A Framework for Classifying and Comparing Graphical Object Oriented Modeling Languages

Ali Kamandi, and Jafar Habibi

Abstract—**Object-oriented modeling has become the de-facto standard in the software development process during the last decades. A great deal of research in this area focuses on proposing modeling languages.**

In order to properly understand, and assess an object oriented modeling language, we believe that a set of criteria or requirements is needed. This Paper presents a framework to investigate and compare graphical object oriented modeling languages. This framework is based on a requirement set for an ideal object-oriented modeling languages.

Index Terms—**Object-Oriented modeling languages, comparison framework, requirement set, UML**

I. Introduction

Suitable modeling languages are needed to describe the conceptual construct underlying software. These languages, which are often graphical, can be used to produce a satisfactory description of the conceptual constructs, frequently prior to writing any code. Prior construction of a model for a derived software system is as essential as having a blueprint for a building or a schematic for a circuit before building them [29]. A modeling language is a language used to specify, visualize, construct, and document a software system.

The UML was born out of the unification of the many object oriented graphical modeling languages that thrived in the late 1980s and early 1990s [11]. It has rapidly been accepted throughout the software industry as the standard graphical language for specifying, constructing, visualizing, and documenting software-intensive systems [3].

The evolution process seems to have gone astray, and as a result, we are witnessing the return of some of the older methodologies (such as RDD, [40]). At the same time, some of the methodologies or variants introduced today (such as EUP [1], OPM [7], and FOOM [36]) do not even adhere to UML modeling conventions. On the other hand, the Executable UML [22] and OMG's Model-Driven Architecture (MDA) [25], the general development approach based on transforming logical models of the system (called Platform-Independent Models – PIMs) into physical implementation models (called Platform-Specific Models – PSMs) [25], is still in its early stages of development.

Realizing the need and potential for further improvement in the field, it is important to point out that the relatively long history of object oriented modeling languages is a rich source of lessons to be learned. In every language, there are features to exploit and pitfalls to avoid.

Several techniques such as empirical studies, model based evaluation and metric based evaluation, have accomplished to evaluate and compare modeling languages. This paper proposes a criteria set that can be used both as requirement specification in new OO modeling language development and as a framework for comparison existing languages.

The rest of this paper is organized as follows: Section II overviews the related work, section III presents state of the art in object oriented modeling languages. Section IV introduces the proposed requirement set for graphical object oriented modeling languages. Finally, section V draws conclusions.

Ali Kamandi is with the Computer Engineering department, Sharif University of Technology, Tehran, Iran (e-mail: kamandi@ce.sharif.edu).

Jafar Habibi is with the Computer Engineering department, Sharif University of Technology, Tehran, Iran (e-mail: habibi@sharif.edu).

II. Related Work

Engels and Groenewegen in [8] introduced a list of requirements for an ideal object-oriented modeling language. They also compared the achievements of the UML with the ideal language according to these requirements. Among the requirements for an ideal language, user-friendliness, precision, understandability, separation of concerns, modularization, scalability, consistency and horizontal/vertical composition are the most important.

Ramsin in his PhD thesis [31] mentioned two requirement sets: one for object-oriented methodologies and another for object-oriented modeling languages. He mentioned two important requirements for OO modeling languages: 1) Support for consistent, accurate and unambiguous object-oriented modeling; and 2) Provision of strategies and techniques for tackling model inconsistency and managing model complexity.

In 1996 Rossi and Brinkkemper [33] 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 meta-model constructs, and were specifically created to be measure the complexity of virtually any (diagrammatic, structurally based) modeling method.

Paige et al. argued that modeling languages, like programming languages, need to be designed [29]. They presented principles for design of modeling languages. They conjectured that the principles are applicable to the development of new modeling languages, and for improving the design of existing modeling languages that have evolved, perhaps through a process of unification [29]. They proposed nine principles including simplicity, uniqueness, consistency, seamlessness, reversibility, scalability, supportability, reliability and space economy.

