UML: A Survey Focused On Use Case Modeling

Georgios Chalkiadakis

Department of Computer Science, University of Toronto

Abstract

The Unified Modeling Language (UML) has been in the past years established as an industry standard for object - oriented modeling. Its ability, however, to serve as an efficient requirements specification tool is highly debated. In this paper, an attempt is made to present an overview of the language focused on use case modeling, which is the language's technique for capturing the system requirements. The usefulness, drawbacks and possible solutions or alternatives to the deficiencies of UML use case modeling are discussed. In addition, the question of whether or not UML should be thought of (and therefore criticized) as a semi-formal requirements specification language is addressed.

1 Introduction

Object-oriented modeling has been established as the dominant modern modeling method[1] used for the analysis and design of a system, replacing the structured analysis modeling techniques. The emergence of Object-Oriented Analysis and Design (OA&D) techniques has been the "natural" consequence of the central assumptions of data-oriented analysis and design methods [8]:

- Organizational data provides quite a stable foundation for modeling a system, since as the system evolves data tends to remain unchanged while procedures tend to change.
- Data should be viewed as an organizational resource independent of the systems that currently process the data.

These assumptions gave ground to the argument that a data/object - oriented model of a system would probably be easier to maintain (since it would be harder to become out-of-date) than a structured analysis model. In addition, they led to the identification of some main principles of object-orientation which distinguish it from other modeling approaches [21]:

- Encapsulation of attributes, operations and services within objects.
- Classification of object-abstractions.
- Inheritance of common attributes between objects.

These basic principles are present in all the object-oriented methods, despite their differences. In general, object-oriented modeling assumes the identification of specific objects and classes of objects, along with relationships between them [1, 3, 4, 6, 14, 25, 28, 30]. Object-oriented modeling promotes modularity, abstraction and re-use [8].

The Unified Modeling Language (UML) is an object-oriented modeling language that has

[1] What we refer to as "methods" in this paper are often called methodologies. However, in an academic context, the word "methodology" should properly be used for the study and comparison of methods.

in the past few years gained significant industry support and has been adopted by the Object Management Group (OMG) as a standard (November 17, 1997). According to what is stated in its specification: "(UML) provides system architects working on object analysis and design with one consistent language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling" [22]. The technique that is employed by UML in order to facilitate the capturing of the functional requirements of a system is use case modeling [17, 20, 29]. Use-case modeling will be presented in considerable detail in the remaining sections of this paper.

The purpose of this paper is to address UML from a requirements engineering point of view, focusing, thus, on UML use case modeling, its usefulness, as well as its deficiencies and problems. Moreover, suggested corrections for the deficiencies and solutions to the problems are presented. Furthermore, a presentation of the most common critiques against UML use case modeling is given, along with some answers to those critiques. A stand is taken that UML is not actually a requirements specification language and therefore should not be heavily criticized for its existing deficiencies with regard to the capturing of requirements; the critiques should be constructive, taking into consideration the great degree of UML's adoption by the industry.

The rest of the paper is structured as follows: Section 2 draws the historical background of the emergence of UML and presents a summary of UML. Section 3 constitutes the main section of this paper, discussing use case modeling, critiques and solutions. Section 4 addresses in brief the rather "philosophical" question of whether theory and research are or should be in antagonistic relations with practice and industry. Finally, Section 5 concludes this paper.

2 UML: History and Brief Summary

This section will provide the reader with a brief historical background on the adoption of the object-oriented modeling paradigm and the emergence of UML. It will also give a brief overview of UML's basic goals and concepts, as well as address the question of whether UML is primarily a "notation" or a "method"[2].

