IDEA GROUP PUBLISHING

701 E. Chocolate Avenue, Suite 200, Hershey PA 17033-1240, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.idea-group.com

ITJ2890

# Theoretical vs. Practical Complexity:
## The Case of UML

Keng Siau, University of Nebraska-Lincoln, USA
John Erickson, University of Nebraska at Omaha, USA
LihYunn Lee, University of Nebraska-Lincoln, USA

## ABSTRACT

*Unified Modeling Language (UML) is the standard visual modeling language for Object Oriented (OO) systems development, but it has been criticized for its complexity, inconsistent semantics, and ambiguous constructs. A set of complexity indices for UML and the nine diagramming techniques in UML was compiled recently. The complexity analysis is formulated based on the number of constructs, associations, roles, and so forth, in a modeling method. We argue that this set of metrics provides an indication of the theoretical complexity of the modeling methods. On the other hand, the theoretical complexity of the modeling methods does not necessarily relate to the practical complexity. We hypothesize that UML's complexity is not as daunting as the metrics imply, because not all the constructs are used all the time. Thus, in addition to theoretical complexity, a set of metrics for estimating practical complexity can be developed, based on the most commonly used constructs (instead of all constructs). In this research, we use secondary data to test our hypothesis that practical complexity is different from theoretical complexity.*

*Keywords: cognition; complexity; metrics; unified modeling language*

## INTRODUCTION

Over the past 10 to 15 years, object-oriented systems analysis and design methodology have emerged from object-oriented programming. Over time, separate modeling methods merged to form a combined analysis and design technique called the Unified Modeling Language (UML) (Booch et al., 1999). UML consists of nine distinct diagramming techniques to support object oriented systems development. Despite the standardization of UML by the Object Management Group, researchers and practitioners criticize UML's complexity and the ambiguity of its constructs (Siau & Lee, 2005; Siau & Loo, 2002; Siau et al., 2002). Siau and Cao (2001) used the complexity metrics developed by Rossi and Brinkkemper (1996) to analyze the diagram-

ming techniques in UML and compared them to other modeling methods.

In this research, we argue that the complexity metrics developed by Rossi and Brinkkemper (1996), and adopted by Siau and Cao (2001) (1) present the theoretical complexity of the modeling techniques and (2) present the theoretical upper limit of the complexity, because the metrics were formulated based on the total number of objects, relationships, and property types in the modeling techniques. We define theoretical complexity as the maximum upper limit of the complexity of a modeling technique, taking into account all its objects, relationships, and property types. Practical complexity, on the other hand, takes into consideration that not all modeling constructs will be used simultaneously, and these various constructs have different importance, and, therefore, they should not be weighted equally. Based on these premises, practical complexity must be less than or equal to theoretical complexity.

We aim to extend the complexity metrics to take into account the actual usage/practice of the modeling method. In other words, in addition to the theoretical complexity, we should be able to provide an estimation of the practical complexity of a modeling method, based on the typical usage of the modeling method. The practical complexity of a modeling language will provide a more realistic estimation and will complement the theoretical complexity. UML has been criticized for being overly complex, and the theoretical complexity metrics bear this out; however, if the UML that people learn and use does not make use of all constructs all the time, then perhaps it is not out of line to conjecture that UML, while undoubtedly complex, is not as complex and difficult to learn and use as the metrical analyses completed to date indicate. If so, this research can be used to suggest better ways of learning and using UML.

In this paper, we test our hypothesis that there exists a practical complexity that is different from the theoretical complexity. This is operationalized as the time needed to interpret a set of use-case and class diagrams, testing that the time required for the two diagrams is different from the theoretical estimation. Although there are many ways to measure complexity, we propose that difficulties that subjects encounter during diagram interpretation will manifest in longer interpretation times, and, therefore, time is one means of measuring complexity. The validity of using interpretation times as a surrogate of complexity is discussed in the latter part of the paper.

The rest of the paper is organized as follows. The next section presents the literature of the area, focusing on complexity and UML. This is followed by the Theoretical Foundations section, Research Question and Hypothesis section, Research Methodology and Design section, Results and Discussion section, and, finally, the Conclusion and Future Research section.

## LITERATURE REVIEW

Complexity can take on many forms. 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 modeling approaches such as UML diagrams. In this context, cognitive complexity can be defined as the mental burden people face as they work with systems development constructs.

Further, in addition to the theoretical basis of cognitive complexity detailed in the

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


literature review, 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 humans using the diagrams as aids to understand and/or develop systems (see Figure 1).

Since cognitive complexity, as defined for this research, is difficult and perhaps 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 17 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*.

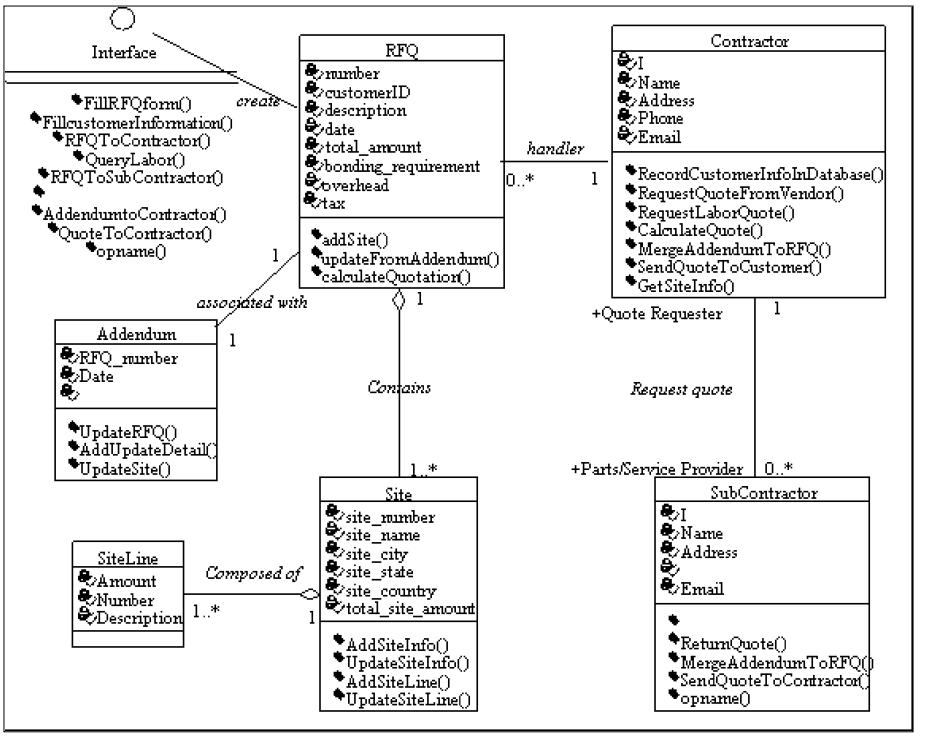
Structural complexity is a part of the structural characteristics of the information or modeling system and, for this research, refers to the elements or constructs that comprise a given diagramming technique. These constructs would include meta-construct types, such as objects (classes, interfaces),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