Krogstie has developed a generic quality framework for discussing the quality of models in general, motivating the focus on language quality as a means to achieve models of high quality. Five areas for language quality are identified with aspects related to both the meta-model and the notation [18]:

- Domain appropriateness: the conceptual basis must be powerful enough to express anything in the domain [38].
- Participant language knowledge appropriateness: the conceptual basis should correspond as much as possible to the way that individuals perceive reality.
- Knowledge externalizability appropriateness: the goal is to ensure that there is no statement in the explicit knowledge of the participant that cannot be expressed in the language.
- Comprehensibility appropriateness: the phenomena of the language should be easily distinguishable from each other, the number of them should be reasonable, the use of the phenomena should be uniform, symbolic simplicity should be a goal and so on.
- Technical actor interpretation appropriateness: it is important that the language lend itself to automatic reasoning. This requires formal syntax and semantics.

III. State of the Art

Before gathering the requirements for an ideal object-oriented modeling approach, we will briefly summarize in this section the current state-of-the art in object-oriented modeling languages in industry and research. This forms the basis for identifying drawbacks and open issues to be investigated in future. Introducing UML, it seems that the method war is finished, but after it several modeling notations and languages are introduced and used in software industry. Among them we will investigate and present some important ones.

A. UML

UML was born out of the unification of the many object oriented modeling languages in 1997 after methodology war in the mid 1990s and has rapidly been accepted throughout the software industry as the standard graphical OO modeling language [26]. UML was intended as a general purpose object-oriented modeling language. UML 2.0 provides more than 13 diagrams for modeling functional, structural and behavioral aspect of software.

B. OML

The OPEN (Object-oriented Process, Environment, and Notation) development methodology provides a modeling language named OML and a development process [13]. The OML language is composed of a COMN (Common Object Modeling Language) notation and a meta-model [10]. The COMN notation, like UML, offers a set of diagram types, which are used to model software systems. Some diagrams document the static structure; others specify the dynamic behavior of an application.

Diagrams of OML include: semantic nets, context diagrams, Layer diagrams, Configuration diagrams, Package diagram, Inheritance diagram, Scenario class diagram, Interaction diagram, Black-box sequence diagram, White-box sequence diagram, Package collaboration diagram, Scenario collaboration diagram, Internal collaboration diagram and State transition diagram.

Several papers concluded that there is a lack of formality and correctness in both the descriptions of diagrams and language constructs, and also the semantics of OO concepts were not complete. [15], [2], [14].

C. BON

The BON (Business Object Notation) Methodology presents a set of concepts for modeling object-oriented software, a supporting notation in two versions-one graphical and one textual-and a set of rules and guidelines to be used in producing the models [39].

BON concentrates on the seamless, reversible specification of software, using the contract model. BON includes several models: system chart, cluster chart, scenario chart, static architecture, class dictionary, class chart, event chart, creation chart, object scenario, class interface and system execution scenario. The notation provides mechanisms for modeling inheritance and usage relationships between classes, and has a small collection of techniques for expressing dynamic relationships. The notation also includes as assertion language [28]. BON provides only a small collection of powerful modeling features that guarantee seamlessness and full reversibility on the static modeling notations. BON is architecture-centric and contract driven, but not use-case driven [28].

D. OPM

Object-Process Methodology (OPM) was introduced by Dori in 1995 [6]. OPM's modeling strength lies in the fact that only one type of diagram is used for modeling the structure, function and behavior of the system. This single-model approach avoids the problems associated with model multiplicity, but the model that is produced can be complex and hard to grasp.

The Object-Process Methodology (OPM) has been shown to successfully describe the structure and behavior of systems using an integrated and coherent set of Object-Process Diagrams (OPDs) [6], [7], [19]. OPD uses elements of types object and process to model the structural, functional and behavioral aspects of whatever is being modeled.

OPM includes a clear and concise set of symbols that form a language enabling the expression of the system's building blocks as well as their relationship to each other.

OPM inherits its capabilities from both object oriented and process oriented paradigms. OPM is an integrated approach to the study and development of software systems. In OPM, objects and processes have equal status and are described as things or entities. OPM handles complex systems by using recursive seamless scaling. OPM is not pure object-oriented, because behavior in OPM is not necessarily encapsulated within a particular object class construct: using stand-alone processes, one can model a behavior that involves several object classes and is integrated into the system structure.

IV. Modeling Language Requirement Set

This section describes a list of proposed criteria and requirements for object oriented modeling languages.