2.1 The Adoption of the Object-Oriented Modeling Paradigm and the Emergence of UML

The adoption of OA&D methods by the industry was not an easy process. Since the emergence of the object-orientation paradigm in the late 80's and until the mid-90's, a great number of object-oriented methods appeared. The many published comparisons of these various methods tended to highlight their differences rather than their commonalities. Every method appeared to be based on a different object model [11]. There were constant debates on which method should be considered the "best"; however, such a choice of "the perfect object-oriented method" was impossible, given the fact that every method had strengths and weaknesses [23]. In this way, the adoption by the industry was problematic: it is a fact that the adoption of a new technology is embedding a high risk which would be difficult for a company to undertake unless it has a clear understanding of the benefits to be gained. Clearly, the "methodologies' wars"[11] that were taking place at the time made it difficult for the industry to evaluate and adopt these methods.

However, in practice the differences between the object-oriented methods were not that big. Many experienced system architects managed to effectively combine ideas and solutions from different methods, and this marked the beginning of attempts to reach convergence. Several method providers began to seek ways to co-operate. [11, 23]

This was the situation when the Unified Modeling Language was conceived by the Rational Software Corporation, as an attempt to unify the Booch [3, 4], Object Modeling Technique (OMT) [25] and Object-Oriented Soft-

[2]By "modeling method", we mean a collection of techniques and guidelines, that in combination with a notation construct a "roadmap" that can be used for the development of a system's model - through analysis and design.

ware Engineering (OOSE) [14] methods. UML is the union of these languages, and attempts to include additional expressiveness to handle modeling problems that these methods did not fully address. UML should provide improved expressiveness and holistic integrity, in respect to other visual modeling languages [22]. As has been mentioned already, it is an OMG standard since 1997.

The adoption of UML as an OMG standard has led to the widespread use of the language in industry. The existence of an industry standard has lowered the cost of training and retooling when changing between projects or across organizations, and has led to the development of new tools to facilitate OA&D.

2.2 Goals of UML

The primary design goals of UML, according to its specification, are as follows [22]:

- To provide the users with a ready-to-use, expressive visual modeling language to develop and exchange meaningful models.
- To furnish extensibility and specialization mechanisms to extend the core concepts, as new needs are discovered and for specific domains.
- To support specifications that are independent of particular programming languages and development processes.
- To provide a formal basis for understanding the modeling language.
- To encourage the growth of the object tools market.
- To support higher-level development concepts such as components, collaborations, frameworks and patterns.
- To integrate best practices in the industry.

2.3 Summary of Basic UML Concepts

UML is an evolution from Booch, OMT, OOSE, other object-oriented methods, and many other sources. These various sources incorporated many different elements from many authors, including non-OO influences. The UML notation is a melding of graphical syntax from various sources, with a number of symbols removed and a few new symbols added [22].

The basic tools employed by the UML notation are the following [13, 22]: use case diagrams, class diagrams, object diagrams, component diagrams, deployment diagrams, activity diagrams, statechart diagrams, collaboration diagrams and sequence diagrams. Most of these diagrams were already present in several modeling languages, before becoming a part of the UML notation.

Other basic concepts of UML, which were for the first time brought together by this language, are [22]:

- extensibility mechanisms (stereotypes, tagged values, and constraints),
- threads and processes,
- distribution and concurrency (e.g., for modeling ActiveX/DCOM and CORBA),
- patterns /collaborations (collaborations are first-class modeling entities and often form the basis of patterns),
- refinement (to handle relationships between levels of abstraction),
- interfaces and components, and
- a constraint language (the Object Constraint Language, which is used to specify the semantics and is provided as a language for expressions during modeling)

2.4 UML: A Method or a Notation

The UML specification contains the language's notation description, as well as the formal definition of a common OA&D metamodel to represent the semantics of OA&D models and IDL specifications for mechanisms for model interchange between OA&D tools [22].

