Can UML Be Simplified?
Practitioner Use of UML in Separate Domains

John Erickson
University of Nebraska at Omaha
Omaha Nebraska USA
johnerickson@mail.unomaha.edu

Keng Siau
University of Nebraska-Lincoln
Lincoln Nebraska USA
ksiau@unl.edu

Abstract

UML's complexity is regularly criticized by practitioners and researchers alike, who argue that such complexity is a considerable detriment to the adoption and use of UML in the field. Attempts have been made to assess and/or measure UML's complexity in a number of ways. Erickson and Siau proposed that a subset (kernel) of UML, composed of the most important constructs, could be equated with the complexity that practitioners face when using the modeling language. This research extends Erickson and Siau's work by proposing a UML kernel in three application areas, real-time, web-based and enterprise systems. Compared to other modeling methods and languages, UML is very complex. As such, identifying a UML kernel will help in the training and usage of the language. In this research, we conduct a Delphi study using UML experts, to identify three UML kernels, and a non-specific kernel, which are then combined into a single kernel.

Key words: UML, complexity, complexity metrics, Delphi study, Real-Time systems, Web-based systems, Enterprise systems.

1.0 Introduction

Many people insist that the world we live in becomes more complex every day. So-called black-box systems, based on Web-Services, Service Oriented Architectures, or Application Service Providers are currently hailed as the wave of the future for many organizations. Essentially, this means that organizations are expected to use these technologies without understanding them because, among other reasons, the technical details and construction of such systems are too complex for users to understand. In addition to added complexity, in almost all cases, we also expect our systems to respond in real time to meet our needs.

In an environment in which systems and software have become so complex that developers have no expectation of a deep understanding by the users, then it is not out of line to suppose that the underlying systems development process has become more complex and perhaps arguably less understandable as well. Further, we propose that if it is reasonable to assume that if the development process in general mirrors the progression to greater complexity, then it is also logical that specific development methods have become concurrently more complex. An example is UML 2.X. While the original UML 1.X was roundly criticized for its complexity, inconsistent semantics, and ambiguity [7, 9, 10, 12, 18, and 30], the early version also lacked truly useful extension mechanisms that would facilitate its use in a variety of domains. All of this leads to a question of why. Why do UML's "owners" not remediate the problem with their metamodel, and remove or minimize the

inconsistencies? The question is of course rhetorical and we do not attempt to answer it herein, but it is an issue that should be considered by the OMG [23] which controls the UML metamodel. While UML 2.0 is advertised as addressing some of these issues, it seems fairly likely that any increase in extensibility comes at the expense of increased complexity, since UML 2.0 includes, among other advertised and extension-related improvements, four new diagram types along with their related constructs.

The impact of increasing complexity upon the people using UML presents another issue on a more human level. As finite beings, people often have problems processing and understanding information that is extremely complex. Further, increases in complexity are usually accompanied by increases in processing problems as well as decreases in comprehension. Domain experts and other specialists are more able to develop successful coping strategies that allow them to minimize the cognitive load problems inherent in processing complex information by various means, including increased automaticity, production creation, retrieval and usage, spreading (neural) activation, and extensions to working memory [1, 2, 3, 16, 17, 22, and 23]. This problem surfaces often as people build information systems, which tend to be increasingly complex. 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. Since typical system (business) users can be assumed to be less adept at using UML than experts, and since models are often a primary means of communication between developers and users, the cognitive load UML presents to those users must be higher than that for the experts. Further, if the experts do not fully utilize UML, then the part(s) of UML that they DO use should consist of those that they consider to be most important.

This research is aimed at developing a means to deal with UML's complexity for practitioners and developers alike. Identifying a kernel for UML will help practitioners and developers to focus on the key constructs and diagrams in UML, and will facilitate training and educational effort. Additionally, this research could be seen as a call to simplify, if possible, the UML metamodel. While that may not be an entirely reasonable goal given the state of nature regarding IBM and the OMG, we present three separate but closely related practitioner-based UML kernels as the results of a Delphi study, and an overall UML kernel, to help clarify how people and organizations are actually using UML in the field. The results also have implications for many organization and people involved with, or contemplating use of UML including researchers, developers, users, and educators.