For example, the class diagram in Figure 2 uses a number of constructs to present the information. There are six classes and one interface. The classes each have a name, as well as attributes and methods (i.e., the RFQ class has eight attributes — number, customerID ... tax; three methods — addSite(), updateFromAddendum(), and calculateQuotation()). In addition, the class has relationships with other classes (contractor, site, addendum) and with the (customer) interface.

A class diagram was chosen as a representative diagram, since class diagrams have a relatively long history of use in both objected-oriented programming and (database) systems development; however, the other UML diagrams can be analyzed in this fashion. Note that even though this (real-world) diagram conveys a relative high volume of information to a person looking at the diagram, it does not use all of the constructs listed as possible (see Appendix A for a comprehensive list of UML constructs), which this research takes to represent theoretical complexity.

The list of UML constructs in Appendix A is taken from Siau and Cao's (2001) results and was developed using Rossi and Brinkkemper's (1996) rubric for estimating the complexity of modeling methods. Rossi and Brinkkemper's approach to defining structural complexity was not developed for any method in particular but rather was intended to be used as a means of leveling the playing field when comparing different modeling methods. The Rossi and

*Figure 2. Class diagram example*




Brinkkemper metrics serve as the operational definition (as well as a measure) of the structural complexity of diagrams, which can be equated with theoretical complexity. The current research further proposes to define practical complexity as a subset of the Rossi and Brinkkemper metrics.

Past MIS research has typically skirted complexity. Kim, Hahn, and Hahn's (2000) study of diagrams and cognitive diagrammatic reasoning enforced interpretation time limits on the subjects in their experiment and, thus, used time only as a constraint. Mcleod and Roeleveld (2002) referred to the relationship between interpretation time and complexity by contending that a person's "ability to learn a concept quickly … depends heavily on the complexity of the model." That implies that there is a relationship between interpretation time and complexity. Vessey and Conger (1994) used time as a measure of interpretation, and Zendler et al. (2001) also used time necessary to structure an application as a construct. However, neither group considered complexity of interpretation directly as part of their research.

Rossi and Brinkkemper's (1996) research proposed and developed a relatively easy-to-use and straightforward means to quantitatively measure system development methods. Specifically, the metrics are based on metamodel techniques and purport to measure the complexity of the method under analysis. According to Rossi and Brinkkemper (1996), complexity is critical to measure, because researchers have shown that complexity is closely related to how easy a specific method is to use and to learn. One of Rossi and Brinkkemper's more crucial caveats is that the measured complexity of a given system does not trans-

late solely into less complex methods being superior to more complex methods.

Siau and Cao's (2001) research applied Rossi and Brinkkemper's (1996) complexity metrics to the UML. Their reasoning for using the particular metric set is that they contend that the metrics are among the most comprehensive and that Rossi and Brinkkemper's approaches "have been used to evaluate the ease-of-use of OO techniques." Siau and Cao (2001) also compared UML's complexity with 36 modeling techniques from 14 methods, as well as each of the 14 methods in aggregate.

One of Siau and Cao's (2001) noteworthy findings is that UML is far more complex (from two to 11 times more complex) in aggregate than any of the other 13 methods. The size-relative overall complexity highlights one of the issues regarding UML. As a result, UML can appear overwhelming to those new to UML. Additionally, when human cognitive limitations to short-term memory are added to this mix, UML can appear even more difficult to master. We compare the relative complexity of use-case and class diagram metrics as the basis of our argument, but it should also be clearly understood that we are not comparing use-case and class diagrams, which are different diagramming techniques and are used for different purposes (e.g., static and dynamic properties of the systems under scrutiny). Rather, we propose that the time needed to interpret use-case and class diagrams can be used as a means of demonstrating that theoretical complexity is different from practical complexity.

Siau and Cao's (2001) UML complexity analysis results for each diagram type are shown in Table 1.

*Table 1.*

| **Technique/Diagram** | **n ($O_T$)** | **n ($R_T$)** | **N ($P_T$)** | $\overline{P}_O$ **($M_T$)** | $\overline{P}_R$ **($M_T$)** | $\overline{R}_O$ **($M_T$)** | $\overline{C}$ **($M_T$)** | **C`($M_T$)** |
|---|---|---|---|---|---|---|---|---|
| Class | 7 | 18 | 18 | 1.71 | 1.22 | 2.57 | 0.1 | **26.40** |
| Use Case | 6 | 6 | 6 | 1 | 0.83 | 1 | 0.17 | **10.39** |
| Activity | 8 | 5 | 6 | 0.75 | 0.2 | 0.63 | 0.13 | 11.18 |
| Sequence | 6 | 1 | 5 | 0.67 | 6 | 0.17 | 0.13 | 7.87 |
| Collaboration | 4 | 1 | 7 | 1 | 8 | 0.25 | 0.14 | 8.12 |
| Object | 3 | 1 | 5 | 1.67 | 3 | 0.33 | 0.33 | 5.92 |
| StateChart | 10 | 4 | 11 | 1 | 0.5 | 0.40 | 0.09 | 15.39 |
| Component | 8 | 10 | 9 | 1 | 3.6 | 1.25 | 0.11 | 15.65 |
| Deployment | 5 | 7 | 5 | 1 | 1.14 | 1.40 | 0.2 | 9.95 |

*Legend*

n ($O_T$): count of object types per technique.

n ($R_T$): count of relationship types per technique.

n ($P_T$): count of property types per technique.

