Evaluating UML: A Practical Application of a Framework for the Understanding of Quality in Requirements Specifications and Conceptual Modeling

John Krogstie
SINTEF Telecom and Informatics and IDI, NTNU
Forskningveien 1
N-0314 Oslo, Norway
E-mail: John.Krogstie@informatics.sintef.no

ABSTRACT

Many researchers have evaluated different parts of UML and have come up with suggestions for improvements to different parts of the language. This paper looks at UML (version 1.3) as a whole, and contains an overview evaluation of UML as a basis for creating models of high quality.
The evaluation is done using a general framework for understanding quality of models and modeling languages in the information systems field. The evaluation is based on both practical experiences and evaluations of UML made by others.
Based on the evaluation, we conclude that, although being an improvement over it is predecessors, UML still has many limitations and deficiencies.

1. INTRODUCTION

According to (Booch, Rumbaugh, & Jacobson, 1999), developing a model for an industrial strength software system before its construction is regarded increasingly as a necessary activity in information systems development

Modeling has been a cornerstone in many traditional software development methodologies for decades. The use of object-oriented modeling in analysis and design started to become popular in the late eighties, producing a large number of different languages and approaches. Lately, UML has taken a leading position in this area, partly through the standardization of the language within the Object Management Group (OMG).
In this paper, we give an assessment of UML (version 1.3) highlighting both the positive aspects and the areas where improvement is needed. We will first present the evaluation framework. We will then evaluate the language quality of UML. In another paper (Krogstie, 2000), we have also evaluated how this in combination with the modeling techniques found in one UML-tool, Rational Rose, can support the development of models of high quality.

2. BACKGROUND ON THE EVALUATION FRAMEWORK

Most existing UML-evaluations focus narrowly on language quality, either by:

- Evaluating UML relative to an existing approach e.g. (Henderson-Sellars, 1998; Paige & Ostroff, 1999)).
- Looking upon detailed aspects of the language and present improvements for these areas e.g. (Hitz & Kappel, 1998).
- Using a framework for assessing different aspects of language quality such as expressiveness in a certain context (Hommes & van Reijswoud, 1999; Prasse, 1998).

Even those using a general evaluation framework look upon the language quality features as the primary goal to achieve. Contrary to this, Krogstie, Sindre, and Lindland (Krogstie, Lindland, & Sindre 1995; Krogstie & Sølvberg, 2000) have developed a framework for discussing the quality of models in general. The framework;

- distinguishes between quality goals and means to achieve these goals. Language quality goals are one type of means. Even if it can be argued from both activity theory and decision theory that the interrelationships between goals and means are being determined through a situated preference function of the modeler, we have found that most modeling techniques in practice primarily contribute to a specific model quality goal.
- is closely linked to linguistic and semiotic theory.
- is based on a constructivistic world-view, recognizing that models are usually created as part of a dialogue between the participants involved in modeling.

Further details on the framework can be found in (Carlsen, Krogstie, Sølvberg, & Lindland, 1997; Krogstie, 1999b; Krogstie & Sølvberg, 2000) where several modeling approaches including OMT and approaches for flexible workflow modeling have been evaluated. What one is able to evaluate using the framework is the *potential* of a modeling approach to support the creation of models of high quality. Used in this way we only utilize parts of the framework as will be illustrated below. How the framework can be specialized for requirements specification models is discussed in (Krogstie, 1999a).

The main concepts of the framework and their relationships are shown in Fig. 1 and are explained below. Quality has been defined referring to the correspondence between statements belonging to the following sets:

- L, the language extension, i.e. the set of all statements that are possible to make according to the graphemes, vocabulary, syntax and structure of the language.
- D, the domain, i.e. the set of all statements which can be stated about the situation at hand.
- M, the externalized model, i.e. the set of all statements in someone's model of part of the perceived reality written in a language.
- K, the relevant explicit knowledge of the audience.
- I, the social actor interpretation, i.e. the set of statements that the audience perceive that the externalized model contain.
- T, the technical actor interpretation, i.e. the model as interpreted by tools.

The main quality types are indicated by solid lines between the sets, and are described briefly below.

