UNIFIED
MODELING
LANGUAGE

UML Summary

version 1.0
13 January 1997

ad/97-01-02

RATIONAL
SOFTWARE CORPORATION
2800 San Tomas Expressway
Santa Clara, CA 95051-0951
http://www.rational.com

Contents

1. PREFACE

This series of documents describes the Unified Modeling Language (UML), a language for specifying, visualizing, and constructing the artifacts of software systems, as well as for business modeling. The UML represents a collection of “best engineering practices” that have proven successful in the modeling of large and complex systems.

The UML definition consists of the following documents:

UML Summary - This is the document you are reading. It serves as an introduction to the UML as well as a road map to the other documents.

UML Semantics - This document describes the formal metamodel that is the foundation for the UML’s semantics. The UML metamodel is presented in UML notation and concise natural language. A ***UML Glossary*** is included as an appendix.

UML Notation Guide - This document describes the UML notation and provides examples.

***UML Process-Specific Extensions* -** This describes the process-specific extensions to the UML, in terms of its extension mechanisms and process-specific diagram icons.

These documents are available on Rational Software’s web site, *http://www.rational.com.* Additional documents are associated with the proposal of the UML to the Object Management Group (OMG) and are available to OMG members as documents ad/97-01-01 through ad/97-01-14. See *http://www.omg.org* for OMG information.

1.1 INTENDED AUDIENCE

This document set is intended primarily as a precise and self-consistent definition of the UML’s semantic and notation constructs. The primary audience of this document set consists of the OMG, other standards organizations, book authors, trainers, and tool builders. The authors assume familiarity with object-oriented analysis and design methods, and the writing style is conceptually intense, often emphasizing precision over understandability. These documents are therefore not written as an introductory text on building object models for complex systems, although they could be used in conjunction with other materials or instruction. This set of documents will become more approachable to a broader audience as additional books, training courses, and tools that apply the UML become available.

1.2 ADDITIONAL INFORMATION AND UPDATES

Additional information, as well as any updates to the UML will appear on Rational Software’s web site, *http://www.rational.com/uml.*

2. MOTIVATION

This section discusses the importance of modeling in the face of increasingly complex system requirements.

Why Modeling Languages are Needed

We build models of complex systems because we cannot comprehend any such system in its entirety. As the complexity of systems increase, so does the importance of good modeling techniques. There are many additional factors of a project's success, but having a rigorous *modeling language* standard is one essential factor. A modeling language must include:

- Model elements — fundamental modeling concepts and semantics
- Notation — visual rendering of model elements
- Guidelines — idioms of usage within the trade

The Future of Software

As software increases its strategic value the industry looks for techniques to automate the production of software We look for techniques to improve quality and reduce cost and time-to-market. These techniques include componentware, visual programming, patterns, and frameworks. We also seek techniques to manage the complexity of systems as they increase in scope and scale. In particular, we recognize the need to solve recurring architectural problems, such as physical distribution, concurrency, replication, security, load balancing, and fault tolerance.

Complexity will vary by application domain and process phase. One of the key motivations in the minds of the UML developers was to create a set of semantics and notation that can adequately address all scales of architectural complexity across all domains.

Integration of Best Practices

A key motivation behind the development of the UML has been to integrate the best practices in the industry, encompassing widely varying views based on levels of abstraction, domains, architectures, life cycle stages, implementation technologies, etc. The specific objectives are covered in the next section.