$\overline{P}_O$ ($M_T$): average number of properties for a given object type.

$\overline{P}_R$ ($M_T$): average number of properties per relationship type.

$\overline{R}_O$($M_T$): number of relationship types that can be connected to a certain object type.

$\overline{C}$ ($M_T$): average complexity for the entire technique.

C′($M_T$): total conceptual complexity of the technique.

## THEORETICAL FOUNDATION

Throughout the analysis and development process, we must also concentrate some of our thought and attention on the idea of cognitive limits. Roughly speaking, as humans, we can hold in our short-term working memory (STWM) a total of between five and nine items simultaneously. Miller (1956) spent a great deal of effort toward establishing those cognitive limits, and his findings have become accepted norms on the subject.

Domain experts in systems development must depend on their own automaticized expert knowledge of systems analysis and development processes in order to fully comprehend the functionalities that the users tell them they must have in a system. Users usually are neither domain experts nor technically proficient and, thus, in general, do not have a clear technical understanding of what information systems really do or how they do what they do. Therefore, the systems analysts, as domain experts, must be able to operate on two levels simultaneously as they interview users regarding system requirements. That is, the analysts must be able to listen to the (non-technical) requirements as presented by users and simultaneously develop an understanding of the underlying technical requirements of each proposed system element, as well as process that information into forms that the user can understand.

Ericsson and Kintsch (1995) have proposed an extension to working memory, which they call long-term working memory (LTWM). According to Ericsson and Kintsch, LTWM is a part of long-term memory (LTM) that acts almost like STWM. In this model, domain experts can recall very quickly the information that is relevant to their area of expertise. The relevant information recall is a bit slower than STWM recall speeds but appears to be significantly faster than LTM recall speeds. Of course, LTWM only works for experts in their domain areas or possibly others, such as experienced users, who have high automaticity for certain tasks. LTWM would help to explain why it is important to interview users from different areas of organizations as part of the analysis process. Users, especially experienced ones, will likely have highly valuable information regarding existing systems processes (i.e., how things are done here) that it is critical for the analysts to capture during the interview process.

LTWM is an important concept to consider when thinking about human cognitive limitations regarding information processing and especially as related to task complexity. What the idea of LTWM conveys is that automaticity can greatly extend the short-term memory limitations of domain experts. LTWM addresses the problem of the information processing bottleneck and suggests that, as people learn even very complex materials, they can somewhat overcome some of their own cognitive limitations. This also implies that a person can learn a very complex methodology, such as UML, more easily by assimilating a small or relatively small part of the overall methodology by using LTWM.

J. R. Anderson is a cognitive psychology researcher and theorist whose models have often been used in MIS and HCI research. Anderson's theory has undergone several revisions, the latest being the one used as a basis for this research (1976, 1983, 1993, 1998). Anderson's Atomic Components of Thought (ACT) attempts to divide cognition into its component parts.

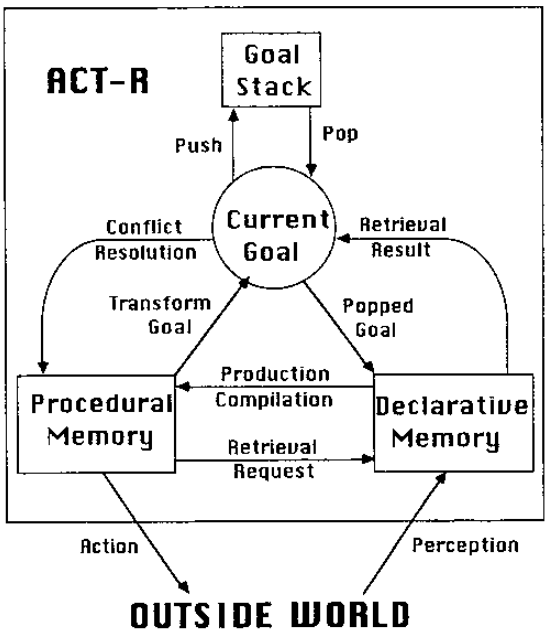
ACT breaks knowledge into two parts — declarative knowledge and procedural knowledge. Declarative knowledge is similar to knowledge captured in an encyclo-

pedia 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). In short, ACT tells us that the more complex a problem is, the longer it will take us to process the problem and come up with a solution (see Figure 3).

Drawing once again on the systems analysis domain, we can see how critical it is that those who develop the modeling techniques and methods do not make them too complex. The reasoning works as follows. As domain experts, the methods and techniques developers have their expertise to draw upon and will likely have participated in many systems development efforts and have years of experience. So, these experts have at their disposal highly developed and specialized stores of domain knowledge and will be able to quickly recall from LTWM far more of that domain knowledge than the typical end user likely knows about systems development methods and techniques. That means that what may be glaringly apparent to the system modeling methods and techniques developers may be quite complex and simply beyond the processing capacity of typical end-users, since systems development methods and techniques are not their domain of expertise.

This highlights the importance of studying systems development methods and techniques complexity. It also stresses the importance of studying systems development methods and techniques complexity,

*Figure 3. ACT-R model*



not only from the viewpoint of modeling techniques developers, but, more importantly, from the perspectives of end users.

In the course of processing productions, a bottleneck develops between what information is waiting in a queue for processing and what is currently being processed. Thus, if one comprehension task is simpler relative to another comprehension task, it will consume fewer resources (processing time, long-term production activation and retrieval, and short-term memory) and, thus, take less overall time to process. As such, time can be one of the possible surrogates for estimating complexity.