2.0 Related Work

2.1 Structural Complexity Metrics

In 1996 Rossi and Brinkkemper [25] proposed and developed a relatively easy to use and straightforward set of measures intended to capture the structural complexity of modeling methods. Their metrics are based on measurement of the metamodel constructs, and were specifically created to be measure the complexity of virtually any (diagrammatic, structurally based) modeling method. Rossi and Brinkkemper [25] further proposed that complexity assessment was important because the complexity of a particular modeling method was long supposed to be closely related to how easy that method was to learn and use. While no claim is made here regarding the efficacy

of the measures themselves, the important point is the connection between complexity and ease of use and ease of learning.

In 2001 Siau and Cao [26] applied Rossi and Brinkkemper's complexity metrics to UML (1.X), and compared its complexity with 36 OO diagramming techniques from 14 distinct methods, as well assessing the overall complexity of each of the 14 methods in aggregate. Unsurprisingly, Siau and Cao [26] found that UML is far more complex (from 2 to 11 times more complex) in aggregate than any of the other 13 methods assessed. The very high overall complexity of UML relative to other methods highlights one of the issues regarding the language, that it can appear intimidating or overwhelming to those new to UML, because of human cognitive load limitations.

2.2 Cognitive Processing Models

Cognitive psychologists have long had a goal of create an understandable and accurate model of how humans process information, simple or complex. Research over at least the past 50 years typifies the effort regarding human cognition and comprehension [1, 2, 3, 14, 22, 23, 28, and 29].

In 1956 Miller [23] proposed a model detailing some possible reasons for the problems humans have processing cognitively complex information. His experiment determined rough limits for what he called Immediate Memory (now more commonly known as Short Term Memory – STM). Miller's 1956 research aimed more at the storage capacity of STM as opposed to the almost unlimited storage capacity of Long Term Memory (LTM). Baddeley's research [2 and 3] took Miller's work and incorporated other objects to the STM model, the Central Executive, the Visualspatial Sketchpad, and the Phonological Loop to the storage component. Baddeley's 1992 model of Working Memory (WM) [2] is still in use today. Many other cognitive processing models exist, but are not cited here for brevity.

2.3 Cognitive Load

Complexity can obviously take many forms. This research examines only 2 of those; structural complexity, and cognitive complexity. Cognitive complexity as related to human perception can be seen as the burden (load) people face in trying to process and understand models of information systems. Structural complexity is more closely connected to the physical properties of the diagramming techniques found in modeling approaches such as UML diagrams.

Cognitive Load Theory assumes that novices have little pre-existing knowledge about any new topic they encounter, and have to split their attention and (cognitive) resources between two or more competing cognitive inputs [22]. First, they are essentially trying to understand the problem domain, creating schemas or productions that create a setting (automaticity) for the problem. Second, they are trying to solve the underlying problem itself [29]. For systems developers this splitting of resources means that it is less likely that 1) accurate models will be created if the novice is a developer, and 2) less understanding of the system will be conveyed if the novice is a user. While developers are expected to hone their expertise over time, business users would not normally operate under that expectation, meaning that the cognitive load problem will not disappear over time, and could foster or be a

source of the user-designer gap issues so commonly cited as top reasons for system failures.

Sweller [28] noted that the increases in complexity (or cognitive load) experienced by novices during problem solving activities did not result in increases in schema creation (a deep understanding of the problem), but only increased problem solution (plug-and-play formulaic solutions). If this can be extended to the real world, it means that novices are less able to reach a deep understanding of a system when they are presented with a model representing the system, because they find it necessary to expend more effort understanding the elements composing the diagram or model itself.

2.4 Connection Between Cognitive and Structural Complexity

We propose to adopt the ideas on the definition of structural complexity as set forth by Briand, Wüst, and Lounis [5]. They believed that the physical (structural) complexity of diagrams affects the cognitive complexity faced by the humans using the diagrams as aids to understand and/or develop systems.