- Physical quality: There are two basic quality means on the physical level
  1. Externalization, that the explicit knowledge of some person has been externalized in the model by the use of a modeling language
  2. Internalizeability, that the externalized model is persistent and available, enabling the other persons involved to make sense of it.
- Empirical quality deals with error frequencies when a model is read or written by different users, coding, and ergonomics of computer-human interaction for modeling tools.
- Syntactic quality is the correspondence between the model and the language extension of the language in which the model is written.
- Semantic quality is the correspondence between the model and the domain. The framework contains two semantic goals:

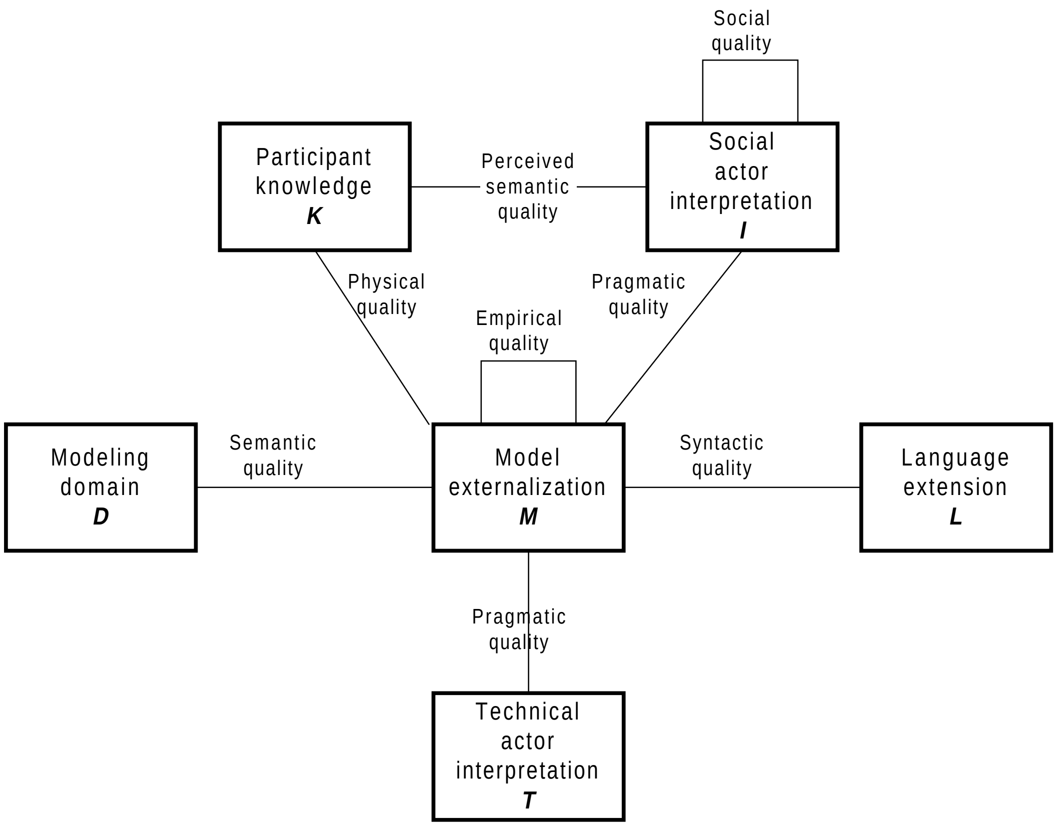

Fig. 1. Framework for discussing the quality of models

- Validity, which means that all statements made in the model are correct and relevant to the problem
- Completeness, which means that the model contains all the statements that are correct and relevant about the domain.

These goals are made more applicable by introducing the notion of feasibility.

- Perceived semantic quality is the similar correspondence between the participants' interpretation of a model and his or her current explicit knowledge.
- Pragmatic quality is the correspondence between the model and the audience's interpretation of it.
- Social quality: The defined goal is agreement among participants' interpretations.

Language quality relates the modeling languages used to the other sets. It is distinguished between two types of criteria:

- Criteria for the underlying (conceptual) basis of the language (i.e. what is represented in the metamodel).
- Criteria for the external (visual) representation of the language (i.e. the notation).