3. UML PAST, PRESENT, AND FUTURE

3.1 UML 0.8 - 0.91

Precursors to UML

Identifiable object-oriented modeling languages began to appear between mid-1970 and the late 1980s as various methodologists experimented with different approaches to object-oriented analysis and design. The number of identified modeling languages increased from less than 10 to more than 50 during the period between 1989-1994. Many users of OO methods had trouble finding complete satisfaction in any one modeling language, fueling the “method wars.” By the mid-1990s, a new iterations of these methods began to appear, most notably Booch ’93, the continued evolution of OMT, and Fusion. These methods began to incorporate each other’s techniques, and a few clearly prominent methods emerged, including the OOSE, OMT-2, and Booch ’93 methods. Each of these were complete methods, but were recognized as having certain strengths. In simple terms, OOSE was a use-case oriented approach that provided excellent support business engineering and requirements analysis. OMT-2 was especially expressive for analysis and data-intensive information systems. Booch-’93 was particularly expressive during design and construction phases of projects and popular for engineering-intensive applications.

Booch, Rumbaugh, and Jacobson Join Forces

The development of UML began in October of 1994 when Grady Booch and Jim Rumbaugh of Rational Software Corporation began their work on unifying the Booch and OMT (Object Modeling Technique) methods. Given that the Booch and OMT methods were already independently growing together and were collectively recognized as leading object-oriented methods worldwide, Booch and Rumbaugh joined forces to forge a complete unification of their work. A draft of the Unified Method (as it was then called), 0.8, was released in October of 1995. Also in the Fall of 1995, Ivar Jacobson joined this unification effort, merging in the OOSE (Object-Oriented Software Engineering) method.

As the primary authors of the Booch, OMT, and OOSE methods, Booch, Rumbaugh, and Jacobson were motivated to create a unified modeling language for three reasons. First, these methods were already evolving toward each other independently. It made sense to continue that evolution together rather than apart, eliminating the potential for any unnecessary and gratuitous differences that would further confuse users. Second, by unifying the semantics and notation, they could bring some stability to the object-oriented marketplace, allowing projects to settle on one mature modeling language and letting tool builders focus on delivering more useful features. Third, they expected that their collaboration would yield improvements in all three earlier methods, helping them to capture lessons learned and to address problems that none of their methods previously handled well.

As they began their unification, they established four goals to bound their efforts:

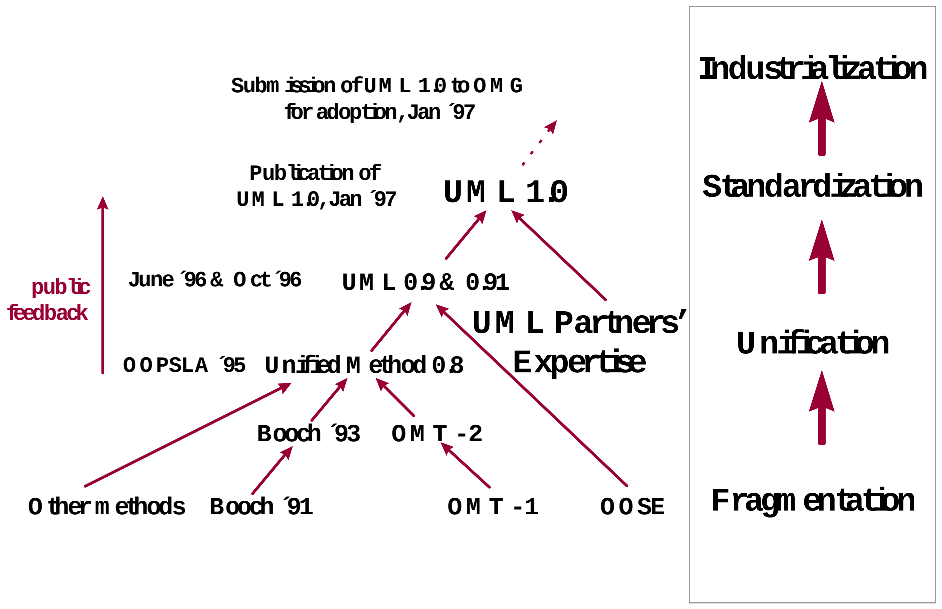
- To model systems (and not just software) using object-oriented concepts
- To establish an explicit coupling to conceptual as well as executable artifacts
- To address the issues of scale inherent in complex, mission-critical systems
- To create a modeling language usable by both humans and machines