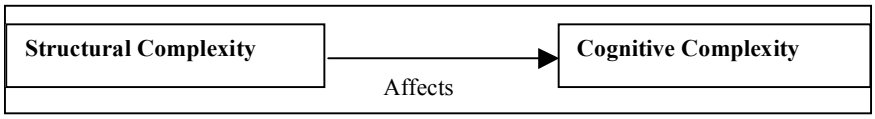

Figure 1: Adapted from Briand, Wüst, and Lounis [5]

Since measuring cognitive complexity is likely to be at best extremely difficult to measure, and at worst perhaps even impossible to measure, we propose that structural complexity can be considered as somewhat of a surrogate for cognitive complexity. While structural complexity may not be the only determinant of cognitive complexity, it is very likely a component. We view structural complexity as a function of the number of distinct elements (or constructs) that compose each specific diagramming technique. Rossi and Brinkkemper's [25] 17 metric definitions together form an approximation of the total structural complexity of the diagramming technique.

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, and interfaces), properties (class names, attributes, methods, and roles), and relationships and associations (aggregations generalizations, specializations). We use the [25] metrics as the operational definition and measure of the structural complexity of diagrams.

Siau et al [27] provided some evidence indicating that the theoretical metrics (total structural complexity) do not adequately represent the complexity practitioners face when using UML class and use case diagrams. In addition, Erickson and Siau [14] provided further support for the idea of practical complexity by conducting a Delphi study aimed at identifying a kernel of UML that was based on developer perceptions of the utility of the various constructs and diagrams comprising UML. The current research uses Erickson and Siau's [14] definition of practical complexity, as a subset of theoretical complexity, and proposes and compares user-defined kernels for real-time, Web-based, and enterprise systems.

2.5 Research Question

Can a user-based UML kernel be identified for specific UML application areas, in particular, Real-Time, Web-based, and Enterprise systems?

The research objectives for this study are, (i) using the Erickson and Siau results [14] determine the most important user-identified constructs in the UML diagrams for real-time, Web-based, and enterprise applications and, (ii) compare the results with the previously identified general case UML kernel. Figure 2 below indicates the intent of this investigation.

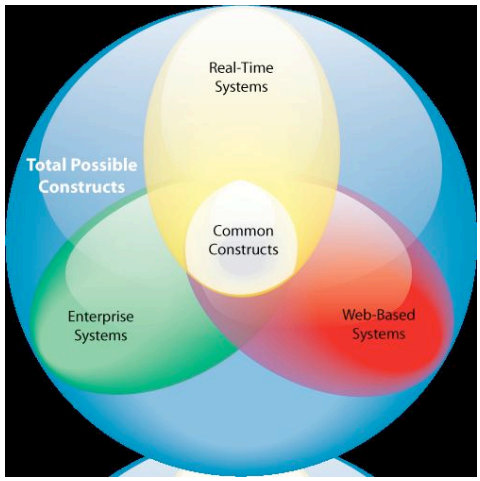

Figure 2: UML Domains and a Possible Core (Kernel)

3.0 Methods

3.1 Delphi Study

The research investigates the formulation of a series of UML kernels by means of a Delphi study. Delphi studies attempt to form a reliable consensus of a group of experts in specialized areas [20]. The approach is a process that focuses on collecting information from the expert group though a series of questionnaires, and providing feedback to the group between questionnaires. Usually the group of experts is geographically dispersed, as will be the case for many of the subjects participating in this research (i.e. the different companies and people involved). The questionnaires are usually designed to allow the collection of expert opinions on the subject, and then to facilitate the refinement or focus of the subsequent versions to narrow in on a consensus.

Many past and present researchers have used the Delphi approach to study various topics. According to Lawrence Day [8], Delphi studies have been used to examine and predict future advances in computers and technology, cosmetics, insurance, and recreation. AT&T studied the future of the telecommunications industry, MacMillan focused on the future of newsprint, and Skandia Insurance tried to identify and rank economic losses related to computer systems [8].

KUNNE [18] indicated that more than 463 research efforts used the Delphi method between 1975 and 1994. The ENRiP (Exploring New Roles in Practice) project sponsored by the University of Sheffield School of Nursing and Midwifery used a Delphi study to assess new roles in the practice of nursing [13]. The use of Delphi studies appears to spread across industries and time, while the methodologies tend to split into two camps; developing a consensus on the future, or developing a consensus on such areas as regional planning [19].

The approach for this study was to develop and administer a series of questionnaires that captures information regarding the UML constructs commonly used while building systems to a group of systems developers essentially identifying the kernel of UML. In our case, the Delphi study consisted of three rounds. Subjects were asked to rate the importance of the various diagrams and constructs in round 1. The results were analyzed and included in the questionnaire for round 2. Subjects were asked to reevaluate their ratings from the first round. Similarly, the results for round 2 were analyzed and included in the questionnaire for round 3. Subjects were again asked to reevaluate their ratings from the second round. A consensus level of 90% or higher was clearly reached after 3 rounds of the survey.

