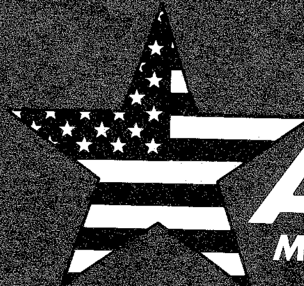
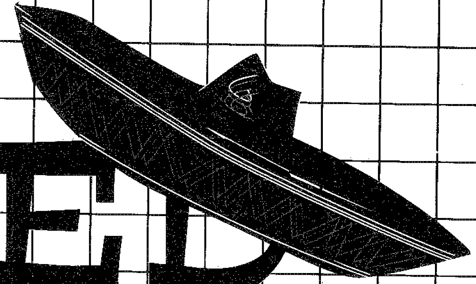
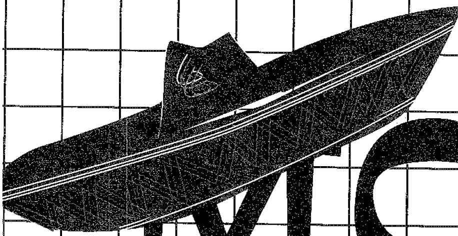


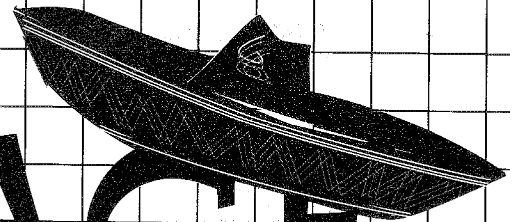
AMERICAN PROGRAMMER®
MARCH 1997 Vol.10 No.3



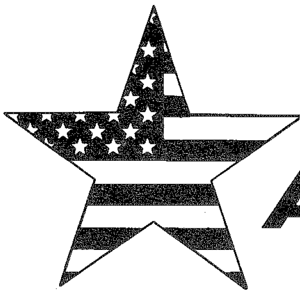
UNIFIED



MODELING



LANGUAGE



AMERICAN PROGRAMMER

CONTENTS

MARCH 1997 VOL. 10 NO. 3

3
LETTERS TO THE EDITOR

4
OO METHODOLOGY ISSUES:
WHAT ARE THEY AND DO THEY MATTER?
Katharine Whitehead

9
UML: A CURSE OR BLESSING FOR THE OO COMMUNITY?
Peter Hruschka

15
CHOOSING BETWEEN OPEN AND UML
Brian Henderson-Sellers and Donald Firesmith

24
GOOD NEWS, BAD NEWS: A PRACTITIONER'S OPINIONS ON UML
Carl A. Argila

31
BARRIERS TO UML ADOPTION
Anthony I. Wasserman

37
UML: THE POSITIVE SPIN
Bertrand Meyer

EDITOR
Ed Yourdon

EDITORIAL BOARD

Larry L. Constantine
Bill Curtis
Tom DeMarco
Capers Jones
Tomoo Matsubara
Roger Pressman
Paul A. Strassmann
Rob Thomsett

American Programmer® (ISSN 1048-5600) is published 12 times a year by Cutter Information Corp., 37 Broadway, Suite 1, Arlington, MA 02174-5552 (617/648-8702 or, within North America, 800/964-8702; Fax 617/648-1950 or, within North America, 800/888-1816); E-mail: lovering@cutter.com, Web site: http://www.cutter.com. American Programmer covers the software scene, with particular emphasis on those events that will impact the careers and jobs of programmers, systems analysts, and data processing managers around the world. Editor: Ed Yourdon. Publisher: Karen Fine Coburn. Managing Editor: Karen Kunkel Pasley. Production Editor: Rosanne DePasquale. Client Services Manager: Kara Lovering. List Manager: Doreen Evans. Reprint Manager: Carolyn Licata. © 1997 by Cutter Information Corp. All rights reserved. American Programmer is a trademark of Cutter Information Corp. No material in this publication may be reproduced, eaten, or distributed without written permission from the publisher. Unauthorized reproduction in any form, including photocopying, faxing, and image scanning, is against the law. Subscription rates are US$435 a year in North America, US$535 elsewhere, payable to Cutter Information Corp. Reprints, bulk purchases, past issues, site licenses, multiple subscription rates available on request.

Since October 1995, there has been a temporary truce in the "methodology wars" that have plagued the object-oriented community since the late 1980s. The cause of the truce was the introduction of a draft version of a "Unified Modeling Language" (subsequently known to one and all as UML) by Grady Booch, James Rumbaugh, and Ivar Jacobson of Rational Software Corporation. All three are highly respected authors and consultants, and the OO methodologies they had developed on their own probably represented an aggregate 80 percent of the marketplace. Thus their decision to work together as "the three amigos" and to consolidate their ideas into a unified whole was significant indeed. Even more significant was the fact that all of this was presented at the annual OOPSLA conference and that plans were laid to submit UML as a formal proposal to the Object Management Group (OMG).