A. Consistency

Consistency means mutual agreement and logical coherence of different models and diagrams. Model inconsistency is a dire problem. UML has exacerbated the situation instead of improving it [7], [31], [17]. Different models produced for a system should not be allowed to contradict each other; alternatively, there should be mechanisms for detecting inconsistencies. Paige et al. argued [29]: "Some modeling languages, e.g., UML, allow designers to describe a system in several independently constructed models – e.g., a class diagram, deployment diagram, use-case diagram, sequence diagram, et cetera – and at implementation time, these models must be checked for consistency, i.e., that something said in one model is not contradicted by something said in another model".

Simons and Graham [35] enumerated the problems experienced by the developers as they embraced the UML notation and engaged in what they considered to be the most appropriate sequence of activities for building UML models. They classified these problems into four categories, which one of them is inconsistency, meaning that parts of UML models are in contradiction with other parts, or with commonly accepted definitions of terms.

Because the collection of models that can be produced using UML is large, and because each model itself may be complex (containing many different abstractions and relationships), checking the consistency of a UML specification is non-trivial, and it is questionable whether it can be automated. A contrasting approach is offered by BON: therein, a single model is constructed for each class, and checking the consistency of this model is straightforward and can be assisted by automated tools [29].

Instead, BON concentrates on what is essential for object-oriented development in general, and tries to define a consistent notation to support the corresponding concepts. The user is then free to complement the notation with whatever more might be needed for a particular project [39].

B. *Comprehensibility*

Some modeling languages are too complex to be effectively mastered, configured, and enacted. Perhaps the most important of all general principles for conceptual models, as well as for notations, is simplicity [29]. The deep results of the natural sciences seem to indicate that nature is inherently simple—that complexity is only introduced by our lack of understanding [39]. Complexity can be interpreted in two ways: structural or cognitive. The cognitive or psychological complexity is usually known as comprehensibility [27].

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 [9].

Cognitive Load Theory assumes that novices have little pre-existing knowledge about any new topic they encounter [20] and 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 [37],[9].

Siau and Cao carried out a comparison of the practical complexity of UML with other Object-Oriented (OO) techniques. Their results concluded that individual diagrams in the UML are not more complex than the diagrams in other OO methods. However as a whole, UML is 2-11 times more complex than other OO methods [34]. Zendler et al. [41] compared the comprehensibility of the coarse-grained concepts in three object-oriented approaches: UML, OML and TOS (Taxonomic Object System). The results showed that when modeling a database-oriented application, the coarse-grained concepts of OML and TOS were better than those of UML.

Otero and Dolado compared UML and OML empirically. In their study two dependent variables are used in order to assess the semantic comprehension: 1) the amount of time spent answering each question (comprehension time) and 2) number of correct answers [27].

The obtained results reveal that the average time required understanding the model in OML was shorter and subjects understood this language and were more consistent in their answers when the COMN notation was involved [27].

Reinhartz-Berger and Dori compared UML and OPM for web applications [32]. In this study, the results suggest that OPM is better than UML in modeling the dynamics aspect of the Web applications. In specifying structure and distribution aspects, there were no significant differences. They concluded that the single OPM diagram type, the Object-Process Diagram (OPD), which supports the various structural and dynamic aspects throughout the system lifecycle, is easier to understand and apply by untrained users [32].

C. *Simplicity (Structural)*

Structural complexity is more closely connected to the physical properties of the diagramming techniques found in modeling approaches such as UML diagrams [9].

Briand, Wüst, and Lounis [4] 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.

The UML structural complexity was evaluated by Hahn and Jinwoo [12] and Purchase et al. [30] to improve the comprehension and use of the UML models. Compared to other modeling methods and languages, UML is very complex [9].

Purchase et al. [30] evaluated the aesthetic effect of the layout of graphic elements in the UML diagrams. From the point of view of usability, the aesthetic preferences are empirically investigated in class and collaboration diagrams, with the purpose of reducing the number of crossings, and increasing the display of symmetry. In conclusion, they obtained a ranking of aesthetic aspects to consider in the future design of UML graph drawing algorithms.

D. *Compactness*

An extensible core set of models and diagrams is preferable to a customizable monstrosity. referring to lightness and simplicity of models, and its being free of nonessential, excess features; hefty and complex notations are hard to understand and master, and difficult to use.