## RESEARCH QUESTION AND HYPOTHESIS

If theoretical complexity predicts complexity encountered by subjects in practice (i.e., practical complexity), we would expect to observe roughly the same degree of differences between use-case and class diagrams in empirical studies as those determined by the complexity indices provided by Siau and Cao (2001). For example, Siau and Cao's results indicate that class diagrams are roughly 2.57 times more complex than use-case diagrams. In other words, if theoretical and practical complexity measure the same construct, as subjects interpret the two diagrams in an experimental context, we would expect to find that class diagrams take roughly 2.57 times longer than use-case diagrams to interpret. However, we argue that practical complexity is less than theoretical complexity. Therefore, the hypothesis:

*Practical complexity is less than theoretical complexity.*

If class diagrams are, indeed, much more complex than use-case diagrams, then it should be possible to empirically test using a time construct as a relative measure of complexity, since ACT-R tells us that more complex tasks take longer to accomplish. Therefore, it should follow that if class diagrams are, indeed, more complex than use-case diagrams, then it should take longer for study subjects to interpret class diagrams than to interpret use-case diagrams.

## RESEARCH METHODOLOGY AND DESIGN

The data is secondary from an experiment conducted by a graduate student on volunteer subjects who had completed a one-semester UML course at a large US Midwestern university. Class and use-case diagrams from two different problem domains, a bank, and a music club, were the objects of interpretation. One of the problem domains, the music club, is more complex than the other (see Diagrams 1 to 4 in Appendix B). Each subject was given, at random, either a class diagram or a use-case diagram from one of the two systems and was asked to think aloud while they interpreted the diagram. Once finished with the diagram, the subjects were given the complementary diagram (i.e., a use-case diagram on the same domain, if the subject was given a class diagram previously, and vice versa) and were asked to continue the interpretation with both diagrams. Then, the process was completed for the second system (i.e., a repeated design) (see Table 2). The entire session was audiotaped. Thirty-one subjects participated in the study, but times for two of the subjects were unobtainable because of problems with the audiotapes, leaving a usable sample size of 29.

The tapes were not timed as part of the original experiment. For this research, interpretation times were obtained simply by timing the taped interpretation of the first diagram for each problem domain. Only the

*Table 2.*

| TREATMENT | DOMAIN | |
|---|---|---|
| | **BANK – ATM** | **MUSIC CLUB** |
| **1** | ***CD,** then CD and UC | ***UC,** then UC and CD |
| **2** | ***UC,** then UC and CD | ***CD,** then CD and UC |
| **Original Research Design (CD=Class Diagram; UC = Use-Case Diagram) *Indicates the data used for this research** | | |

first interpretation for each problem domain was used (as indicated by * in Table 2), because the second part of the experiment involved using both use-case and class diagrams simultaneously. Time was measured to the nearest second.

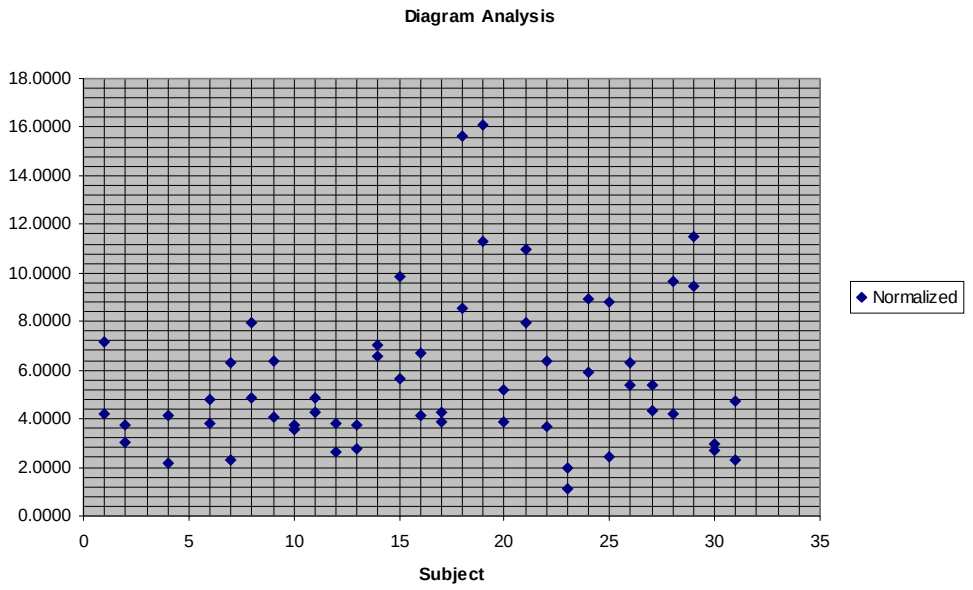
## RESULTS AND DISCUSSION

For analysis purposes, the time data was normalized to remove the effects of the different system complexity discussed previously. This was accomplished by dividing the time by the number of possible constructs in the respective diagrams, resulting in a measure of the analysis time per construct and making an apples-to-apples comparison of the different diagram types realistic and possible.

Once the data was normalized, it appeared that there was two outlier points (more than three standard deviations form the mean), which could have exerted undue influence on the final results; these points were removed from the data set for the statistical analysis (see Figure 4).

Figure 5 presents the descriptive statistics for the data set, where group 1 indicates class diagrams, and group 2 indicates use-case diagrams. Figure 6 shows the results of the homogeneity of variances test between the two groups. Generally, the omnibus F-test is relatively robust to dif-

*Figure 4. Scatter plot*



ferences in group variances, if the groups are the same size. In this case, the group sizes are different, and so the Levene's results are arguably more important to consider. However, the results are non-significant at the 0.05 level, which means that we may have some confidence in the F-test.

The one-way analysis of variance conducted indicates a non-significant result for the interpretation times for the two diagram types (see ANOVA results in Figure 7). Since there are just two groups, the one-way ANOVA produces the same results as a t-test.

Thus, we reject the null hypothesis and conclude that practical complexity is less than theoretical complexity. This also indicates that theoretical complexity might not be an accurate measure of practical complexity, since there is no significant difference between use-case and class diagrams in our empirical study, but there are substantial differences between use-case and class diagrams in the complexity indices. This suggests that a different set of metrics for practical complexity might be warranted.