Five quality areas for language quality are identified, with aspects related both to the metamodel and the notation.

2.1 Domain appropriateness

Ideally, the conceptual basis must be powerful enough to express anything in the domain, i.e. not having construct deficit (Wand & Weber, 1993). On the other hand, you should *not* be able to express things that are not in the domain; i.e. what is termed construct excess (Wand & Weber, 1993). If one is looking for the suitability of modeling languages across a number of different, potentially unknown domains, one usually use the term expressiveness.

There are an infinite number of statements that we might want to make, and these have to be dealt with through a limited number of phenomena classes to support comprehensibility appropriateness, as will be discussed below. This means that

- The phenomena must be general rather than specialized.
- The phenomena must be composable, which means that we can group and connect related statements in a natural way. When only domain appropriateness is concerned, it is an advantage if all thinkable combinations are allowed.
- The language must be flexible in precision:

The only requirement to the external representation is that it does not destroy the underlying basis.

One approach to evaluating domain appropriateness, which will be used here, is to look at how the modeling perspectives found useful for the relevant modeling tasks are covered. Seven general modeling perspectives have been identified (Krogstie & Sølvberg, 2000). Structural, functional, behavioral, rule-oriented, object-oriented, language-action-oriented, and role and actor-oriented. More detailed evaluations of languages within these perspectives can be based on evaluation frameworks such as (Embley, Jackson, & Woodfield, 1995; Iivari, 1995). Another approach is to base an evaluation on an ontological theory, see e.g. (Opdahl, Henderson-Sellers, & Barbier, 1999). Domain appropriateness is primarily a mean to achieve physical quality, and through this, to achieve semantic quality.

2.2 Participant language knowledge appropriateness

This area relates the participant knowledge to the language. The conceptual basis should correspond as much as possible to the way individuals perceive reality. This will differ from person to person according to their previous experience, and thus will initially be directly dependent on the participants in a modeling effort. On the other hand the knowledge of the participants is not static, i.e. it is possible to educate persons in the use of a specific language. In that case, one should base the language on experiences with languages for the relevant types of modeling, and languages that have been used successfully earlier in similar tasks. Participant language knowledge appropriateness is primarily a mean to achieve physical and pragmatic quality.

2.3 Knowledge externalizability appropriateness

This area relates the language to the participant knowledge. The goal is that there are no statements in the explicit knowledge of the participant that cannot be expressed in the language. Since this is highly dependent on the participants, we do not look into this aspect of language quality in the paper. Knowledge externalizeability appropriateness is primarily a mean to achieve physical quality.

2.4 Comprehensibility appropriateness

This area relates the language to the social actor interpretation. For the conceptual basis we have:

- The phenomena of the language should be easily distinguishable from each other. (Vs. construct redundancy (Wand & Weber, 1993)).
- The number of phenomena should be reasonable. If the number has to be large, the phenomena should be organized hierarchically, making it possible to approach the framework at different levels of abstraction or from different perspectives.
- The use of phenomena should be uniform throughout the whole set of statements that can be expressed within the language. Using the same construct for different

phenomena or different constructs for the same phenomena depending on the context will tend to make the language confusing (vs. construct overloading (Wand & Weber, 1993)).
- The language must be flexible in the level of detail. Statements must be easily extendible with other statements providing more details. At the same time, details must be easily hidden.

As for the external representation, the following aspects are important:
- Symbol discrimination should be easy.
- It should be easy to distinguish which of the symbols in a model any graphical mark is part of.
- The use of symbols should be uniform; i.e. a symbol should not represent one phenomenon in one context and another one in a different context. Neither should different symbols be used for the same phenomenon in different contexts if there is no good reason for this.
- One should strive for symbolic simplicity.
- The use of emphasis in the notation should be in accordance with the relative importance of the statements in the language.
- Composition of symbols can be made in an aesthetically pleasing way. A negated example of this is a process modeling language which mandates that all inflow enters the same side of the process symbol, which can lead to the resulting model having unnecessarily many crossing or long lines.

Comprehensibility appropriateness is primarily a mean to achieve empirical and pragmatic quality.