Devising a notation for use in object-oriented analysis and design is not unlike designing a programming language. First, one must bound the problem: Should the notation encompass requirements specification? Should the notation extend to the level of a visual programming language? Second, one must strike a balance between expressiveness and simplicity: Too simple a notation will limit the breadth of problems that can be solved; too complex a notation will overwhelm the mortal developer. In the case of unifying existing methods, one must also be sensitive to the installed base: Make too many changes, and you will confuse existing users. Resist advancing the notation, and you will miss the opportunity of engaging a much broader set of users. The UML definition strives to make the best tradeoffs in each of these areas.

The efforts of Booch, Rumbaugh, and Jacobson resulted in the release of the UML 0.9 and 0.91 documents in June and October of 1996. During 1996, the UML authors invited and received feedback from the general community. They incorporated this feedback, but it was clear that additional focused attention was still required.

3.2 UML 1.0 WITH THE UML PARTNERS

During 1996, it became clear that several organizations saw UML as strategic to their business. A UML Partners consortium was established with several organizations willing to dedicate resources to work toward a strong UML definition. Those contributing most to the UML 1.0 definition included: Digital Equipment Corp., HP, i-Logix, Intellicorp, IBM, ICON Computing, MCI Systemhouse, Microsoft, Oracle, Rational Software, TI, and Unisys. This collaboration produced UML 1.0, a modeling language that is well defined, expressive and powerful, and generally applicable.



The UML Partners contributed a variety of expert perspectives, including, but not limited to the following: OMG and RM-ODP technology perspectives, business modeling, state machine semantics, types, interfaces, components, collaborations, refinement, frameworks, distribution, and meta-metamodel. The final result—UML 1.0—was a collaborative team effort. A list of individuals contributing to the UML is in the Acknowledgments section.

3.3 UML Present

The UML is being submitted to the OMG for considered adoption as a standard, coincident with the publication of this document.

The UML is nonproprietary and open to all. It addresses the needs of user and scientific communities, as established by experience with the underlying methods on which it is based. Many methodologists, organizations, and tool vendors have committed to use it. Since the UML builds upon similar semantics and notation from Booch, OMT, OOSE, and other leading methods and has incorporated input from the UML partners and feedback from the general public, widespread adoption of the should be straightforward.

There are two aspects of "unified" that the UML achieves: First, it effectively ends many of the differences, often inconsequential, between the modeling languages of previous methods. Secondly, and perhaps more importantly, it unifies the perspectives among many different kinds of systems (business versus software), development phases (requirements analysis, design, and implementation), and internal concepts.

3.4 UML Future

As of this writing, the following milestones are planned:

- The UML Partners will submit the UML document set to the OMG on January 16th, 1997.
- The UML Partners will respond to OMG and public feedback during first half of 1997.
- The OMG will decide about adopting UML as a standard in mid-1997.
- The UML methodologists, as well as other authors in the industry will publish additional collateral and books throughout 1997.

Standardization of the UML

Many organizations have already announced support for the UML as their organization's standard, since it is based on the modeling languages of leading OO methods. The UML is ready for widespread use. The UML 1.0 release is a stable and usable version. These documents are suitable as the primary source for authors writing books and training materials, as well as developers implementing visual modeling tools. Additional collateral, such as articles, training courses, examples, and books, will soon make the UML very approachable for a wide audience.

In January of 1997, the UML version 1.0 documents will be submitted, along with others, in response to the Object Management Group (OMG) Analysis & Design Task Force's RFP-1. The OMG will decide about adopting UML as a standard, hopefully by mid-

1997. As with all definitions, expect the UML to evolve. The process of preparing for the submission to the OMG, for example, has and will continue to provide a valuable source of input.