3.2 Participant Demographics

The subjects were asked to respond to their use of the standard UML constructs in general as well as real-time, Web-based and Enterprise applications. The application specific extensions for UML were used as the meta-constructs for each application domain. The expert respondents were also asked to rate the standard UML diagrams and constructs for each specific domain.

44 subjects agreed to participate in the study, and were sent Questionnaire 1. 29 returned useable surveys; a presentation of their demographic information follows. The average development experience of the 29 final respondents was 9.5 years, and the average UML development experience was 4.5 years.

Twenty eight of the respondents indicated that they had at least some experience in enterprise application development, with an average of 5.6 years. Twenty respondents also indicated web development experience, averaging 2.1 years, while nine respondents averaged 2.7 years of real-time application development experience. In addition, 2 respondents possessed experience in other development areas, one with 7 years of software development tool experience, and the other with 20 years in an unnamed development area.

The job titles and industry demographics indicate that 22 of the 29 initial respondents were in some form currently involved in academics, with 6 in the computer industry and 1 in financial services industry (as an application developer). Nine respondents classified themselves as students at the time of the survey, but since their stated development and UML experience met the criteria, this indicates that much of their experience in development was gained prior to entering school. Second, 14 respondents worked in academia outside of the United Sates (Canada, 3; Argentina, 1; Spain, 2; Norway, 3; Netherlands, 1; France, 2; Germany, 1; and Finland 1) meaning that many consult as developers outside the academic setting as a normal part of their lives. In addition, many of the US respondents US also possessed significant development consulting experience.

4.0 Results and Discussion

While the individual extension construct kernels could prove useful for domain practitioners, it is the experts' assessment of the 9 standard (UML 1.X) diagrams and related constructs that are used here to compose a picture of UML's kernel. The final order of importance was as follows in the below tables.

The participants were asked to rate the relative importance of the various UML diagrams and constructs in building systems. They were asked to rate the importance on a scale of 1 to 5; 1 as very important and 5 as very unimportant, and 0 if the diagram or construct was never used. The analysis was relatively basic in that means and standard deviations were the only calculations made. After the first round, the respondents were also asked whether the particular construct or diagram in question should be included in a UML kernel. This means that there were 2 ways to get at the kernel information; the Yes/No kernel question, and the mean scores of the importance ratings. Generally, a mean score of 2.00 or less corresponded fairly closely with a high level of "Yes" to include in the kernel consensus.

Table 1: UML Overall Results (non domain specific)

Construct	Mean	Standard Deviation	% "Yes" for Kernel
Class	***1.00***	***0.00***	***100.0%***
Use Case	***1.61***	***0.79***	***90.9%***
Sequence	***1.73***	***0.70***	***95.5%***
Statechart	***1.81***	***0.51***	***100.0%***
Component	2.31	0.70	31.8%
Activity	2.41	0.55	27.3%
Collaboration	2.57	0.87	22.7%
Deployment	2.69	0.75	9.1%
Object	3.00	0.86	9.1%

Table 2 UML Real-Time System Diagram results

Construct	Mean	Standard Deviation	% Y for Kernel
Class	***1.23***	***0.60***	***100.0%***
Statechart	***1.31***	***0.48***	***92.3%***
Sequence	***1.46***	***0.52***	***100.0%***
Use Case	***1.92***	***0.86***	***100.0%***
Component	2.00	0.82	23.1%
Activity	2.11	0.33	53.8%
Deployment	2.38	0.74	0.0%
Object	2.91	1.04	0.0%
Collaboration	3.10	0.88	0.0%

Table 3: UML Web-based System Diagram Results

Construct	Mean	Standard Deviation	% Y for Kernel
Class	***1.07***	***0.27***	***100.0%***
Use Case	***1.43***	***0.85***	***92.9%***
Sequence	***1.68***	***0.61***	***100.0%***
Statechart	***1.83***	***0.39***	***100.0%***
Component	2.00	0.82	42.9%
Deployment	2.44	0.88	14.3%
Activity	2.67	0.50	14.3%
Collaboration	2.91	0.83	14.3%
Object	3.40	1.17	7.1%