Nevertheless, a question might arise with regard to whether UML is an OA&D method or primarily a notation to enhance OA&D. UML is a modeling language that has indeed tried to specify a notation to capture the best out of many design methods, by unifying existing notation tools provided by these methods, and trying "to codify the best practices...encountered in successful object-oriented projects worldwide" [2]. It does not actually provide a fixed ordering of phases for OA&D. Probably Booch's view of what a method should be is the underlying philosophy of UML:

the architects should work iteratively and incrementally, augmenting formal diagrams with informal techniques as appropriate to the problem at hand [3].

However, attempts are underway to describe object-oriented methods or methodological frameworks that would employ UML:

- The Rational Unified Process[3] is an attempt to define a software engineering process that would guide the members of a software engineering team through each phase of the development project [27].
- The Object-oriented Process, Environment and Notation (OPEN) is a methodological framework created by a group of methodologists and researchers, which tries to merge existing OA&D methods [11]. The OPEN method recommends the use of a specific notation, called OML (the "Open Modeling Language"), but this notation may be substituted by UML notation, since it is viewed as being a modular component of the method [11].

In addition, it is important to mention that the UML literature proposes that the model for describing a software system should be a "4+1" architectural model. This architectural model, the construction of which can be facilitated through the use of UML notation, is made up of [19]:

- A *logical* view, which is the object model of the design;
- A *process* view, which captures the concurrency and synchronization aspects of the design;
- A *physical* view, which describes the mapping(s) of the software onto the hardware and reflects its distributed aspect;
- A *development* view, which describes the static organization of the software in its development environment;
- And, finally, the *use case* view, which illustrates the description of the architecture through the use of "use cases", or "scenarios". The scenarios are synthesized to abstract requirements.

[3] This is a process suggested by the corporation that originally conceived UML and it is related to certain products that utilise UML.

The use case view serves two main purposes: It can be used as the driver to discover the architectural elements during the design and also has a validation and illustration role after the design is complete [19].

3 Capturing User Requirements: Use Case Modeling in UML

This section will focus on use case modeling, as described in UML literature. The strengths, deficiencies and limitations of the technique will be presented, along with solutions and remarks on the existing critiques.

3.1 What use case modeling is and what use case modeling is not

The use case modeling is the technique during which use case models are produced. A use case model employs "actors" and "use cases" in order to depict a scenario of the system's use. The use case model provides a way of describing the external view of a system and its interactions with the outside world. The "actors" represent roles played by various people or other systems, while the "use cases" represent typical interactions that the actor has with the system [9]. According to the formal UML specification, the actor is "a coherent set of roles that users of use cases play when interacting with these use cases. An actor has one role for each use case with which it communicates", while a use case is "the specification of a sequence of actions, including variants, that a system (or other entity) can perform, interacting with actors of the system" [22]. Jacobson, who was the first to introduce the concept of the use case, defines a use case as the "behaviourally related sequence of transactions in a dialogue with the system" [14].

These descriptions show that a use case represents a part of the system's functionality. In fact, use case models can be used to model functionality of entities at different levels of abstraction. They can be used to model functionality of business entities, sub-systems, components or even programming language classes [29]. Thus, use case models can be used as the basic unit of estimation and deliverable [17]. They are useful tools for the illustration and validation of the system; they are central to planning the evolutionary development process, and can be used to drive system testing [14, 19].

UML employs the use case modeling technique in order to capture the functional requirements of a system [9, 17, 29]. The use cases are not intended to capture all of the systems requirements. It should be stressed that there is no intention whatsoever to capture the non-functional requirements of the system by the use of use cases.

To draw a little more on the previous point, it should be clear that the use case model is about *what* the system will do at a high level, with respect to its users. It does not intend to capture *how* the system will do anything. Neither do the use cases capture anything the actors do that does not involve the system [17].

A use case in UML may help the system's architect to identify the user's goals, but *not* the system functions. Structuring the software so that it mimics a use case is not suggested by UML, since the use cases provide just an external view of the system [9]. In addition, it is not suggested to put in a use case diagram any formal or detailed information such as a relationship between objects or a sequence of actions performed by actors. UML provides other tools (e.g. class diagrams or sequence diagrams) that may be employed to capture this kind of information. Finally, use case decomposition should be done only as long as it serves for capturing the high-level functional requirements [17].