Discussion Groups and Providing Feedback

There are several electronic discussion forums that are appropriate for general discussion about the UML, including the internet news group *comp.object.*

The UML partners will log and consider comments on the UML submitted via e-mail to *uml_feedback@rational.com.* Constructive comments that reference specific sections in the UML documents will be more easy to incorporate. Depending on the volume of comments, we may not be able to respond to each e-mail individually.

Industrialization

Many organizations and vendors have already embraced the UML. The number of endorsing organizations is expected to grow significantly over time. These organizations will continue to encourage the use of the Unified Modeling by making the definition readily available and by encouraging other methodologists, tool vendors, training organizations, and authors to adopt the UML.

The real measure of the UML's success will be its use on successful projects and the increasing demand for supporting tools, books, training, and mentoring.

4. SCOPE OF THE UML

The Unified Modeling Language (UML) is a language for specifying, constructing, visualizing, and documenting the artifacts of a software-intensive system.

First and foremost, the Unified Modeling Language fuses the concepts of Booch, OMT, and OOSE. The result is a single, common, and widely usable modeling language for users of these and other methods.

Second, the Unified Modeling Language pushes the envelope of what can be done with existing methods. In particular, the UML authors targeted the modeling of concurrent, distributed systems, meaning that UML contains elements that address these domains.

Third, the Unified Modeling Language focuses on a standard modeling language, not a standard process. Although the UML must be applied in the context of a process, it is our experience that different organizations and problem domains require different processes. (For example, the a development process for shrink-wrapped software is an interesting one, but building shrink-wrapped software is vastly different from building hard-real-time avionics systems upon which lives depend.) Therefore, the efforts concentrated first on a common metamodel (which unifies semantics) and second on a common notation (which provides a human rendering of these semantics). The UML authors will not necessarily standardize a process, although we will continue to promote a development process that is *use-case driven, architecture centric, and iterative and incremental*.

4.1 PRIMARY ARTIFACTS OF THE UML

What are the primary artifacts of the UML? This can be answered from two different perspectives: the UML definition itself and how it is used to produce project artifacts.

4.1.1 UML-Defining Artifacts

To aid the understanding of the artifacts that constitute the Unified Modeling Language itself (the "inside" view), this document set consists of a *UML Summary* (which you are now reading), *UML Semantics*, *UML Notation Guide*, and *UML Process-Specific Extensions* document. Some context for each of these is described below. In addition to these documents, books are planned that will focus on understandability, examples, and common usage idioms.

4.1.1.1 UML Semantics

The *UML Semantics* document describes the precise model that underlies the UML, presented both in prose as well as in the UML notation itself. The UML partners began with a precise metamodel, using the notation of the UML itself supplemented by English text. The purpose of the metamodel was to provide a single, common, and definitive statement of the syntax and semantics of the elements of the UML. The presence of this metamodel has made it possible for its developers to agree on semantics, de-coupled from the human-factors issues of how those semantics would best be rendered. Additionally, the metamodel has made it possible for the team to explore ways to make the modeling language much more simple by, in a sense, unifying the elements of the Unified Modeling Language. (For example, commonality among the concepts of types, patterns, and use cases was discovered.) The authors expect select individuals to express this metamodel even more precisely by describing its semantics using formal techniques.

A metamodel is a language for specifying a model, in this case an object model. Metamodels are important because they can provide a single, common, and unambiguous statement of the syntax and semantics of a model. The "level" of meta in a model is somewhat arbitrary, and the UML developers consciously chose a semantically rich level, because that level is necessary to enable the semantically rich agreement necessary for tool interchange and design of complex systems.

4.1.1.2 UML Notation Guide

The *UML Notation Guide* describes the UML notation and provides examples. The graphical notation and textual syntax are the most visible part of the UML (the "outside" view), used by humans and tools to model systems. These are representations of a user-level model, which is semantically an instance of the UML metamodel. The standard diagram types are listed in the next section below.

