., 2002, "Complexity of UML: Theoretical versus Practical Complexity", Workshop on Information Technology and Systems (WITS). Barcelona, Spain, December 16-18, pp. 13-18.

Sieber, T. Siau, K. Nah, F. Sieber, M. 2000, "Sap Implementation at the University of Nebraska", Journal of Information Technology Cases and Applications, Vol. 2, No. 1, pp. 41-72.

Zendler, A., Pfeiffer, T., Eicks, M., and Lehner, F., 2001, " Experimental Comparison of Coarse Grained Concepts in UML, OML and TOS", *Journal of Systems and Software,* Vol. 57, pp 21-30.