More than a year has gone by, and two more revisions to the draft UML have appeared. By the time you receive this issue of *American Programmer*, the final draft will have been sent to OMG. Meanwhile, the majority of CASE vendors, consultants, authors, and methodologists appear to have accepted UML as the "heir apparent" to the OO methodology kingdom. But at least one alternative, known as OPEN, is also being submitted to OMG as a proposed standard for object-oriented analysis and design modeling notation. And the preliminary experiences and reactions from practitioners and observers suggest that UML has some problems as well as some benefits. If this is the case, and if UML is being considered for adoption by application development organizations around the world, this would be a good time to review its merits — and that's what this issue of *American Programmer* is all about. The three amigos were too deeply engrossed in the final revisions to the draft UML to be able to contribute an article for this issue, but we were able to solicit some very thought-provoking commentaries from a number of leading consultants and methodologists.

We begin with a review by Katharine Whitehead of the key methodology issues facing the object-oriented community. Katharine and I, together with three other colleagues from Software AG, coauthored *Mainstream Objects* in 1996, a textbook that had an objective of methodology synthesis and integration similar to UML's. She is also the principal developer of Seer Technologies' new OO method and a member of the OPEN Consortium that offers an alternative to UML.

Next we hear from Peter Hruschka, whose methodology experience includes several years of OO work as well as an earlier involvement with the structured methodologies. Like Carl Argila, whose article appears later in the issue, Hruschka reports on his experiences teaching early versions of UML to practitioners and applying UML as a consultant in real-world development projects. After reviewing the strong points and weak points of UML, he concludes by saying, "UML has unified some of the existing OO notations, thus creating a single point of reference for many important concepts. . . . [But] if you watch the evolution of the UML, you will notice that it becomes more complex from month to month, with more and more concepts being added to the graphical notation. . . . This is understandable, since UML tries to please as many methodologists as possible. It is, however, a dangerous trend, as UML runs the risk of becoming another PL/I."

Next, Brian Henderson-Sellers and Donald Firesmith compare UML and OPEN. Henderson-Sellers is the lead developer and spokesman for OPEN, and Firesmith is also highly involved in the effort (as well as being, like Henderson-Sellers, the author of several OO textbooks). As a result, it's not too surprising that their presentation concludes that OPEN has advantages and UML has disadvantages. But you'll find their discussion even-handed and low-key; they raise some excellent points for consideration.

Carl Argila's paper returns us to the practitioner's perspective. Carl and I have taught OO analysis/design workshops together for some four years, and we've coauthored a case-study book showing two thoroughly worked-out examples of an OO analysis and design model. Like Hruschka, Argila spends his time working on real-world projects, where the academic, esoteric aspects of a methodology often seem far less relevant. Like all of the authors in this issue, Argila stresses the benefits of having a common methodological approach, and he argues that "UML, for better or worse, is the best hope we have of establishing an industry-wide object-oriented notation and work product standard." His advice for practitioners of UML is priceless: I recommend that project teams post Argila's conclusion on their community bulletin boards to avoid falling into the trap of believing that

UML (or OPEN, or any other methodology, for that matter) is a "silver bullet."

One of the subthemes running through the discussions by Argila, Hruschka, and Whitehead is that UML is so complex that it may be difficult for the ordinary practitioner to understand it and use it properly. This is one of the issues raised by Anthony Wasserman, the former CEO of Interactive Development Environments (developer of the Software Through Pictures CASE tool) and a highly respected OO methodologist in his own right. In addition to the training difficulties caused by methodological complexity, Wasserman also warns of the costs organizations will incur in moving from their existing OO methodology to UML, as well as the difficulties CASE tool vendors will experience in providing thorough and comprehensive tool support for UML. He concludes by reminding us that "the vast majority of the software development community doesn't use any method or design notation at all, remaining unconvinced of the merits of these methods and notations relative to their current development techniques and tools."