There are several possible reasons for the inadequacy of theoretical complexity in estimating practical complexity. First, although Siau and Cao's (2001) complexity indices indicated that class diagrams are about 2.57 times more complex than use-case diagrams, the analysis was based on all objects, relationships, and property types, as formulated by Rossi and Brinkkemper (1996). In practice, not all constructs in each diagram are used all the time. For example, the class diagram can contain many relationship types (e.g., association, aggregation, composition, generalization, dependency) and objects (e.g., abstract class, notes, constraints, packages, subsystems, interface), but a typical class diagram only uses a subset of these. As such, theoretical complexity may not be the best measure of the complexity that a user will encounter in practice.

Second, one of the typical reasons for complexity is the limitation of short-term memory (Siau, 2004). Miller (1956) argued that the bottleneck in human cognition is the limitation of seven plus or minus two chunks of information in short-term memory. Although short-term memory is a concern, decomposing a complex problem into subproblems can alleviate the limitation (Siau, 1999; Siau et al., 1997). For example, when the subjects were asked to understand the class diagram, which can have many object, relationship, and property types, the subjects would not attempt to understand every element in the class diagram simultaneously. A subject would decompose the diagram into manageable subdiagrams and would understand each subdiagram, in turn. In this case, the short-term memory limitation is partially overcome. Current complexity metrics do not take this into account. The ability for us to decompose complex problems into subproblems needs to be factored into complexity metrics formulation.

Third, we argue that there is a need to assign weights to different constructs. For example, a construct that is more likely to result in a short-term memory problem should be weighted more in the complexity metrics than one that is less likely to result in a short-term memory constraint. With respect to UML, for example, we would argue that objects are less likely, when compared to relationships, to result in short-term memory constraint, as they are more or less independent. Relationships, on the other hand, need to be interpreted with associated objects in order to make sense. Hence, one relationship will consume more short-term memory resources than an object.

*Figure 5. Descriptives*

Normalized

| | N | Mean | Std. Deviation | Std. Error | 95% Confidence Interval for Mean | | Minimum | Maximum |
|---|---|---|---|---|---|---|---|---|
| | | | | | Lower Bound | Upper Bound | | |
| 1 | 27 | 5.4269963 | 3.4474058 | .6634536 | 4.0632480 | 6.7907446 | 1.97440 | 16.08700 |
| 2 | 26 | 5.7584769 | 3.1405098 | .6159046 | 4.4899976 | 7.0269563 | 1.08700 | 11.51280 |
| Total | 53 | 5.5896094 | 3.2729227 | .4495705 | 4.6874802 | 6.4917387 | 1.08700 | 16.08700 |

*Figure 6. Test of homogeneity of variances*

Normalized

| Levene Statistic | df1 | df2 | Sig. |
|---|---|---|---|
| .542 | 1 | 51 | .465 |

*Figure 7. ANOVA results*

Normalized

| | Sum of Squares | df | Mean Square | F | Sig. |
|---|---|---|---|---|---|
| Between Groups | 1.455 | 1 | 1.455 | .134 | .716 |
| Within Groups | 555.570 | 51 | 10.894 | | |
| Total | 557.025 | 52 | | | |

Finally, and perhaps most importantly, we also may argue from an 80/20 perspective that practical complexity is more relevant to systems development than theoretical complexity. In essence, the 80/20 rule of thumb (Kobryn, 2002) says that 80% of common software solutions (i.e., software development projects) usually can be completely specified by using only 20% of the language constructs. If that is true, then only the most commonly used constructs constitute the majority of software development efforts, and that 20% of the language therefore should define practical complexity. Moreover, if many of the constructs are rarely or never used, it would not be necessary to learn the complete syntax of the language in order to develop the majority of systems. Of course, field validation of these contentions beyond an experimental setting and developing more accurate metrics will be necessary, if this is to provide value to researchers, to those revising the UML specification, and to software development experts, in general.

## CONCLUSION AND FUTURE RESEARCH

Our results indicate that theoretical complexity might not accurately predict practical complexity. The long-term objective of this research is to propose a set of metrics to estimate practical complexity of modeling methods. Although formulating practical complexity is difficult, and practical complexity depends on many circumstances (e.g., project domains, structured/semi-structured/unstructured), coming up with an estimation of practical complexity is possible and useful. For example, the Function Point Analysis (Albrecht, 1983), Constructive Cost Model (Boehm, 1981), and Activity-Based Costing Model (Cooper & Kaplan 1988) are illustrations of the usefulness of estimation, even rough estimation.

In addition, UML 2.0 is ready for release, and identifying a "kernel" of the language already has been identified as a goal by a number of those involved in developing the new version. These results also can provide a possible future direction for researchers studying complexity in development methodologies, since theoretical or total complexity would become a less prominent factor (although still extremely important in determining limits) in dealing with human cognitive limitations involving systems development.

## REFERENCES


Albrecht, A.J., & Gaffney, J.E. (1983). Software function, source line of codes, and development effort prediction: A software science validation. *IEEE Transaction on Software Engineering, 19*, 639-648.

Anderson, J. (1976). *Language, memory and thought*. Lawrence Erlbaum Associates.

Anderson, J. (1983). *The architecture of cognition*. Harvard University Press.

Anderson, J. (1993). *Rules of the mind*. Lawrence Erlbaum Associates.

Anderson, J., & Lebiere, C. (1998). *The atomic components of thought*. Lawrence Erlbaum Associates.

Boehm, B.W. (1981). *Software engineering economics*. Englewood Cliffs, NJ: Prentice-Hall.

Booch, G., Jacobson, I., & Rumbaugh, J. (1999). *The unified modeling language user guide*. Addison-Wesley.

Briand, L., Wüst, J., & Lounis, H. (1999). A comprehensive investigation of quality factors in object-oriented designs: An industrial case study. *Proceedings of the Twenty-First International Conference on Software Engineering*. Los Angeles, California.

Cooper, R., & Kaplan, P.S. (1988). Measure costs right: Make the right decisions. *Harvard Business Review*. 96-103.

Dobing, B., & Parsons, J. (2000). Understanding the role of use cases in UML: A review and research agenda. *Journal of Database Management, 11*(4), 28-36.

Ericsson, K., & Kintsch, W. (1995). Long-term working memory. *Psychological Review, 102*(2), 211-245.

Kim, J., Hahn, J., & Hahn, H. (2000). How do we understand a system with (so) many diagrams? Cognitive processes in diagrammatic reasoning. *Information Systems Research, 11*, 384-303.