3.2 Critiques to UML Use Case Modeling

There is little doubt that use case modeling was a considerable attempt to elicit requirements upstream of object modeling. However, there have been many arguments against the way use case modeling is employed by UML. Drawbacks of the technique have been identified in several papers; attempts to deal with the problems have also been presented in the literature. This subsection will present the problems that are mentioned in three papers [10, 26, 29] that focus on UML deficiencies, along with the answers given (when such answers are provided in these papers). At some points where solutions are not provided by the original authors, an attempt to address some of the critiques will be made. In addition, discussion on the critiques is provided, when this is deemed appropriate.

3.2.1 "The Role of Roles in Use Case Diagrams"

In their paper "The Role of Roles in Use Case Diagrams" [29], Wegmann and Genilloud identify five limitations of use cases and try to overcome these by making the roles of actors explicit. In brief, they identify the following limitations:

1. *It is not possible in UML to model, in one use case diagram, a business process specification between multiple system types and actor types. UML use case diagrams force the designer to have only one system of interest in a use case diagram.*

   The solution proposed is to consider the use cases as non-system centric, as in the case of Catalysis [7]. It is proposed that a use case is described on two levels, a declarative description level to describe the effects of a joint action on all participant objects and an operational description level to decompose the joint action. The specification's description of a use case should change accordingly, and no reference to "the system" should be present.

2. *It is not possible for a system to play different roles in different occurrences of the same use case specification, in UML use case diagrams. The source of the problem is that UML refers to actors instead of roles.*

   The solution proposed is to modify the definition of an actor, so that an actor may be identified by its role in the use case context, rather by its name. A mechanism to bind roles to actors is also suggested.

3. *It is not clear in the UML specification whether the use case modeling technique*

has provisions for representing the number of actor instances participating in a use case occurrence. An actor instance is not acknowledged as such, neither is it clear if a role is a type or an instance.

The suggested solution to this problem is that UML defines all concepts as both a concept type and a concept instance. More specifically, an actor [instance] and an actor [class], as well as a role [instance] and a role [class] should be defined.

4. *There is no way to represent constraints on when use cases may occur.*

   The answer to this is to number the occurrences of the use cases, in order to illustrate the sequence of their "execution". An alternative way to overcome this difficulty is to use state diagrams to specify execution constraint between use cases [12].

5. *There is a question on what the definition of an "actor" should be.*

   The authors propose that a completely new definition for the actor's concept is given, a definition that would specify the actor as a composition (n.b.: not a set) of roles. However, the designer should be enabled to state which entity realizes the actor's role. Other concepts such as "entity role" and "entity context" are also coined.

The authors claim that their approach adds expressiveness, consistency and simplicity to UML. The modifications they suggest allow the representation of all entities in use case diagrams. These modifications change the UML specification only in a limited degree. Nevertheless, one could not but note that the "philosophy" of the proposed approach is quite different to the way UML currently considers use case modeling. The approach presented transforms the use case diagrams in such a way that they become more similar to the class diagrams and interaction diagrams. However, this may not be actually a need in UML use case modeling (see subsection 3.1).

3.2.2 "Problems and Deficiencies of UML as a Requirements Specification Language"

In his paper "Problems and Deficiencies of UML as a Requirements Specification Language" [10], Martin Glinz investigates the suitability of UML as a semiformal requirement specification language, using a case study as an example. He identifies problems and deficiencies of UML concerning use case models and system decomposition, and discusses whether these may be overcome and how. The following problems are identified:

1. *Active objects are not allowed in UML use case diagrams, and, thus, a UML use case model cannot specify interaction requirements where the system shall initiate an interaction between the system and an external actor.*

   The author considers this an "accidental" deficiency, which can be removed with a slight modification in UML metamodel allowing active objects in use case diagrams, or with the use of the stereotype "active" for use cases.