The BON notation strives for simplicity and tries to minimize the number of concepts. For example, there are only two basic relations between classes: the inheritance relation and the client relation. To obtain multiple views of a system, we also need relations between classes and clusters and between clusters and clusters. However, instead of introducing new concepts BON uses the compression mechanism (generalizing the class relations), and give them well-defined semantics when clusters are involved. [39]

Despite its evolution and status as standard language, UML has also been criticized for its complexity, inconsistent semantics and ambiguous constructs. Some of its detractors question the usefulness of having 13 diagramming techniques in UML, since the language becomes more complex and more difficult to learn [35]. Siau and Cao argued that 80% of systems are developed by using only 20% of the language constructs, relating to a practical or use-based complexity [34].

Hodgett studied usage of UML diagrams in Australian information technology industry. The study showed that few used or continued to use all the views or aspects of the modeling language. The parts selected depended on the application and the comments indicated that part of the lack of support was because most business applications are not real time and do not have the degree of size or complexity that might justify the use of a wider range of UML views. It was recognized that Use Cases were the core of the methodology. Class and Object diagrams followed as the most accepted parts of the language while Activity diagrams were universally disregarded. Criticism of these latter diagrams seemed to center around the time taken to model minutiae while comments from programmers indicated that this level of detail was not required [16].

E. Extensibility

Extensibility specifies the degree to which the modeling language can be extended to support new concepts. Extensibility is an attribute of something that allows it to last or continue, or to be expanded in range or scope [24]. UML is a good example for extensibility. It provides several mechanisms such as stereotypes and profiles for language extension.

F. Traceability

Traceability specifies the degree to which models can be shown to have stemmed from the requirements. Requirements engineering is still the weak link, and requirements traceability is rarely supported; requirements are either not adequately captured or partially lost or corrupted during the development process [23], [31]. The main issue to consider is that both functional and non-functional requirements must be considered.

G. Coverage of standard software development activities

This criterion means covering activities constituting or supporting the generic software development lifecycle (Analysis, Design, Implementation, Test, and Maintenance). There are many ways to describe a software system, but all descriptions can be characterized as being either static or dynamic. Static descriptions document the structure of a system: what the components are and how these components are related to each other. Dynamic descriptions, by contrast, document how the system will behave over time. In an object-oriented context, this means describing how objects interact at execution time; how they invoke operations on each other (passing messages in Smalltalk terminology) and how the information content of the system changes [39].

H. Scalability

Scalability means manageability of complexity (Hierarchical Structure); Provision of strategies and techniques for managing model complexity [31]. We must make sure that the notation will scale up, and still be useful for large systems [39]. Hierarchical structure is the main mechanism to manage complexity of large systems. The second thing to note when scaling up is that flat partitioning is not enough to get comprehensive views of large systems. We need views at different levels of detail and the ability to "zoom" between them [39].

As the size of a system increases, some mechanism becomes required to limit the visibility of information to only those objects of interest at a particular time [24].

BON proposed cluster as a facility to group classes into higher-level units. UML uses packages (and subsystems) for this purpose. OML uses packages too. OPM separates diagrams at different abstraction levels.

I. Clear Definition

One important criteria of language design is accuracy, unambiguous, and consistency of definition. The concepts of the language should be easily distinguishable from each other. The number of underlying concepts should be reasonable. Use of a concept or notation in different models and diagrams should be unified [18]. Language should be documented in a precise and accurate way. Its documentation should be readable and understandable for modelers. Language definition should be accurate enough that tool developers can develop CASE tools and modeling environments without ambiguity. The syntax and semantics should, in addition to being expressive, also be well-defined [24].

Simons and Graham enumerated the problems related to language definition, experienced by the developers, which includes ambiguity, meaning that some UML models are under-specified, allowing developers to interpret them in more than one way and adequacy, meaning that some important analysis and design concepts could not be captured using UML notations. These problems are summarized in figure 1.

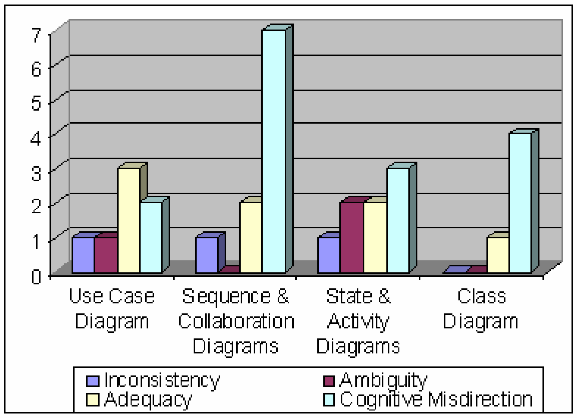

Figure 1: distribution of problems reported by Simons and Graham

J. Generality

Not restricted to any specific application domains, special need of certain applications or programming languages [39].