2.5 Technical actor interpretation appropriateness
This area relates the language to the technical actor interpretations. For the technical actors, it is especially important that the language lend itself to automatic reasoning. This requires formality (i.e. both formal syntax and semantics. The formal semantics can be operational, logical, or both), but formality is not sufficient, since the reasoning must also be efficient to be of practical use. This is covered by what we term analyzability (to exploit the mathematical semantics) and executability (to exploit the operational semantics). The power of a formal semantics lies in three aspects (Wieringa, 1998):
1. The process of making a (more) formal model may reveal errors and ambiguities at an early stage.
2. Formal and even automated proofs may be available.
3. The remaining (or unprovable) rules may be translated into executable constraints in some imperative language.

Different aspects of technical actor interpretation appropriateness are means to achieve syntactic, semantic and pragmatic quality (utilizing formal syntax, mathematical semantics, and operational semantics respectively).

3. EVALUATION

We first position UML in relation to the sets of the quality framework.
Domain: According to (OMG, 1999), UML is a language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems. In other words, UML is meant to be used in analysis of business and information, requirements specification and design. UML is

meant to support the modeling of (object-oriented) transaction systems, real-time and safety critical systems. For those areas being directly related to the modeling domain, we will differentiate the discussion according to the different domains.
Language: We have based the evaluation on UML (version 1.3) (OMG, 1999). We have not looked at OCL nor the defined profiles, but concentrated on the core language. This version has been characterized as the first mature release of UML (Kobryn, 1999).
The sets' Knowledge', 'Model', and 'Interpretation' must be judged from case to case in the practical application of the modeling language and tools. Also when it comes to weighting the different criteria against each other, this must be done in the light of the specific modeling task, such as has been done e.g. by Østbø (2000).

When using the quality framework to do an evaluation like this, we should have the following in mind:

- It is possible to make good models in a poor modeling language.
- It is possible to make poor models in a comparatively good modeling language.
- You will always find some deficiencies in any language. On the other hand, it is useful to know the weak spots to avoid the related potential problems. Such deficiencies should in general be addressed with the use of modeling techniques, and an overall methodology. None of these areas are addressed in UML.

The primary aim of this evaluation is to help people using UML to recognize the existing weaknesses. On a longer term, it is also meant to give input to areas that should be addressed in later versions of the standard.

The basis for the evaluation is in addition to the framework

- UML 1.3 language specification (OMG, 1999).
- Practical experience using UML both by the author and by other people interviewed by the author in both an industrial and an academic setting. This includes;
    - Experiences by consultants in the use of UML for analysis and requirements specification of applications within the financial services sector. The consultants were interviewed after the project was finished.
    - Experiences from the design of an object-oriented groupware system. Developers of this system were interviewed during the development of the third release of the system, parts of UML being used also on earlier releases.
    - Experiences from the use of UML for the documentation of generic frameworks (Østbø, 2000).
- Additional evaluations found in the literature (Bergner, Rausch, & Sihling, 1998; Bézivin & Muller, 1998; Castellani, 1999; France & Rumpe, 1999; Ovum, 1998; Prasse, 1998).

Due to the limitation on the length of a paper of this kind and the breadth of the evaluation, we will only have room for presenting the major results. See e.g. (Østbø, 2000) for a more detailed description of using the framework for evaluating UML.
The UML semantics (based on the 'metamodel') is the basis for evaluating the conceptual basis, whereas the notation guide is used as a basis for the evaluation of the external representation.

3.1 Domain appropriateness

Looking briefly on the coverage of the seven modeling-perspectives, we find:

- The object-oriented perspective is primarily relevant during analysis of information and design (Davis, 1995). UML has (not surprisingly) a very good support for modeling according to an object-oriented perspective, although with a limited modeling capability regarding responsibilities.