4.1.1.3 UML Process Extensions

The *UML Process Extensions* document proposes certain process-specific values of the UML extension mechanisms (i.e. stereotype, tagged values, and constraints), as well as their associated icons, if any.

4.1.2 Development Project Artifacts

The choice of what model projections one creates has a profound influence upon how a problem is attacked and how a solution is shaped. *Abstraction*, the focus on relevant details while ignoring others, is a key to learning and communicating. Because of this:

- Every complex system is best approached through a small set of nearly independent views of a model; No single view is sufficient.
- Every model may be expressed at different levels of fidelity.
- The best models are connected to reality.

In terms of the views of a model, the UML defines the following graphical diagrams:

- use case diagram
- class diagram
- behavior diagrams
  - state diagram
  - activity diagram
  - sequence diagram
  - collaboration diagram
- implementation diagrams
  - component diagram
  - deployment diagram

These diagrams provide multiple perspectives of the system under analysis or development. The underlying model integrates these perspectives so that a self-consistent system can be analyzed and built. These diagrams, along with supporting documentation, are the primary artifacts that a modeler sees, although the UML and supporting tools will provide for a number of derivative views. These diagrams are further described in the *UML Notation Guide*.

Notation and Semantics History

The UML is an evolution from Booch, OMT, OOSE, most 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 (because they were confusing, superfluous, or little-used) and with a few new symbols added. The ideas in the UML come from the community of ideas developed by many different people in the object-oriented field. The UML developers did not invent most of these ideas; rather their role was to select and integrate ideas from the best OO and computer-science practices. The actual genealogy of the notation and underlying detailed semantics is complicated, so it is discussed here only to provide context, not to represent precise history.

Use-case diagrams are similar in appearance to those in OOSE.

Class diagrams are a melding of OMT, Booch, class diagrams of most other OO methods. Process-specific extensions (e.g., stereotypes and their corresponding icons) can be defined for various diagrams to support other modeling styles.

State diagrams are substantially based on the statecharts of David Harel with minor modifications. The *Activity diagram*, which shares much of the same underlying semantics, is similar to the work flow diagrams developed by many sources including many pre-OO sources.

Sequence diagrams were found in a variety of OO methods under a variety of names (*interaction*, *message trace*, and e*vent trace*) and date to pre-OO days. *Collaboration diagrams* were adapted from Booch (*object* diagram), Fusion (*object interaction graph*), and a number of other sources.

Collaborations are now first-class modeling entities, and often form the basis of *patterns*.

The *implementation diagrams* (*component* and *deployment* diagrams) are derived from Booch's *module* and *process* diagrams, but they are now component-centered, rather than module-centered and are far better interconnected.

Stereotypes are one of the extension mechanisms and extend the semantics of the metamodel. User-defined icons can be associated with given stereotypes for tailoring the UML to specific processes.

Each of these concepts has further predecessors and many other influences. We realize that any brief list of influences is incomplete and we recognize that the UML is the product of a long history of ideas in the computer science and software engineering area.

4.2 OUTSIDE THE SCOPE OF THE UML

The UML 1.0 definition intentionally does not address standards for tools and process, since these represent fairly separate concerns. Standardizing a language is necessarily the foundation for tools and process.

Tools

The Object Management Group's RFP (OADTF RFP-1) was a key driver in motivating the UML definition. The primary goal of the RFP was to enable tool interoperability. However, tools and their interoperability are very dependent on a solid semantic and notation definition, such as the UML provides. Although the UML defines a *semantic* metamodel, not a *tool* metamodel, the two should be fairly close to each other, and tool interchange can be defined consistently with the semantic and notation definitions.

Coincident with the UML submission to the OMG, the UML Partners have also submitted a UML-compliant tool interface definition, using CDIF/EIA standards as its bulk transfer encoding and syntax.

The UML documents do include some tips to tool vendors on implementation choices, but do not address everything needed. For example, they don't address topics like diagram layout, coloring, user navigation, animation, or other desirable features.