UML is not specified to special programming language or environment. Also there exist some extensions and special frameworks based on UML for special environments and frameworks such as Java, C++, J2EE and CORBA. BON was developed inherently for Eiffel language.

K. *Reversibility*

To promote reuse and achieve true seamlessness, the core elements of a notation for analysis and design should represent concepts that are directly mapable not only to, but also from, an executable object-oriented language. Besides making it possible to maintain long-term consistency between specification and implementation, reversibility is also important for the reuse of analysis and design elements [39]. Reversibility is one of the main objectives of BON. UML, OPM and OML are weak in this area.

L. *Domain appropriateness*

Is the notation suited to a particular application domain? Some notations are targeted at real-time and embedded or interactive systems applications, while other notations are more suited for information systems development. Domain appropriateness is also proposed by Krogstie [18]. “Ideally, the conceptual basis must be powerful enough to express anything in the domain, on the other hand you should not be able to express things that are not in the domain [38],[18].

UML is a general modeling language, specially having extension mechanisms such as profiles and stereotypes make it possible to model various kinds of systems with UML. Conallen presented how to model web applications with UML [5].

Erickson and Siau presented that UML class, use case, sequence and statechart diagrams are best suitable as its core generally [9]. They also showed that class, statechart, sequence and use case are more appropriate for Real-time systems. Class, use case, sequence, statechart are more appropriate for web-based systems. And class, use case, sequence and activity diagram are more suitable for enterprise systems.

M. *Tool Support*

The availability of suitable CASE tool to facilitate the description and examination of the system from various points of views using the specified modeling language. UML has several CASE tools, but others have only some CASE tools.

N. *Analyzability*

Language should lend itself to automatic reasoning. This requires formal syntax and semantics. Formal semantics can be operational, logical, or both. Formality is not sufficient since the reasoning must also be efficient for practical use. BON is more powerful relating to analyzability, because of its formalism added to classes. Others have less formalism. UML is very weak regarding analyzability, because of having several modeling diagrams make it difficult to integrate models and construct one integrated model that can be used for analysis [17]. Performance analysis, validation, verification (dead lock detection, etc.) are the major topics in this area.

O. *Space economy*

The amount of information that can be conveyed by an overview of some part of a system (what can be made to fit on one page).

Developers can build very compact models using OPM, because it merges both dynamic and static views. BON is in the second position. UML having several diagrams (some of them are not orthogonal) is not very good in this respect. Hodge-Mock in the last position, because it used different diagrams for concepts that can be merged in one diagram (i.e. it separated inheritance diagram from class diagram).

P. *Support business*

Medvidovic [21] proposed three lamp posts that each perfect ADL must balance between them: technology, domain and business. System’s business position includes its relationship to other products, time-to-market, and so on. An effective modeling language must strike a proper balance between a strict focus on recurring technical concerns mandated by different application domains and business contexts [21].

Unfortunately all of these languages are weak in this aspect. Neither of them have features for predicting business status of system which is modeled within the language.

Q. *Expressiveness*

The *expressiveness* of each method's notation is evaluated for its support for such concepts as aggregation, generalization/ specialization, and object interaction. If the notation is not sufficiently expressive the user is required to encode the representation in an ad hoc manner, often as unformatted text attached to the model, or maintained separately from the model. This leads to inconsistent, more complex, and less easily understood models [24]. All of mentioned notations have enough concepts for modeling object oriented systems (such as class, inheritance, aggregation, association and so on).

R. *Orthogonality*

Ideally, each model represents some aspect of the system not represented completely by another model, yet each model provides "clues" or direction in the partial or complete creation of other models [24]. Each model within a notation should contribute to the overall understanding of the problem or design. Yet each model should not be entirely orthogonal to every other model in the notation. This interaction of models also assists in the verification and software quality assurance of the models.