- The structural perspective is primarily relevant during analysis of information and design. This perspective is also well supported, although not as well as in languages made specifically for this purpose (Halpin & Bloesch, 1999). Traditional abstraction-mechanisms such as aggregation, classification, and generalization are provided, but other modeling languages such as OML (Henderson-Sellars, 1998) and different languages for semantic data modeling have a more precise and expressive representation of these abstraction mechanisms.
- The behavioral perspective can be useful in all domains, but is particularly used within design. UML supports the behavioral perspective using Statecharts, but does not support the refinement of Statecharts in a satisfactory way (Hitz & Kappel, 1998).
- The functional (process) perspective is supported on a high level through Use Case modeling (a.k.a. 0-level DFD), for requirements modeling and activity diagrams, which can be used to show control flow and the production of data or objects in a process flow. This is primarily useful for design. Many have also attempted using activity diagrams for business models. Hommes & van Reijswoud (1999) argues that the modeling concepts in the business process domain are not easily mapped to UML. Software processes can be modeled using the interaction diagrams.
- The actor-role perspective can be relevant both in analysis of business and design. It is partly covered using the Collaboration Diagrams. Using roles in Sequence Diagrams or ‘swimlanes’ in Activity Diagrams, we also get a role-oriented view, but there is no intuitive way to represent organizational and group-structures and relationships in UML, something which is very useful for the analysis of businesses.
- Single rules can be used in all domains. It is possible to formulate single static rules in OCL. There are some general problems with constraints expressed in an OO modeling framework such as the representation of rules that relates several objects of different classes (Høydalsvik & Sindre, 1993). Temporal and deontic constraints are hard to express. The same problem applies to non-functional requirements, such as performance, reliability, or security requirements. There are also technical problems with visibility of e.g. private attributes used in constraints. There is no support for goal-hierarchies (Mylopoulos, Chung, & Tu, 1999), a technique primarily used in analysis of business and requirements specification.
- The language-action perspective, which is most useful for the analysis of businesses, is not supported.

A metamodel of UML is defined (using UML), and there exist extension mechanisms to make the language more applicable in specific domains. UML contains only lightweight extension mechanisms, such as stereotypes, constraints and tagged values (compared to metaclasses, which is regarded as a heavyweight extension mechanism). On the other hand, these mechanisms are better than what is found in most other modeling languages.

UML contains several language elements that are first useful during design. These language mechanisms should not be used in analysis and requirements specification, even in areas where the transition from analysis to design is ‘seamless’. (There are quite some evidence that especially for business systems, this transition is far from seamless even when using object-oriented modeling in both domains (Davis, 1995; Høydalsvik & Sindre, 1993; Lauesen, 1998)). Proper guidelines for avoiding this are not consistently provided, and there is no support for avoiding using analysis and design concepts in the same model. It is generally believed that a good method should help you to keep information about what a system should do, separated from how that functionality should be implemented in a specific implementation environment. The connections between such models should also be possible to express (Ovum, 1998). UML gives limited support in this regard.

Although comprehensive, UML can not be used to specify complete applications. The limitations are both that it does not provide an action language (Mellor, Tockey, Arthaud, & LeBlanc, 1998) to support the analysis of complete specifications, and that it lacks some constructs for e.g. architecture (Hilliard, 1999), user-interfaces (Kovacevic, 1998) and hypermedia (Baumeister, Koch, & Mandel, 1998) to support more complete code-generation.

3.2 Comprehensibility appropriateness and participantlanguage knowledge appropriateness

Some main observations on this area:

- Several inconsistencies exist, partly because of the use of meta-modelling which has resulted in the inheritance of sometimes meaningless (or at least undefined) properties. As an example, a Constraint is a model element that constraints other model-elements (including potentially a constraint), and any model element (including a constraint) might have a state model. Circularities in the definitions are also found (Castellani, 1999).
- For those being familiar with the main OO-modeling concepts and main OO modeling-languages, the core of UML should not represent a too steep learning curve. On the other hand, UML has a total of 233 different concept (Castellani, 1999) and it is not surprising that some redundancy and overlap is witnessed. Some examples:
  - The concepts Signal and Operation call are almost identical.
  - How to differentiate between the use of types and the use of classes is poorly defined.
  - Guards, Preconditions, Postconditions, Constraints, Multiplicity, and Invariants are all different types of rules. On the other hand, these terms might be so well established that it causes few problems in practice.
- Symbol differentiation:
  - Both classes and objects are shown using rectangles.
  - Use-cases, States in Statecharts and Activities in Activity Diagrams are all shaped more or less like an ellipse.
  - The same symbol is used for choice in Activity diagram, aggregation, and n-ary associations.