Process

Many organizations will use the UML as a common language for its project artifacts, but will use the same UML diagram types in the context of different processes. The UML is intentionally process independent, and defining a standard process was not a goal of the UML or OMG's Request-for-Proposal.

The UML authors *do* recognize the importance of process. The presence of a well-defined and well-managed process is a key discriminator between hyperproductive projects and unsuccessful ones. The reliance upon heroic programming is not a sustainable business practice. A process 1) provides guidance as to the order of a team's activities, 2) specifies what artifacts should be developed, 3) directs the tasks of individual developers and the team as a whole, and 4) offers criteria for monitoring and measuring a project's products and activities.

Processes by their very nature must be tailored to the organization, culture, and problem domain at hand. What works in one context (shrink-wrapped software development, for example) would be a disaster in another (hard-real-time, human-rated systems, for example). The selection of a particular process will vary greatly, depending on such things like problem domain, implementation technology, and skills of the team.

Booch, OMT, OOSE, and many other methods have well-defined processes, and the UML can support most methods. There has been some convergence on development process practices, but there is not yet enough consensus for standardization. What will likely result in the industry is general agreement on best practices and potentially the embracing of a process framework, within which individual processes can be instantiated. Although the UML does not mandate a process, its developers have recognized the value of a *use-case driven, architecture-centric, iterative, and incremental* process, so were careful to enable (but not require) this with the UML.

4.3 Comparing UML to Modeling Languages of Booch, OMT, OOSE, and other Methods

It should be made clear that the Unified Modeling Language is not a radical departure from Booch, OMT, or OOSE, but rather the legitimate successor to all three. This means that if you are a Booch, OMT, or OOSE user today, your training, experience, and tools will be preserved, because the Unified Modeling Language is a natural evolutionary step. The UML will be equally easy to adopt for users of many other methods, but their authors must decide for themselves whether to embrace the UML concepts and notation underneath their methods.

The Unified Modeling Language is more expressive yet cleaner and more uniform than Booch, OMT, OOSE, and other methods. This means that there is value in moving to the Unified Modeling Language, because it will allow projects to model things they could not have done before. Users of most other methods and modeling languages will gain value by moving to the UML, since it removes the unnecessary differences in notation and terminology that obscure the underlying similarities of most of these approaches.

With respect to other visual modeling languages, including entity-relationship modeling, BPR flow charts, and state-driven languages, the UML should provide improved expressiveness and holistic integrity.

Users of existing methods will experience slight changes in notation, but this should not take much relearning and will bring a clarification of the underlying semantics. If the unification goals have been achieved, UML will be an obvious choice when beginning new projects, especially as the availability of tools, books, and training becomes widespread. Many visual modeling tools support existing notations, such as Booch, OMT, OOSE, or others, as views of an underlying model; when these tools add support for UML (as some already have) users will enjoy the benefit of switching their current models to the UML notation without loss of information.

Existing users of any OO method can expect a fairly quick learning curve to achieve the same expressiveness as they previous knew. One can quickly learn and use the basics productively. More advanced techniques, such as the use of stereotypes and properties, will require some study, since they enable very expressive and precise models, needed only when the problem at hand requires them.

4.4 New features of the UML

The goals of the unification efforts were to keep it simple, to cast away elements of existing Booch, OMT, and OOSE that didn’t work in practice, to add elements from other methods that were more effective, and to invent new only when an existing solution was not available. Because the UML authors were in effect designing a language (albeit a graphical one), they had to strike a proper balance between minimalism (everything is text and boxes) and over-engineering (having an icon for every conceivable modeling element). To that end, they were very careful about adding new things, because they didn’t want to make the UML unnecessarily complex. Along the way, however, some things were found that were advantageous to add because they have proven useful in practice in other modeling.