Kobryn, C. (2002). Will UML 2.0 be agile or awkward? *Communications of the ACM, 45*(1), 107-110.

Mcleod, G., & Roeleveld, D. (2002). Method evaluation in practice: UML/RUP vs. inspired method. *Proceedings of the International Workshop on Evaluation of Modeling Methods in Systems Analysis and Design*.

Miller, G. (1956). The magical number seven, plus or minus two: Some limits on our capacity for processing information. *The Psychological Review, 63(*2), 81-97.

Rossi, M., & Brinkkemper, S. (1996). Complexity metrics for systems development methods and techniques. *Information Systems, 21*(2), 209-227.

Rumbaugh, J., Jacobson, I., & Booch, G. (1999). *The unified modeling language reference manual*. Addison-Wesley.

Siau, K. (1999). Information modeling and method engineering: A psychological perspective. *Journal of Database Management, 10*(4), 44-50.

Siau, K. (2004). Informational and computational equivalence in comparing information modeling methods. *Journal of Database Management, 15*(1), 73-86.

Siau, K., & Cao, Q. (2001). Unified modeling language (UML) — A complexity analysis. *Journal of Database Management, 12*(1), 26-34.

Siau, K., & Lee, L. (2004). Are use case and class diagrams complementary in requirements analysis? An experimental study on use case and class diagrams in UML. *Requirements Engineering***,** 9(4), 229-237.

Siau, K., & Loo, P. (2002). Difficulties in learning UML: A concept mapping analysis. *Proceedings of the Seventh CAiSE/ IFIP8.1 International Workshop on Evaluation of Modeling Methods in Systems Analysis and Design*. Toronto, Canada, (pp. 102-108).

Siau, K., Erickson, J., & Lee, L. (2002). Complexity of UML: Theoretical versus practical complexity. *Proceedings of the Twelfth Workshop on Information Technology and Systems*. Barcelona, Spain, (pp. 13-18).

Siau, K., Wand, Y., & Benbasat, I. (1997). The relative importance of structural constraints and surface semantics in information modeling. *Information Systems, 22*(2, 3), 155-170.

Vessey, I., & Conger, S. (1994). Requirements specification: Learning object, process, and data methodologies. *Communications of the ACM, 37*(5),102-113.

Zendler, A., Pfeiffer, T., Eicks, M., & Lehner, F. (2001). Experimental comparison of coarse-grained concepts in UML, OML and TOS. *Journal of Systems and Software, 57*(1), 21-30.

*Keng Siau is a full professor of management information systems at the University of Nebraska, Lincoln (UNL). He is currently serving as the editor-in-chief of the* Journal of Database Management *and as the book series editor for* Advanced Topics in Database Research*. He received his PhD from the University of British Columbia (UBC), where he majored in MIS and minored in cognitive psychology. His master's and bachelor's degrees are in computer and information sciences (National University of Singapore). Dr. Siau has more than 200 academic publications. He has published more than 70 refereed journal articles, appearing (or forthcoming) in journals such as* Management Information Systems Quarterly, Communications of the ACM, IEEE Computer, Information Systems, IEEE Transactions on Systems, Man, and Cybernetics, IEEE Transactions on Professional Communication, IEEE Transactions on Information Technology in Biomedicine, IEICE Transactions on Information and Systems, Data and Knowledge Engineering, International Journal of Human-Computer Studies, Behaviour and Information Technology, Quarterly Journal of Electronic Commerce*, and others. In addition, he has published more than 80 refereed conference papers, edited/co-edited 10 scholarly and research-oriented books, edited/co-edited nine proceedings, and written more than 15 scholarly book chapters. He served as organizing and program chairs of the International Workshop on Evaluation of Modeling Methods in Systems Analysis and Design (EMMSAD) (1996-2005).*

*John Erickson is an assistant professor at the University of Nebraska at Omaha. His current research interests include study of unified modeling language as an object-oriented systems development tool, and software engineering and their application to minimizing the user-designer communication gap, including research into problems with ERP implementations relating to the user-designer gap.*

*LihYunn Lee was a graduate student at the University of Nebraska-Lincoln. Her master's thesis focused on the informational values of use case diagram and class diagram in UML. She is an information systems administrator for an ERP project at a high technology company in Rockville, MD.*

# APPENDIX A: UML CORE CONCEPTS AND CONSTRUCTS

| Class Diagrams | | | |
|---|---|---|---|
| **Object Types** | **Property Types** | **Relationship Types** | **Role Types** |
| Class | Class name | Association | Association |
| Interface | Class attribute | Aggregation | Aggregation |
| Package | Class operation | Composition | Composition |
| Subsystem | Association end | Generalization | Generalization |
| Object | Association name | Single inheritance | Single inheritance |
| Note | Association constraint | Multiple inheritance | Multiple inheritance |
| Constraint | Association qualifier | Qualified association | Qualified |
| Tagged value | Aggregate end | Instantiate | Instantiate object |
| Stereotype | Generalization constraint | Dependency realization | Dependency part |
| | Interface name | Dependency usage | Realization part |
| | Interface attribute | | Usage part |
| | Interface operation | | |
| | Package name | | |
| | Subsystem name | | |
| | Object name | | |
| | Object attribute | | |
| | Note comment | | |
| | Constraint string | | |
| | Stereotype string | | |
| | Tagged value string | | |

| Use-Case Diagrams | | | |
|---|---|---|---|
| **Object Types** | **Property Types** | **Relationship Types** | **Role Types** |
| Actor | Actor name | Association | Association |
| Use case | Use-case name | Extend | Generalization |
| Package | Package name | Generalization | Base use case |
| Subsystem | Subsystem name | Dependency | Extended use case |
| Note | Note comment | Uses | Included use case |
| Constraint | Constraint string | Include | |
| Tagged value | Stereotype string | | |
| Stereotype | Tagged value string | | |

| Activity Diagrams | | | |
|---|---|---|---|
| **Object Types** | **Property Types** | **Relationship Types** | **Role Types** |
| Activity state | Activity name | Transition | Guard condition name |
| Action state | Action name | Branch | |
| Object | Object name | Fork | |
| Swimlane | Swimlane name | Join | |
| Initial state | Note comment | Object flow | |
| Stop state | Constraint string | | |
| Note | Stereotype string | | |
| Constraint | Tagged value string | | |
| Tagged value | | | |
| Stereotype | | | |

| Sequence Diagrams | | | |
|---|---|---|---|
| **Object Types** | **Property Types** | **Relationship Types** | **Role Types** |
| Actor | Message name | Message | Create |
| Object | Actor name | | Destroy |
| Lifeline | Object name | | Call |
| Focus of control | Note comment | | Return |
| Note | Constraint string | | Send |
| Constraint | Stereotype string | | |
| Tagged value | Tagged value string | | |
| Stereotype | | | |