Table 4: UML Enterprise System Diagram Results

Construct	Mean	Standard Deviation	% Y for Kernel
Class	***1.00***	***0.00***	***100.0%***
Use Case	***1.53***	***0.77***	***94.7%***
Sequence	***1.58***	***0.69***	***100.0%***
Activity	***1.86***	***0.66***	***94.7%***
Deployment	2.27	0.79	21.1%
Component	2.33	0.82	42.1%
Statechart	2.58	0.77	36.8%
Collaboration	2.81	0.83	10.5%
Object	3.06	1.12	15.8%

4.1 Discussion

The results indicate participant agreement on 3 of the 4 kernels. Respondents agree that not only 3 of the diagrams should be included in a kernel, but also on the ordering of the 3 diagram types in the various domains. With minor variations in mean and consensus, the 3 diagram types the Delphi participants all agreed on regardless of system type, were Class, Use Case and Sequence diagrams. Based on these relatively clear results, we propose that a kernel of UML include those three diagrams and associated constructs. In addition, since Class diagrams model a static system view, and use case diagrams essentially model a process view, it also makes sense that at least one modeling technique that captures some of the dynamic elements of systems (Sequence diagrams) be included in a core or kernel of UML.

Possible reasons for this might be that UML is use case driven, and use case models generally represent the starting point for a UML based modeling project. However, the Delphi group almost universally rated Class diagrams as most important. It should be noted that class diagrams usually involve implementation, and is possibly why practitioners emphasize it more. This issue also highlights a critical problem with many development efforts, the user-designer gap. Another possible interpretation of these results is that practitioners tend to use UML to model more heavily in the Analysis stage of the system development process, and less for Design, Testing, and Implementation. The least useful diagrams were perceived to be object, deployment, component, or deployment, although there was some disagreement regarding the utility of collaboration diagrams.

For the specific domains we recommend as follows. Real-time systems projects are more likely to require modeling of state changes and state machines, so statechart (state machine) diagrams are important for that application type. Enterprise systems are more likely to require models of activities, and thus Activity diagrams would be more critical for that domain. Finally, for Web-based systems, the Delphi participants identified an identical kernel to the non-domain specific kernel.

5.0 Conclusion and Limitations

While the UML core identified here is naturally arbitrary and based on user importance ratings, the results do provide a working compilation of the constructs that developers most commonly use when building systems. The premise of the research here is not to propose that the remaining constructs be excluded from UML, but rather that those features be retained in the language, perhaps in specialization modules. In contrast to Erickson and Siau's 2004 [14] results, this research identifies a slightly

different UML kernel, one that likely reflects the needs of applications operating in multiple environments. However, it is also noteworthy that the kernel differences are relatively small as well, since the kernels identified are identical in both cases, except for the order of the diagrams.

The research results are naturally limited by a number of factors. Delphi studies have been often criticized for their lack of rigor. The selection of the experts for a successful Delphi is also critical, and while we made every effort to ensure that the participants were true UML experts, it will always be possible to debate the issue of expertise. The study was conducted in 2003-2004, and the recruitment of participants lead the conduct of the study slightly, so the 4.5 years of average UML experience at that point in time corresponded to a rough maximum possible, since UML was adopted as a standard in 1997. In addition, participants had an average of 9.5 years of development experience, so while some were in academia at the time of their participation, their experience in development was the determining factor for inclusion as participants in the study. Experience in the various domains may have been a limiting factor, but since the UML domain specifications were also relatively new at the time the data was collected, then the UML development experience as well as the domain experience could be seen as overlapping.

This research should be considered as a small first step in identifying a use-based UML kernel. The OMG has also identified a kernel for the language, and we will examine the similarities and differences if possible in the future. Another limitation deals with the 3 chosen domains. At the time the study was conducted, we chose domains for which a fully specified UML extension had been constructed. Future research will examine other domains as other specifications become available.

These results have a number of possible implications. First, researchers in the method engineering area have been among those critical of UML, not only for its complexity, but also for the inconsistencies in the meta-model. These results could help guide efforts to remediate some of the complexity and inconsistencies. Second, the OMG and other interested parties (the Model Driven Architecture area) could also use these results to seriously question and examine how people are really using UML in the field. Tied together, this could actually mean progress rather than regress. Finally, educators in the business of teaching system development techniques, and practitioners, in the business of using system development techniques, could also profit from these results by spending precious educational, training and development resources on what is really important in the quest to improve systems.