It is difficult to avoid some overlaps of this kind in a complex language like UML.

- Uniform use of symbols. The predecessor of the structural model in UML, OMT had quite a few deficiencies in this area, some of which have been addressed:
  - Contrary to OMT, there is in UML a uniform way of showing cardinality (multiplicity).
  - Associations are shown in two ways, compared to the four ways of showing associations in OMT.
  - Different symbols are used for a role if it is external to the system (pin-man) or internal (rectangle).
  - An interface is shown in two different ways (as a circle, or as a class-symbol).

Many of these deficiencies are relatively unproblematic, since different aspects are focussed on in the different models. Almost all CS and IS-degrees now include a course or more where UML is lectured and used. On the other hand, having too many issues like this makes it more difficult to learn the language and comprehend models made using the language. This is specifically important in models used to analyze business and information, and requirements specification, which are meant to be comprehended by many people with different backgrounds.

- In the structural model, emphasis is set on classes and objects through the size of the symbols, which is sensible. Most of the usage of filled symbols found in OMT is removed, with the exception of full aggregation, which is an intuitive use of this effect. That the class-symbols get different size and shape dependant on the number of attributes and operations that are defined makes these potentially visually complex. This is outweighted by the fact that the diagrams have much fewer nodes than if these concepts should be represented separately as done in many other languages for structural modeling. The same positive remark can be made on the onion-notation inherited through adopting the use of Statecharts. The possibility of grouping classes/objects in packages and composite classes and objects is also potentially a positive aspect in this connection, which is an improvement over its predecessors.
- Some symbols are unnecessarily complex, e.g. the components in Components Diagrams. There are historical reasons for this, to make it easier for people used to the Booch notation to recognize these symbols.

3.3 Technical actor interpretation appropriateness

The UML-syntax is rigorously defined, and the language is described through a metamodel made in the structural model of UML with accompanying OCL-rules and natural language description. Using UML to model UML means that some of the definitions are circular, and this leaves UML (formally speaking) undefined. This would be unproblematic if most practitioners already understood the meaning of the concepts (classes, inheritance, and associations) that are involved. To illustrate that this can be problematic, we can point to that the concept 'inheritance' is used in three different ways in the literature. UML supports one of these. There are significant improvements in the metamodel of UML 1.3, although it falls short of a strict metamodeling approach (Kobryn, 1999). UML neither has a formal mathematical nor an operational semantics, and there is no support for the generation of design models from analysis models, or implementation models from design models. OCL also give the potential for some of the analysis and consistency checking that a formally defined mathematical semantics would. On the other hand, it is unclear what tool support will be developed. Other groups have proposed extending Z to provide a formal semantics to UML (Evans, 1998). A formal (operational) action language would also be useful to be able to support a wider repertoire of modeling techniques (Mellor, Tockey, Arthaud, & LeBlanc, 1998).

4. CONCLUSION AND FURTHER WORK

Many improvements can be found in UML compared to its predecessors. Due to its strong support, UML is probably the best general modeling language to adopt as a basis for object-oriented development if one is not already using another language with good tool support that one is satisfied with. Another positive aspect is the inclusion of a process-oriented language for requirements specifications (Hitz & Kappel, 1998).

Thanks to this, the very first step in object-oriented development does not encompass finding objects in the problem domain, but the identification of the system functionality as required by the users. Most of the accidental problems such as inconsistencies in the language-descriptions found in earlier version of UML seem to be addressed in UML 1.3, but there are still major concerns:

A major lesson from the introduction of CASE tools based on structured approaches can be summarized with ‘No methodology – no hope’ (Parkinson, 1990). This is a major point also made by OVUM (1998). Even if it has not been possible to agree on a standard process, outline process guidelines need to be included – even if the best that can be done is to describe a number of alternatives. This would have helped users to understand UML. Particularly problematic is the analysis/design confusion. As discussed by among other Davis (1995) there are fundamental differences between the models related to analysis, design, and requirement specification. What our investigation has also illustrated is that although there is a perceived need to extend the expressiveness and formality of the language, the language has several weaknesses regarding comprehensibility appropriateness, and is already looked upon as difficult to comprehend, with a steep learning curve (France & Rumpe, 1999). It has been suggested that a successful response to these challenges will require the OMG adopt a sculpting approach (where “less is more”), rather than a mudpacking approach. It is then an open question what should be kept in the core of UML, and what should be kept as part of internally consistent, but separate profiles or in standard model libraries. In our opinion, UML currently has its main strengths in the creation of design models for traditional object-oriented systems, and this domain (with the extension and necessary elaboration of use cases to be able to say something sensible about requirements) should probably define the scope for the core language. Profiles should then be developed just as rigorously for extensions using a full meta-modeling approach, and tools should enable the use of those extra profiles than are deemed necessary for the modeling activity at hand.

This work will be followed up and updated as new versions of UML are made available. We will in the future also look upon how different UML-based methodologies help in addressing the problematic areas still found in UML.

ACKNOWLEDGEMENTS

We would like to thank Reidar Conradi for input to an earlier version of this work.

REFERENCES

Baumeister, H., Koch, N., & Mandel, L. (1999). Towards a UML Extension for Hypermedia Design. In (France & Rumpe, 1999) (pp. 614-629).

Bergner, K., Rausch, A., & Sihling, M. (1998). Critical Look upon UML 1.0. In Schader, M. and Korthaus, A. (Eds.), *The unified modeling language – Technical aspects and applications*. Physica-Verlag, Heidelberg.

Bézivin, J. & Muller, P-A (Eds.) (1998). *UML’98- Beyond the notation. June 3-4* Mulhouse, France. Springer-Verlag.

Booch, G., Rumbaugh, J. & Jacobson, I (1999). *The Unified Modeling Language: User Guide* Addison-Wesley.

Carlsen, S., Krogstie, J., Sølvberg, A., & Lindland, O.I.(1997). Evaluating Flexible Workflow Systems. In J. F. Nunamaker, & R. H. Sprague (Eds.), *Proceedings of the Thirtieth Annual Hawaii International Conference on System Sciences (HICCS'97). Volume II Information Systems- Collaboration Systems and Technology, January,* (pp. 230-239).

Castellani, X. (1999). Overview of Models Defined with Charts of Concepts. In E. Falkenberg, K. Lyytinen, & A. Verrijn-Stuart (Eds*.), Proceedings of the IFIP8.1 working conference on Information Systems Concepts (ISCO4); An Integrated Discipline Emerging September 20-22* , Leiden, The Netherlands, (pp. 235-256).

Davis, A. (1995). Object-oriented requirements to object-oriented design: An easy transition*? Journal of Systems and Software 30*(1/2) July/August, 151-159.

Embley, D. W., Jackson, R. B., &Woodfield, S. N. (1995). OO System Analysis: Is it or Isn't it? *IEEE Software 12* (3) July 19-33.

Evans, A., France, R., Lano, K., & Rumpe, B.(1998) Developing the UML as a Formal Modeling Language. In (Bézivin & Muller, 1998) (pp. 346-348).

France, R. & Rumpe, B. (Eds.) (1999). *UML'99 – The Unified Modeling Language – Beyond the standard*. Springer Verlag.

Halpin, T. & Bloesch, A. (1999) A. Data Modeling in UML and ORM: a comparison. *Journal of Database Management 10*(4) oct-dec, 4-13.

Henderson-Sellars, B. (1998). OML – Proposals to Enhance UML. In (Bézivin & Muller, 1998), (pp. 349-364).

Hilliard, R. (1999). Using the UML for Architectural Description. In (France & Rumpe, 1999). (pp. 32-48).

Hitz, M. & Kappel, G. (1998). Developing with UML – Some Pitfalls and Workarounds. In (Bézivin & Muller, 1998), (pp. 9-20).

Hommes, B-J. & van Reijswoud, V. (1999). The quality of Business Process Modeling Techniques. In E. Falkenberg, K. Lyytinen, & A. Verrijn-Stuart (Eds*.), Proceedings of the IFIP8.1 working conference on Information Systems Concepts (ISCO4); An Integrated Discipline Emerging September 20-22* , Leiden, The Netherlands, (pp. 117-126).