2. *UML cannot model a "rich system context", as it does not allow interaction between actors.*

   Again, this is considered an "accidental" deficiency, which can be overcome by a local modification in the metamodel, allowing associations between actors.

3. *UML can neither express structure between use cases nor a structural hierarchy of use cases in an easy and straightforward way. Sequential, parallel or iterative relationships between use cases cannot be described in UML.*

   This deficiency is up to an extent similar to the 4th deficiency identified in [29]. It is considered an "essential" deficiency by the author, one that cannot be overcome without modifications that affect major concepts in the language. The introduction of a set of stereotypes for use cases and dependency relationships is coined, but even then the consistency between use case structures and the structure introduced by the subsystem decomposition cannot be removed.

However, it could be argued that this deficiency can be considered as such only if there is a demand for a high degree of use case decomposition. This is not always the case, and the UML literature foresees that there may be tradeoffs between the need for simplicity and the extent of use case decomposition (see subsection 3.1).

4. *UML provides no adequate means for dealing with use case interaction. Individual use cases may be modeled as state machines, but UML cannot model states being shared between use cases.*

5. *A UML use case model cannot express state-dependent system behaviour adequately. Use cases cannot be combined with a model of objects and states.*

6. *UML cannot model the behaviour of high-level system components such as subsystems. UML regards subsystems as containers of behavioural entities only, and hence disallows associating a state machine with a subsystem.*

   These three deficiencies are related and considered essential by the author, and the remedy suggested is the semantic integration of structure, functionality, behaviour and user interaction.

   Again, however, it could be argued that it is a choice of the UML creators not to deal with these problems. The use of the "4+1" architectural views is suggested instead. It is a choice not to combine notational tools that are employed in different architectural views. The combination, for example, of use cases, objects and states is not considered a need. The use of the logical view and the process view is considered the method to analyse the high-level interactions that take place in the use case view.

7. *Modeling information flow in a system consisting of subsystems is awkward in UML.*

8. *UML cannot adequately model the decomposition of a distributed system.*

9. *UML cannot model all aspects of a composite entity like a subsystem together in a single view.*

   These problems are considered fundamental by the author; this means that they cannot be treated without modifications in the foundations of UML. Such a fundamental treatment would be "abandoning the concept of a loosely coupled collection of aspect models and moving towards an integrated model with a uniform decomposition mechanism". The author in fact proceeds to suggest the ADORA (Analysis and Description of Requirements and Architecture) architecture [15] as a potential alternative to UML, though he admits that "for the practical use of a universal semiformal requirements specification language in industry, there is currently no alternative to UML". From a UML point of view, the response to the 7th deficiency, for example, would be that it is not of primary concern in an object-oriented OA&D language to model information flow. Moreover, it could be suggested that the employment of activity, sequence and collaboration diagrams combined in a semiformal way would probably suffice.

   It would probably be redundant at this point to mention that these three deficiencies are not considered as such from the UML literature point of view. The "loosely coupled collection of aspect models" is a fundamental choice in UML, and is tightly bound to the "semiformal" nature of the language.

3.2.3 "30 Things that Go Wrong in Object Modelling with UML 1.3"

The paper "30 Things that Go Wrong in Object Modelling with UML 1.3" [26] by Simons and Graham lists "a catalogue of problems experienced by developers, using various object modeling techniques brought into prominence by the widespread adoption of UML standard notations." The problems that will be mentioned here are the ones that relate to use case modeling.

First of all, the authors note that UML 1.3 addressed certain problems originating from the UML 1.1 semantics of the "uses" and "extends" relationships (used for use case analysis and decomposition), by replacing them with the "include" and "extend" relationships. These are now "stereotypes of dependency" instead of being "stereotypes of generalisation", like their predecessors. The modification helped to remove conflicts that had arisen in practise by the misunderstanding of the earlier "generalisation /specialization" semantics.