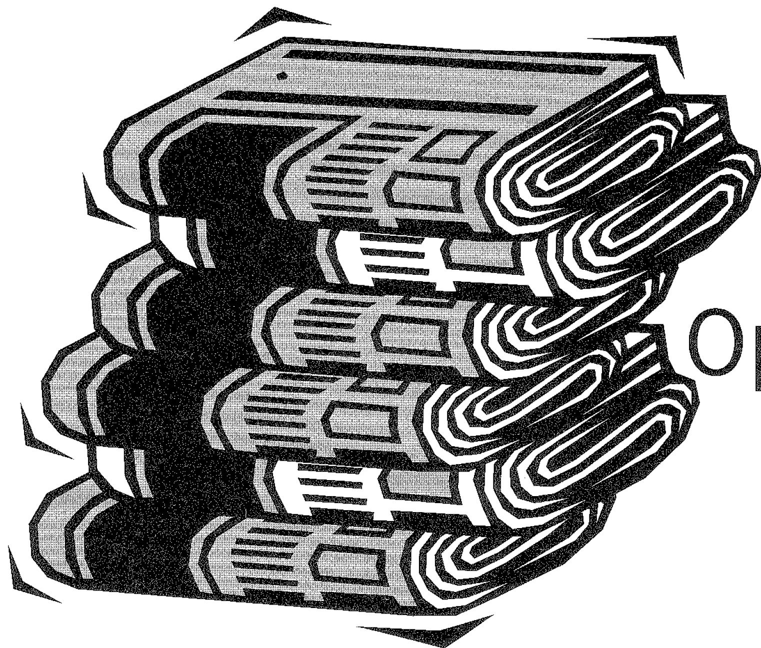
Finally, Bertrand Meyer provides a satirical look at UML, in the form of a letter from a hypothetical graduate student to his professor. While the essay is light-hearted in nature, it also contains some precise and thought-provoking observations and critiques of UML. Meyer, well known as the author of the Eiffel language and several outstanding textbooks on object-oriented software engineering, has obviously thought about these issues at great length.

It's important that you interpret all of these articles, and all of the opinions and predictions and warnings within them, in the proper context. While one might be tempted to characterize all of the negative comments as "sour grapes" from the methodologists whose work did not achieve the recognition that UML has received (or in the case of Henderson-Sellers and Firesmith, methodologists who still hope that their OPEN methodology *will* receive the same level of recognition and popularity), I think it's important to take them as constructive criticism and also as advice for practical implementation of UML. Methodologists and purists may continue to bicker about the intellectual merits of UML for years to come, but practitioners such as Argila and Hruschka seem inclined to believe that (a) UML is probably the best we can hope for at the present time, and (b) in any case, it's the one most of us will be stuck with. Hruschka and Wasserman both warn that in the worst case, UML could turn out to be the methodological equivalent of PL/I. Perhaps it would be better to say that UML is like Java. It's not perfect, and it may face serious competition from other technology camps. Nevertheless, it has some terrific possibilities, and we should be focusing on how best to use the new technology.

Having discussed OO methodologies at length, we'll turn our attention next month to a topic that includes not only analysis and design (whether OO or otherwise), but also the activities that precede analysis: requirements management. This is an exciting field, with new tools and books appearing from several vendors. We've lined up an excellent group of authors to share their insights with you.

Meanwhile, we invite you to share your insights with us. And as you'll see on page 3, there's a new forum for doing this. Bowing to popular demand, we've decided to institute a letters to the editor section in *American Programmer* each month. So if you see an idea that particularly intrigues you, an experience you've shared — or a wrong-headed opinion that really gets your goat — tell us all about it! You can write to us at Letters to the Editor, American Programmer, Cutter Information Corp., 37 Broadway, Arlington, MA 02174; fax us at 617/648-8707; or (let's be realistic) e-mail us at ed@yourdon.com. We look forward to hearing from you!

Ed Yourdon ★
Internet: ed@yourdon.com
Web site: http://www.yourdon.com



GOOD NEWS, BAD NEWS: A Practitioner's Opinions on UML

Carl A. Argila

Falling over an abstraction;
a dark instance of the class
bitter inheritance

ROME WASN'T BUILT IN A DAY . . .

As a student of history, I'm amazed at how many aspects of our culture, taken for granted today, were the products of long and difficult struggles, hundreds of years ago. Take, for example, the Roman alphabet. This miraculous invention allows me to convey my ideas, across space and time, with just 26 letters!

The Roman alphabet did not simply come about by accident or evolution. Although there were many versions and forms of the Roman alphabet, standardization did not happen until the year 799. History attributes this accomplishment to three men: an obscure Roman emperor of the Isaurian Dynasty, Boocificus, and his two faithful servants Rumbaticus and Jacobicus.

In the year 799, Emperor Boocificus convened an assembly of academicians and scholars of the day. The purpose of this meeting was to agree upon a Unified Alphabet for Writing Latin (UAWL). Fortunately a record of this conference has been passed down to us; the following have been extracted from this record:

Emperor Boocificus, trembling like a JAVA applet and flanked by his two faithful servants Rumbaticus and Jacobicus, addresses the audience. . . .