# APPENDIX A: UML CORE CONCEPTS AND CONSTRUCTS (CONT.)

| Collaboration Diagrams | | | |
|---|---|---|---|
| **Object Types** | **Property Types** | **Relationship Types** | **Role Types** |
| Object | Object name | Link | Local |
| Actor | Sequence number | | Global |
| Note | Message name | | Transient |
| Constraint | Actor name | | Create |
| Tagged value | Path stereotype | | Destroy |
| Stereotype | Note comment | | |
| | Constraint string | | |
| | Stereotype string | | |
| | Tagged value string | | |

| Object Diagrams | | | |
|---|---|---|---|
| **Object Types** | **Property Types** | **Relationship Types** | **Role Types** |
| Object | Object name | Link | Local |
| Note | Class name | | Global |
| Constraint | Object attribute | | Transient |
| Tagged value | Note comment | | |
| Stereotype | Constraint string | | |
| | Stereotype string | | |
| | Tagged value string | | |

| Statechart Diagrams | | | |
|---|---|---|---|
| **Object Types** | **Property Types** | **Relationship Types** | **Role Types** |
| State | Event name | Transition | Guard condition |
| Substate | State name | Branch | |
| Event | Action name | Fork | |
| Action state | Activity name | Join | |
| Activity state | Event trigger | | |
| Initial state | Entry action | | |
| Final state | Exit action | | |
| Object | Internal transition | | |
| Note | Deferred event | | |
| Constraint | Note comment | | |
| Tagged value | Constraint string | | |
| Stereotype | Stereotype string | | |
| | Tagged value string | | |

| Component Diagrams | | | |
|---|---|---|---|
| **Object Types** | **Property Types** | **Relationship Types** | **Role Types** |
| Deployment component | Component name | Generalization single inheritance | Generalization part |
| Work product component | Path name | Generalization multiple inheritance | Single inheritance part |
| Execution component | Package name | Association | Multiple inheritance part |
| Interface | Subsystem name | Aggregation | Association |
| Package | Interface name | Composition | Aggregation |
| Subsystem | Interface operation | Realization | Composition |
| Note | Interface attribute | Dependency | Dependency part |
| Constraint | Note comment | Dependency-usage | Usage part |
| Tagged value | Constraint string | | Imports |
| Stereotype | Stereotype string | | Exports |
| | Tagged value string | | |

# APPENDIX A: UML CORE CONCEPTS AND CONSTRUCTS (CONT.)

| Deployment Diagrams | | | |
|---|---|---|---|
| **Object Types** | **Property Types** | **Relationship Types** | **Role Types** |
| Processor | Processor name | Association | Association |
| Distribution unit | Distribution unit name | Aggregation | Aggregation |
| Device | Device name | Composition | Composition |
| Note | Note comment | Dependency realization | Dependency part |
| Constraint | Constraint string | Dependency usage | Realization part |
| Tagged value | Stereotype string | | Usage part |
| Stereotype | Tagged value string | | |