However, the authors proceed to state that the following inadequacies of the "include" and "extend" still hold:

1. *The semantics of "include" are under-specified.*
2. *The semantics of "extend" cannot handle exceptions.*
3. *The semantics of "extend" cannot handle alternative history.*

   The authors propose that an assertion that "included cases are atomic subroutines that are executed as a whole" is added to the semantics of the "include" keyword, in order to clarify the semantics. However, the association with programming languages' concepts ("subroutines") may not be considered completely satisfactory.

   The statements 2 and 3 are quite strong. However, they may not be quite fair:

   The argument in defence of statement 2 is that, while the "extend" relationship introduces "an inserted optional behaviour ... equivalent to a single-branch if-statement, in which control returns to the point of call afterwards", it is used to indicate "exceptions" by developers - and when an exception is executed, control never returns to this point. This, however, may be a far-fetched argument: there is no evidence that the UML authors consider the extend keyword as the equivalent of an "if-statement", nor the meaning of the word "exception" in UML literature is necessarily completely consistent with the meaning which the word has in programming languages' literature. The defence of statement 3 is again based on the previously mentioned understanding of the "extend" keyword as introducing a "programming language - like" exception.

Continuing their critique on use case modeling, the authors state the following:

- *Use case modeling misses long-range logical dependency.*
- *The granularity of use cases and logical tasks units are different.*
- *Use case dependency is non-logical and inconsistent.*
- *Use case modeling is unsound and must be deconstructed.*

Furthermore, the authors assume that *"the UML authors may undermine attempts to impose logical consistency by asserting that use cases "cannot be forward or reverse engineered""*. In addition, in the "Conclusions" section of [26], the suggestion is made that tasks analysis replaces use cases. The assumption and suggestion mentioned, seem to ignore the fact that the use of use case modeling is an inherent characteristic of UML, and that the use case view is just one of the "4+1" architectural views that UML provides (see subsection 2.4). However, it is a fact that there is still work to be done in order to deal with the inadequacies present in UML use case modeling specification.

3.3 Is UML a Requirements Specification Language? Discussion.

The list of deficiencies presented in the previous subsection may seem to be a long one. However, this should not make the technique seem "repulsive", since some deficiencies have remedies, while for others there is a question of whether they are actual deficiencies.

In any case, UML does not deserve to be heavily criticised for the weakness of the use case modeling, because UML should not be considered as a requirements specification language. It is a modeling language that employs a specific technique (the use case modeling) in order to capture the functional requirements of the system to be modeled. Actually, the UML specification clearly states that the language only **partially** deals with requirements specification *"Should the notation encompass requirement specification? Yes, partially."* [22]:Ch.1, p.13.

Moreover, the fact that the UML notation is semiformal should not a priori be considered a deficiency. The UML authors tried to "strike a balance between expressiveness and simplicity" [22]. The semiformal notation was one of the factors that enabled UML to become an industry standard. However, there are suggestions and attempts underway to move towards a formalisation of UML (e.g. [5, 16]). There is no doubt that UML would gain clarity, correctness and precision, if these efforts succeeded. Nevertheless, whether a target of formalisation can be successfully pursued without loss of simplicity is questionable. Afterall, to use an ethnomethodological argument, the existence of a definitive formal specification language is dubious. As Potts and Newstetter [24] put it: *"...(The) requirements engineering research should investigate the interpretation and use of specification languages rather more, and investigate the formal properties of such languages rather less."*

4 Practice "vs." Theory, Research "vs." Industry

It is a fact that many researchers question the clarity, the precision, or the very usefulness of UML as a modeling language - some may even consider that "developers are misguided" [26] by using UML for their OA&D activities. On the other hand, it is a fact that since its adoption as an OMG standard UML has become the software industry's "obvious choice for all kinds of visual modeling work" [23]. As an approved Publicly Available Specification (PAS) submitter to the International Organisation for Standardisation (ISO), the OMG has even proposed the UML specification for international standardisation [18]. These facts indicate that UML is likely to be here to stay, because it is needed - possibly until it is replaced by a more efficient language/notation. The presence of other methods' integration approaches such as OPEN, shows that the industry is really able to take decisive steps to adopt or create solutions when there is a need for them. The potential of integration of the UML notation with a generic industry-adopted modeling process, cannot be overseen.