Simple languages such as BON and OPM have orthogonal models, but large languages such as UML have many concepts and diagrams, which some of them are not orthogonal, for example, sequence diagram and communication diagram are very similar and both of them can be used for behavioral modeling.

V. Conclusion

This paper proposed a new framework and criteria set for evaluation, classification and comparison of object oriented modeling languages. This criterion set can be used also as a requirement set for new object oriented modeling language development. This requirement set is essential in order to properly understand, and assess an object oriented modeling language. Table 1 shows a summarization of comparison of modeling languages such as UML, BON, OML and OPM. This comparison is based on existing studies and evidences, and more research is needed to improve its accuracy and validity.

Table 1: Comparison of modeling languages according to proposed framework

Criteria\Language	UML	BON	OML	OPM
Comprehensibility	--	++	-	++
Consistency	--	++	-	+++
Structural Simplicity	+	+	+	+
Compactness	-	+	-	++
Extensibility	++	-	-	-
traceability	++	++	+	-
Coverage of Software Dev. Cycle	++	+	++	-
Scalability	++	++	++	+
Clear Definition	+	++	+	++
Generality	+++	+	+	+
Reversibility	-	+	-	-
Domain Appropriateness	++	+	++	++
Tool Support	+++	+	+	+
Analyzability	-	++	-	++
Space Economy	-	+	-	+
Support Business	-	-	-	-
Expressiveness	+++	++	++	++
Orthogonality	--	++	-	++

REFERENCES

[1] Ambler, S. W., and L. L. Constantine, *The Unified Process Inception Phase*, CMP Books, Gilroy, CA., 2000.

[2] F. Barbier, and B. Henderson-Sellers, "Object modeling languages: An evaluation and some key expectations for the future", *Annals of Software Engineering 10*, 2000, pp. 67-101.

[3] Booch, G., J. Rumbaugh, and I. Jacobson, *Unified Modeling Language-User's Guide*, Addison-Wesley, Reading, Mass, 1999.

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

[5] Conallen, J., *Building Web Applications with UML*. Reading: Addison-Wesley, 1999.

[6] D. Dori, "Object-process analysis: Maintaining the balance between system structure and behavior", *Journal of Logic and Computation 5, 2 (April)*, 1995, pp. 227-249.

[7] Dori, D., *Object-Process Methodology: A Holistic Systems Paradigm*, Springer, Berlin-New York, 2002.

[8] G. Engels, and L. Groenewegen, "Object-oriented modeling: a roadmap", *In Proceedings of the Conference on the Future of Software Engineering –ACM/ICSE*, 2000, pp. 103-116.

[9] J. Erickson, and K. Siau, "Can UML Be Simplified? Practitioner Use of UML in Separate Domains", *In Proc. of the Workshop on Exploring Modeling Methods for Systems Analysis and Design (EMMSAD'07), Trondheim, Norway*, Tapir Academic Press, Trondheim, Norway, 2007, pp. 87-96.

[10] Firesmith, D. G., B. Henderson-Sellers, I. Graham, M. Page-Jones, Open *Modeling Language (OML) reference Manual*, SIGS Books & Multimedia, 1998.

[11] Fowler, M., *UML Distilled: a brief guide to the standard object oriented modeling language*, 3rd ed. Addison-Wesley, 2004.

[12] J. Hahn, and K. Jinwoo, "Why are some diagrams easier to work with? Effects of diagrammatic representation on the cognitive integration process of system analysis and design", *ACM Transactions on Computer-Human Interaction 6 (3)*, 1999, pp. 181-213.

[13] Henderson-Seller, B., A. Simons, and H. Younessi, *The OPEN Toolbox of Techniques*, Addison Wesley, Reading, MA, 1998.

[14] B. Henderson-Sellers, and D. G. Firesmith, "Comparing OPEN and UML: the two third-generation OO development approaches", *Information and Software Technology 41*, 1999, pp. 139-156.

[15] B. Henderson-Sellers, G. Collins, R. Due´, I. Graham, "A qualitative comparison of the two processes for object-oriented software development", *Information and Software Technology 43 (12)*, 2001, pp. 705-724.

[16] R. A. Hodgett, "The acceptance of Object-Oriented Development Methodologies in Australian Organizations and the Place of UML in Information System Programs", *Information Science*, 2003.