# APPENDIX B: DIAGRAMS

*Diagram B(1). Class diagram for a bank problem domain*

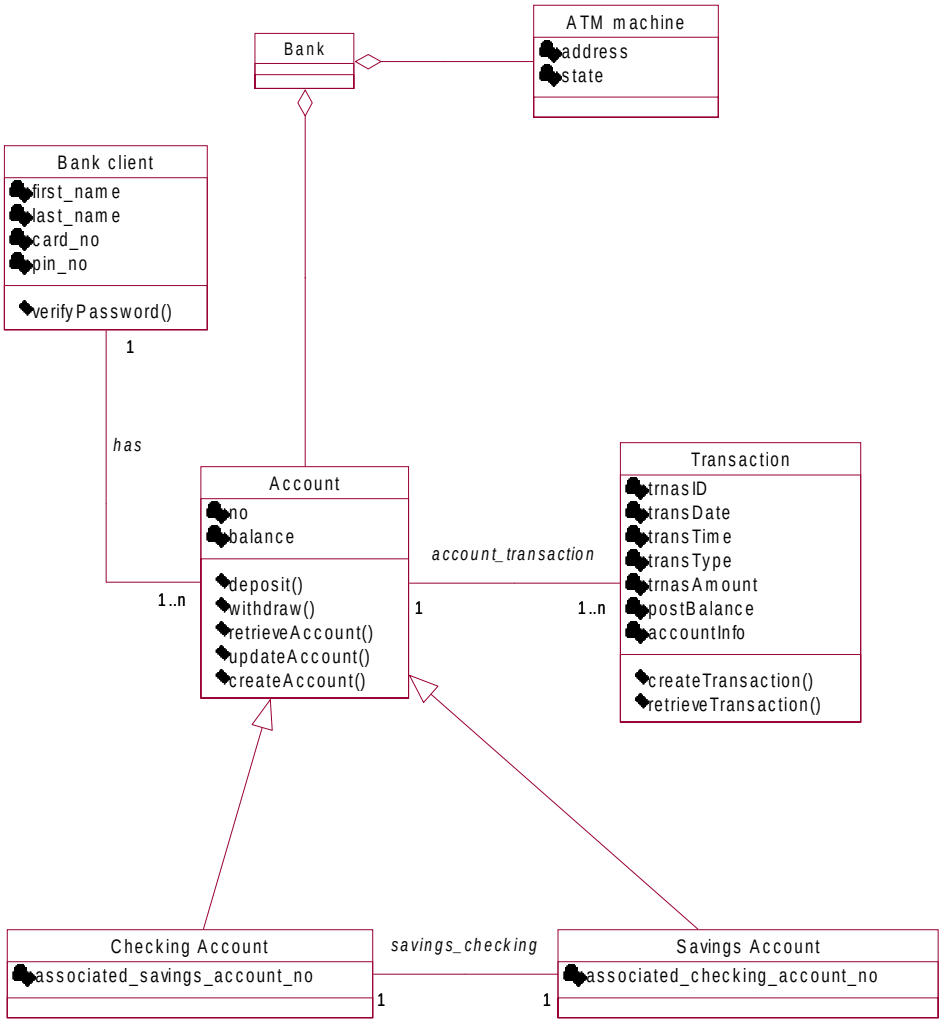


*Diagram B(2). Class Diagram for a Music Club Problem Domain*

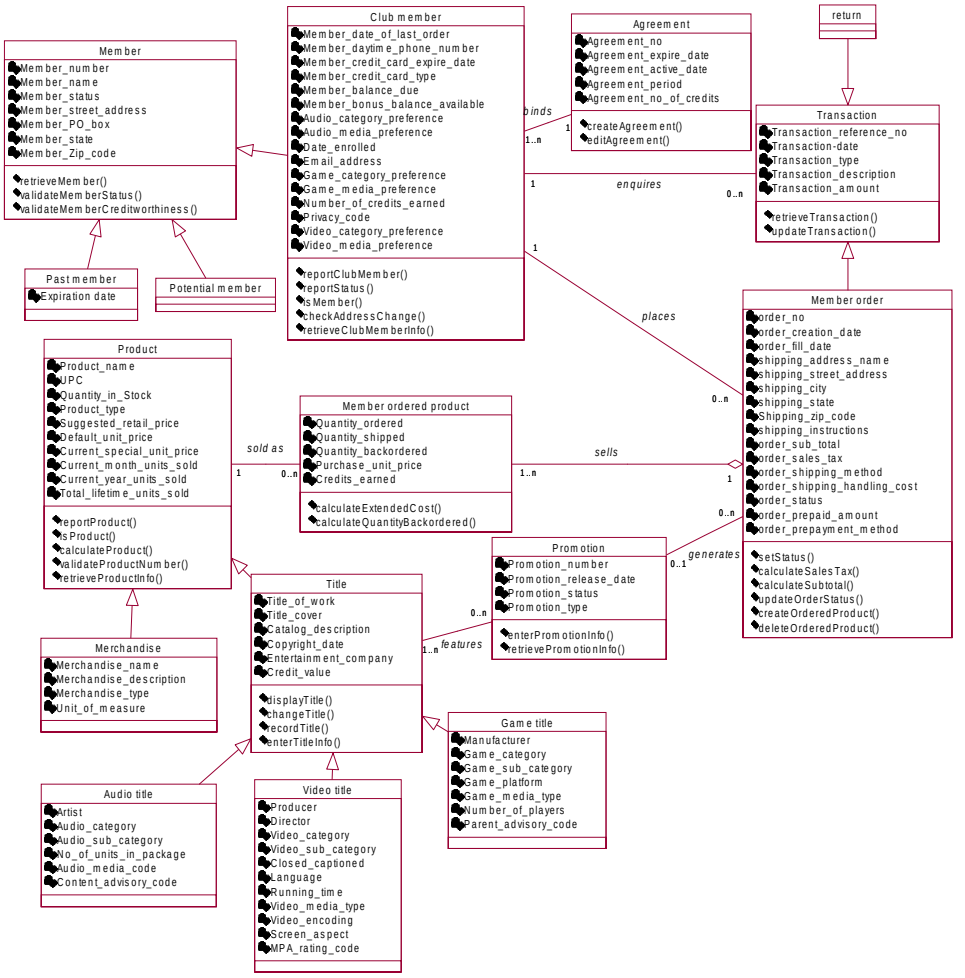


*Diagram B(3). Use-Case Diagram for a Bank Problem Domain*

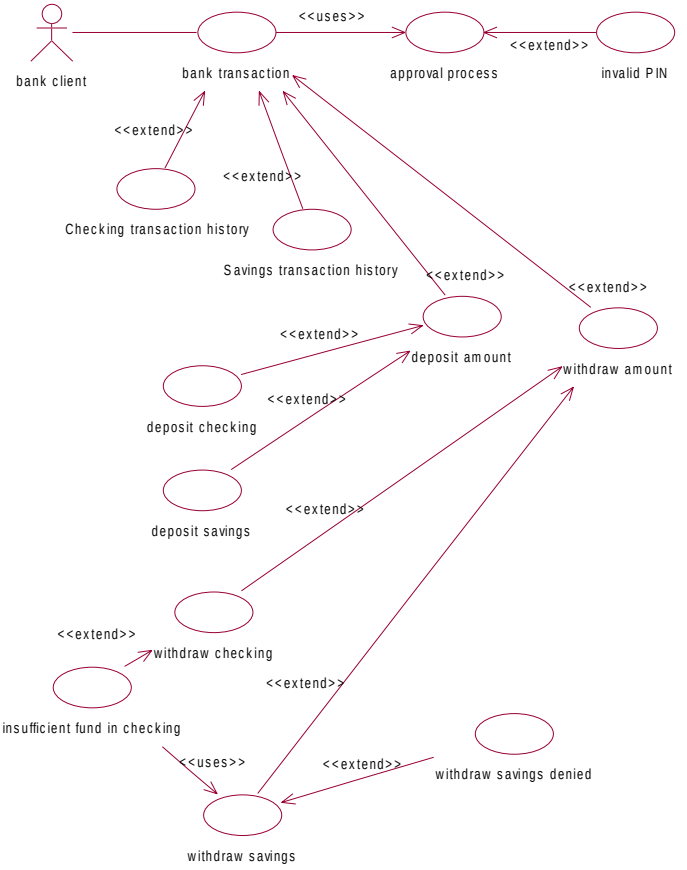



*Diagram B(4). Use-Case Diagram for a Music Club Problem Domain*