But what about the theoretical deficiencies that have been identified in use case modeling, and have not so far been overcome? It is unavoidable that in many occasions decisions are taken in practice that may seem to be inconsistent with theory; all the same, these decisions may prove to be fruitful. Research should not -of course - be driven by the industry's choices; however, it not only is unavoidable but useful, as well, that it deals with the industry's needs. Thus, it seems that the relationship between research and industry should be a cooperative rather than an antagonistic one. The interchange of innovative ideas and standard practices between the members of these communities cannot but prove to be thriving for both.

5 Conclusions

This paper addressed UML from a requirements engineering point of view, focusing on use case modeling, as this is the technique that UML employs in order to capture the functional requirements of a system. An overview of use case modeling was presented, along with its deficiencies. The deficiencies were discussed and remedies for some of them were presented. It was argued that UML is more of a notation rather than a method, and that it is not a requirements specification language, since it only partially addresses the capturing of requirements.

UML is an evolving modeling language, and the potential to overcome its deficiencies exists. This is because UML is an extensively used language and adopted industry standard and, thus, in the centre of attention of the related research. Provided that steps are taken towards the integration of OA&D research suggestions and industry practices, an improved version of UML may be utilised in order to achieve increased efficiency in the capturing of requirements, as well as in the modeling activity in general.

Acknowledgements

The author would like to thank Dr. Steve Easterbrook, Bowen Hui and Theophanis Tsandilas for their constructive comments.

About the Author

Georgios Chalkiadakis is a Ph.D. student in the Department of Computer Science at the University of Toronto. He has received an M.Sc. in Computer Science from the Computer Science Department at the University of Crete (1999). He had been working for several years in the Center for Medical Informatics and Health Telematics Applications of the Institute of Computer Science in the Foundation for Research and Technology Hellas, and is quite an experienced UML user. His current research interests are preference aggregation, game theory, reinforcement learning and the imposition of social rules in multiagent systems.

References

[1] S.C. Bailin. An Object-Oriented Requirements Specification Method. *Comm. ACM*, 32(5):608–623, May 1989.

[2] G. Booch. Quality Software and the Unified Modeling Language. http:// www.rational.com/ products/ whitepapers/285.jsp.

[3] G. Booch. What is and what isn't Object-Oriented Design? *Am.Programmer*, 2(7–8):14–21, Summer 1989.

[4] G. Booch. *Object Oriented Analysis and Design with Applications.* Benjamic - Cummings Publishing, 1994.

[5] R. Breu, U. Hinkel, C. Hofmann, C Klein, B. Paech, B. Rumpe, and V. Thurner. Towards a Formalization of the Unified Modeling Language. In *Mehmet Aksit, Satoshi Matsuoka (ed.): Proceedings of the 11th European Conference on Object-Oriented Programming (ECOOP'97), LCNS 1241*, Jyvaskyla, Finland, June 1997. Springer Verlag.

[6] S. Brinkkemper, S. Hong, A. Bulthuis, and G. van den Goor. Object-Oriented Analysis and Design Methods: A Comparative Review. January 1995. http:// wwwis.cs.utwente.nl:8080/ dmrg/ OODOC/ oodoc/ oo.html.

[7] D.F. D'Souza and A.C. Wills. *Objects, Components and Frameworks with UML - The Catalysis Approach.* Addison-Wesley, 1999.