[17] A. Kamandi and J. Habibi, "Evaluating UML according to modeling language design principles and new requirements", *Information and Knowledge Technology (IKT2007)*, Mashhad, Iran, Nov. 27-29, 2007.

[18] J. Krogstie, "Evaluating UML using a generic quality framework", *In Idea Group Publishing (Eds.), UML and Unified Process*, 2003, pp. 1-22.

[19] Y. Liu, L. Wenyin, and C. Jiang, "Object-Process diagrams as explicit graphic tool for Web Service composition", *Journal of Integrated Design and Process Science 8 (1)*, 2004, pp. 113-127.

[20] R. Mayer, "Models for Understanding", *Review of Ed. Research 59,* 1989, pp. 43-64.

[21] N. Medvidovic, E. M. Dashofy, and R. N. Taylor, "Moving architectural description from under the technology lamppost", *Information and Software Technology 49*, 2007, pp. 12-31.

[22] Mellor, S., and M. Balcer, *Executable UML*, Addison-Wesley, 2002.

[23] B. Nuseibeh, and S. Easterbrook, "Requirements engineering: A roadmap", *In Proceedings of the Conference on the Future of Software Engineering- ACM/ICSE 2000*, 2000, pp. 35-46.

[24] A Comparison of Object-Oriented Development Methodologies. The Object Agency, Inc., 1995

[25] OMG, Model Driven Architecture (MDA). Object Management Group (OMG), 2001

[26] OMG, Unified Modeling Language Specification (v2.0). Object Management Group (OMG), 2004.

[27] M. C. Otero, and J. J. Dolado, "An empirical comparison of the dynamic modeling in OML and UML", *The Journal of Systems and Software 77,* 2005, pp. 91-102.

[28] R. F. Paige, and J. S. Ostroff, "A Comparison of the Business Object Notation and the Unified Modeling Language", *In R. France, B. Rumpe (eds.): UML99-The Unified Modeling Language, Beyond the Standards. Second Int. Conf. Fort Collins, Co. LNCS 1723, Springer*, 1999, pp. 67-82.

[29] R. F. Paige, J. S. Ostroff, and P. J. Brooke, "Principles for modeling language design", *Infromation and Software Technology 42,* 2000, pp. 665-675.

[30] H. C. Purchase, J. Allder, and D. Carrington, "User Preference of graph layout aesthetics: A UML study", *Proceedings of Graph Drawing: 8th International Symposium GD 2000, LNCS 1984*, 2001, pp. 5-18.

[31] Ramsin, R.: The Engineering of an Object-Oriented Software Development Methodology. Ph.D. Thesis, University of York, York, UK, 2006.

[32] I. Reinhartz-Berger, and D. DORI, "OPM vs. UML-Experimenting with Comprehension and Construction of Web Application Models", *Empirical Software Engineering (Springer) 10,* 2005, pp. 57–79

[33] M. Rossi, and S. Brinkkemper, "Complexity Metrics for Systems Development Methods and Techniques", *Information Systems 21 (2),* 1996, pp. 209-227.

[34] K. Siau, and Q. Cao, "Unified Modeling Language (UML)- A Complexity Analysis" Journal of Database Management 12 (1), 2001, pp. 26-34.

[35] A. J. H. Simons, I. Graham, "30 things that go wrong in object modeling with UML 1.3*", In Behavioural Specifications of Businesses and Systems. Kluwer Academic Publishers,* 1999, pp. 237-257.

[36] P. Shoval, and J. Kabeli, "FOOM: Functional- and object-oriented analysis and design of information systems: An integrated methodology", *Journal of Database Management 12 (1)*, 2001, pp.15-25.

[37] J. Sweller, "Cognitive load During Problem Solving: Effects on learning", *Cognitive Science 12*, 1988, pp. 257-285.

[38] Y. Wand, and R. Weber, "On the ontological expressiveness of information systems analysis and design grammars", *Journal of Information Systems 3 (4)*, 1993, pp. 217-237.

[39] Walden, K., and J. Nerson, *Seamless Object-Oriented Software Architecture*, Prentice-Hall, Englewood Cliffs, NJ., 1995.

[40] Wirfs-Brock, R., and A. McKean, *Object Design: Roles, Responsibilities and Collaborations*, Addison-Wesley, Reading, Mass, 2002.

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