References

1. Anderson, J., and Lebiere, C. 1998. The Atomic Components of Thought, Lawrence Erlbaum Associates.
2. Baddeley, A. 1992. "Working Memory". *Science*. Vol. 255. P. 556-559.
3. Baddeley, A. 2003. "Working Memory: Looking Back and Looking Forward". *Neuroscience*. Vol. 4. P. 829-839.
4. Booch, G., Rumbaugh, J., and Jacobson, I. 1999. The Unified Modeling Language User Guide. MA. Addison-Wesley.
5. 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. P. 345-354.
6. Conallen, J., 2000. Building Web-Applications with UML. Addison-Wesley.

7. Burton-Jones, A. and Meso, P. 2002. "How Good are These UML Diagrams? An Empirical Test of the Wand and Weber Good Decomposition Model". International Conference on Information Systems. P. 101-114.
8. Day, L. 2002 "Delphi Research in the Corporate Environment". In H. Linstone and M. Turoff (eds.). The Delphi Method Techniques and Applications. http://www.is.njit.edu/pubs/delphibook/index.html.
9. Dobing, B. and Parsons, J. 2000. Understanding the Role of Use Cases in UML: A Review & Research Agenda. *Journal of Database Management*. Vol. 11. No. 4. P. 28-36.
10. Dori, D. 2002. "Why Significant UML Change is Unlikely". *Communications of the ACM*. Vol. 45. No. 11. P. 82-85.
11. Douglass, B. 2000. Real-Time UML Second Edition: Developing Efficient Objects for Embedded Systems. Addison-Wesley.
12. Duddy, K. 2002. "UML2 Must Enable a Family of Languages". *Communications of the ACM*. Vol. 45. No. 11. P. 73-75.
13. ENRiP. 2001. University of Sheffield http://www.snm.shef.ac.uk/research/enrip/refs.htm
14. Erickson, J. and Siau, K. 2004. "Theoretical and Practical Complexity of Unified Modeling Language: Delphi Study and Metrics Analyses". International Conference on Information Systems. Washington, DC. December.
15. Erickson, J., Siau, K. 2003. "Unified Modeling Language: The Good, The Bad, and The Ugly," in: Toppi, H., Brown, C. (eds.). IS Management Handbook. Auerbach.
16. Ericsson, K. and Kintsch, W. 1995. "Long-Term Working Memory". *Psychological Review*. Vol. 102. No. 2. P 211-245.
17. Green, G. 2004. "The Impact of Cognitive Complexity on Project Leadership Performance". *Information and Software Technology*. Vol. 46. P. 165-172.
18. Kobryn, C. 2002. "What to Expect from UML 2.0". *SD Times*. accessed 10/22-2002.
19. KUNNE web site. 2002. accessed 2003. http://www.kunne.no/meritum
20. Ludwig, B. 1997. "Predicting the Future: Have you considered using the Delphi Methodology?". *Extension Journal*. October. Vol. 35. No. 5.
21. Marshall, C. 2000. Enterprise Modeling with UML Designing Successful Software Through Business Analysis. Addison-Wesley.
22. Mayer, R. 1989. "Models for Understanding". *Review of Ed. Research*. Vol. 59. P. 43-64.
23. 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.
24. Object Management Group Web site. http://www.omg.org/gettingstarted/what_isuml.htm.
25. Rossi, M. and Brinkkemper, S. 1996. "Complexity Metrics for Systems Development Methods and Techniques". *Information Systems*. Vol. 21. No. 2. P. 209-227.
26. Siau, K., and Cao, Q. 2001. "Unified Modeling Language (UML) – A Complexity Analysis". *Journal of Database Management*. Jan – Mar.
27. 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.
28. Sweller, J. 1988. "Cognitive load During Problem Solving: Effects on learning". *Cognitive Science*. Vol. 12. P. 257-285.
29. Sweller, J. and Chandler, P. 1994. "Why Some material is Difficult to Learn". *Cognition and Instruction*. Vol. 12. P. 185-233.
30. 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. P. 21-30.