[8] R.G. Fichman and C.F. Kemerer. Object-Oriented and Conventional Analysis and Design Methodologies. *IEEE Computer*, 25(10), October 1992.

[9] M. Fowler and K. Scott. *UML Distilled : A Brief Guide to the Standard Object Modeling Language.* Addison-Wesley, 2000.

[10] M. Glinz. Problems and Deficiencies of UML as a Requirements Specification Language. In *Proceedings of the 10th International Workshop on Software Specification and Design (IWSSD-10)*, pages 11–22, San Diego, November 2000.

[11] B. Henderson-Sellers. The Open Methodology. *Object Magazine*, 6(9):56–59, November 1996.

[12] P. Hruby. Structuring Specification of Business Systems with UML (with an Emphasis on Workflow Management Systems). In *OOPSLA Workshop: Business Object Design and Implementation IV: From Business Objects to Complex Adaptive Systems*, Vancouver B.C., 1998.

[13] I. Jacobson. Applying UML in the Unified Process. In *UML World Conference (Presentation)*, New York, 1999. http://www.rational.com/media/uml/ up_uml.zip.

[14] I. Jacobson, M. Christerson, M. Jonsson, and P. van Overgaard. *Object-Oriented Software Engineering: A Use-Case Driven Approach.* Addison-Wesley, 1992.

[15] S. Joos. *ADORA-L - A modeling language for specifying software requirements.* PhD thesis, University of Zurich.

[16] S. Kent. Constraint diagrams: visualising invariants in OO models. In *OOPSLA' 97*. ACM Press, 1997.

[17] E. Kenworthy. Use Case Modelling - Capturing User Requirements. December 1997. http://www.zoo.co.uk/~z0001039/ PracGuides/ pg_use_cases.htm.

[18] C. Kobryn. UML 2001: A Standardization Odyssey. *Communications of the ACM*, 42(10), October 1999.

[19] P. Kruchten. Architectural Blueprints - The "4+1" View Model of Software Architecture. *IEEE Software*, 12(6):42–50, November 1995.

[20] P. Kruchten. *The Rational Unified Process: An Introduction*. Addison-Wesley, Reading, Mass., 1998.

[21] P.H. Loy. A Comparison of O-O and Structured Development Methodologies. *Software Engineering Notes*, 15(1):44–48, January 1990.

[22] OMG. *OMG Unified Modeling Language Specification*. March 2000.

[23] M. Penker. The Unified Modeling Language. July 1998. http://www.boldsoft.com/news/columns/column1998july.htm.

[24] C. Potts and W.C. Newstetter. Naturalistic Inquiry and Requirements Engineering: Reconciling Their Theoretical Foundations. In *Proceedings of the Third IEEE International Symposium on Requirements Engineering*, pages 118–127, Annapolis, USA, 1997. IEEE Computer Society Press.

[25] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modeling and Design*. Prentice-Hall, Englewood Cliffs, New Jersey, 1991.

[26] A.J.H. Simons and I. Graham. 30 things that go wrong in object modelling with UML. In *Kilov H., Rumpe B. and Simmonds I. (ed.): Precise Behavioural Specification of Businesses and Systems (Chapter 17)*. Kluwer Academic Publishers, 1999.

[27] Rational Software. Rational Unified Process. http:// www.rational.com/media/ products/ rup/D174D.pdf.

[28] A.I. Wasserman, P.A. Pircher, and R.J. Muller. An Object-Oriented Structured Design Method for Code Generation. *ACM Software Engineering Notes*, 14(1):32–55, January 1989.

[29] A. Wegmann and G. Genilloud. The Role of Roles in Use Case Diagrams. In *Proceedings for Conference UML 2000*, pages 210–224, York, UK, October 2000.

[30] R. Wirfs-Brock, B. Wilkerson, and L. Wiener. *Designing Object-Oriented Software*. Prentice-Hall, Englewood Cliffs, New Jersey, 1990.