There are several new concepts that are included in UML, including:

- stereotypes
- responsibilities
- extensibility mechanisms: stereotypes, tagged values, and constraints
- threads and processes
- distribution and concurrency (e.g. for modeling ActiveX/DCOM and CORBA)
- patterns/collaborations
- activity diagrams (for business process reengineering)
- clear separation of type, class, and instance
- refinement (to handle relationships between levels of abstraction)
- interfaces and components

Many of these ideas were present in various individual methods and theories but UML brings them together into a coherent whole. In addition to these major changes, there are many other localized improvements to the Booch, OMT, and OOSE semantics and notation.

5. ACKNOWLEDGMENTS

This section acknowledges the efforts of those who contributed significantly to defining UML 1.0 and making it successful.

As previously mentioned, please send specific feedback on the UML via e-mail to *uml_feedback@rational.com.*

UML Methodologists

- Grady Booch, egb@rational.com
- Ivar Jacobson, ivar@rational.com
- Jim Rumbaugh, rumbaugh@rational.com

UML 1.0 Core Team

- Digital Equipment
  - Paul Patrick, patrick@send.enet.dec.com
  - Jim Rye, rye@send.enet.dec.com
- Hewlett-Packard
  - Martin Griss, griss@hpl.hp.com
  - Reed Letsinger, letsinger@hpl.hp.com
- i-Logix
  - Eran Gery, erang@ilogix.co.il

- Prof. David Harel, harel@wisdom.weizmann.ac.il
- ICON Computing
  - Desmond D’Souza, dsouza@iconcomp.com
- James Odell
  - James Odell, 71051.1733@compuserve.com
- MCI Systemhouse
  - Cris Kobryn, ckobryn@acm.org
  - Joaquin Miller, miller@shl.com
- Microsoft
  - Philip A. Bernstein, philbe@microsoft.com
  - Rick Hargrove, rickha@microsoft.com
  - Andy Moss, andymo@microsoft.com
- Oracle
  - Guus Ramackers, gramacke@uk.oracle.com
- Rational Software
  - Ed Eykholt, eykholt@rational.com
  - Grant Larsen, gjl@rational.com
  - Dave Tropeano, davet@rational.com
- Texas Instruments
  - John Cheesman, j-cheesman@ti.com
  - Bob Hodges, bhodges@ti.com
  - Glenn Hollowell, glenn@ti.com
  - Keith Short, keiths@ti.com
- Unisys
  - Sridhar Iyengar, Sridhar.Iyengar@mv.unisys.com

Other Contributors and Supporters

We appreciate the contributions, influence, and support of the individuals listed below. In a very few number of cases, individuals mentioned here have not formally endorsed the UML, but are nonetheless appreciated for their influence.

> Hernan Astudillo, Dave Bernstein, Michael Blaha, Gary Cernosek, Michael Jesse Chonoles, Magnus Christerson, Dai Clegg, Peter Coad, Derek Coleman, Steve Cook, Ward Cunningham, Raj Datta, Mike Devlin, Bruce Douglass, Staffan Ehnebom, Maria Ericsson, Johannes Ernst, Don Firesmith, Martin Fowler, Eric Gamma, Dipayan Gangopadhyay, Richard Helm, Michael Hirsch, Yves Holvoet, Jon Hopkins, John Hsia, Ralph Johnson, GK Khalsa, Philippe Kruchten, Paul Kyzivat, Martin Lang, Mary Loomis, Robert Martin, Bertrand Meyer, Mike Meier, Randy Messer, Greg Meyers, Paul Moskowitz, Gunnar Overgaard, Jan Pachl, Bill Premerlani, Jeff Price, Jerri Pries, Terry Quatrani, Rich Reitman, Rudolf M. Riess, Kenny Rubin, Danny Sabbah, Ed Seidewitz, Gregson Siu, Jeff Sutherland, Dan Tasker, Andy Trice, Dan Uhlar, John Vlissides, Paul Ward, Rebecca Wirfs-Brock, Bryan Wood, Ed Yourdon, and Steve Zeigler.