Høydalsvik, G. M. & Sindre, G. (1993). On the Purpose of Object-Oriented Analysis. In A. Paepcke (Ed.), *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'93) Septembe*r (pp. 240-255) ACM Press.

Iivari, J. (1995). Object-orientation as structural, functional, and behavioral modeling: A comparison of six methods for object-oriented analysis. *Information and Software Technology 37* (3) , 155-163.

Kobryn, C. (1999). UML 2001. A standardization Odyssey. *Communication of the ACM 42* (10) October, 29-37.

Kovacevic, S.(1998). UML and User Interface Modeling, in (Bézivin & Muller, 1998), (pp. 253-266).

Krogstie, J., Lindland, O.I., & Sindre, G. (1995). Defining Quality Aspects for Conceptual Models. In E. D. Falkenberg, W. Hesse, & A. Olive (Eds.). *Proceedings of the IFIP8.1 working conference on Information Systems Concepts (ISCO3); Towards a consolidation of views, March 28-30* (pp. 216-231), Marburg, Germany.

Krogstie, J. (1999a). Pulling together the understanding of quality in requirements specifications and modeling. *In Proceedings of Norsk Informatikkonferanse (NIK'99) November 15-17* (pp. 315-326), Trondheim, Norway.

Krogstie, J. (1999b). Using Quality Function Deployment in Software Requirements Specification. In A. L. Opdahl, K. Pohl, & E. Dubois. *Proceedings of the Fifth International Workshop on Requirements Engineering: Foundations for Software Quality (REFSQ'99), June 14-15,* (pp. 171-185), Heidelberg, Germany.

Krogstie, J. & Sølvberg, A. (2000) *Information Systems Engineering : Conceptual Modeling in a Quality Perspective*, Draft of Book, Information Systems Groups, NTNU, Trondheim, Norway.

Krogstie, J (2000). Using a Semiotic Framework to Evaluate UML for the Development of Models of High Quality. In Siau, K. and Halpin, T. (Eds.) *Unified Modeling Language: Systems Analysis, Design, and Development Issues* IDEA group

Lauesen, S. (1998). Real-life Object-oriented Systems. *IEEE Software 15*(2), 76-83.

Mellor, S. J., Tockey, S. R., Arthaud, P., & LeBlanc, P. (1998). An Action Language for UML. In (Bézivin & Muller, 1998), (pp. 307-318).

Mylopoulos, J., Chung, L., & Tu, E. (1999). From Object-oriented to Goal-oriented Requirements Analysis. *Communications of the ACM. 42* (1), January , 31-37.

OMG (1999). UML Version 1.3

Opdahl, A., Henderson-Sellers, B., & Barbier, F. (1999). An Ontological Evaluation of the OML Metamodel. In E. Falkenberg, K. Lyytinen, & A. Verrijn-Stuart (Eds*.), Proceedings of the IFIP8.1 working conference on Information Systems Concepts (ISCO4); An Integrated Discipline Emerging September 20-22* , Leiden, , (pp. 217-232) The Netherlands.

Ovum (1998). Ovum evaluates: CASE products. Guide to UML .

Paige, R. F. & Ostroff, J. S. (1999). A comparison of the Business Object Notation and the Unified Modeling Language. In (France & Rumpe, 1999) (pp. 67-82).

Parkinson, J. (1990). Making CASE Work. In K. Spurr and P. Layzell (Eds.) *CASE on Trial* (pp. 213-242). John Wiley & Sons.

Prasse, M. (1998). Evaluation of Object-oriented Modeling Languages. A Comparison between OML and UML. In M. Schader, & A. Korthaus (Eds*.): The unified modeling language – Technical aspects and applications*. (pp. 58-78). Physica-Verlag, Heidelberg.

Wieringa, R. (1998). A Survey of Structured and Object-Oriented Software Specification Methods and Techniques. *ACM Computing Surveys 30* (4) December, 459-527.

Østbø, M. (2000). *Anvendelse av UML til dokumentering av generiske systemer (In Norwegian).* Unpublished master thesis Høgskolen i Stavanger, Norway, 20 Juni.