"Friends, Romans, and Countrymen, we meet here today to agree upon a standardized alphabet, which shall be used throughout the empire to communicate by means of the written word.

"We all agree that writing has a future; the Greeks have been doing it for centuries. However, over the years, numerous alphabets have evolved, creating chaos and misunderstanding.

"You are all aware of the frequent chariot accidents on the Appian Way — attributed to signs written in different alphabets.

"Our stadium operators tell us that they could reduce their operating costs enormously if advertising could be written in a single alphabet.

"Employers are finding it increasingly difficult to hire scribes with up-to-date alphabet skills. This is worsened by the fact that our universities are training students in dying alphabets.

"Although we all agree that the UAWL is needed, few of us agree as to what that standard should be.

"We have received numerous proposals. Many of you have suggested that our alphabet have one letter for each spoken sound; this is an appealing idea. As we investigated this approach, however, we realized that regional variations in speech would result in words being spelled differently throughout the empire.

"Many of you are familiar with the widely promoted concept of an alphabet with 7±2 letters. Promoters claim that an alphabet with more than 7±2 letters could never be memorized by the human brain. Upon looking further into this approach, we have discovered that it is being promoted by the Egyptian papyrus conglomerate, LETNI, in collaboration with Hindustan ink mogul Bi-Llga-Tes. It seems that a 7±2 letter alphabet would result in longer words, thereby consuming more papyrus and more ink. Clever, but we won't fall for it!

"And then, of course, we have a very vocal minority of religious zealots. They claim that it is the divine will of Jupiter and Juno that men and women have *different* alphabets; that women should have a 20-letter alphabet and that men should have a 21-letter alphabet. Such ideas will be short-lived — I have ordered all of them exiled to Persia."

You get the idea.

We all accept that, for better or for worse, we use the Roman alphabet as a tool — we don't even think about it! I've never heard any of my writer friends say, "The only thing standing between me and a Pulitzer Prize is a better alphabet!" Or imagine an author telling his or her editor that the book wasn't finished on time because of problems with the alphabet.

Not that the Roman alphabet is perfect. Personally, I have trouble distinguishing between d's and b's, unless they're used in context. (Or is it b's and d's?) And while we're reengineering the alphabet, let's do something with those Z's, which can easily be confused with 2s. Or 1, l, and I. Or 0 and O.

As versatile and robust as the Roman alphabet has proven to be, it has been known for hundreds of years that prose writing is inadequate to communicate complex scientific and engineering concepts. In chemistry, physics, civil engineering, architecture, and, of course, software engineering, other forms of written communication have evolved. These have typically been charts, diagrams, or other "visual tools." These visual tools differ from informal pictures or sketches in that they are composed from an "alphabet" of symbols. In turn, these symbols must be aggregated in accordance with some rules — the syntax of the diagram.

Most of you are familiar with the flowchart. This simple visual tool has been used in software engineering for over 50 years; it is most frequently used to describe sequential program logic. In creating a flowchart, an alphabet of shapes is aggregated in accordance with certain connection rules. Just as with words written in the Roman alphabet, there are flowcharts that are "misspelled" or "syntactically incorrect."

WHAT IS UML?

In software engineering, the evolution of concepts and techniques has continually demanded more robust communication techniques. The evolution of "structured methods" in the 1960s and 70s forced the venerable flowchart to give way to other visual tools — the structure chart, the data flow diagram, the entity-relationship diagram, and many more.

Today, our industry is moving toward object orientation as a paradigm for understanding, designing, and implementing software systems. Those of you familiar with object technology will appreciate the absurdity of attempting to represent object-oriented concepts with flowchart symbols; the concepts of *module*, *decision*, and *iteration* simply do not correspond to concepts such as *encapsulation*, *inheritance*, or *abstraction*.

Over the past decade, there has been enormous interest in object technology and techniques. The number of object-oriented methods has proliferated from about 5 to over 50 just since 1990. However, virtually all of these communication techniques represent and communicate the same underlying concepts. Differences between the various techniques are 90 percent cosmetic and 10 percent substance — and the substance component has been converging for the most widely used methods.

The Unified Modeling Language (UML) is a synthesis of the three most widely used object-oriented techniques: Booch-93 (Grady Booch), OMT-2 (James Rumbaugh, et al.), and OOSE (Ivar Jacobson) [2, 3, 4].

In 1995, Messrs. Booch and Rumbaugh released an initial specification for what they described as the *Unified Method* [2]. In this document, Messrs. Booch and Rumbaugh asserted that this specification would eventually provide the industry with a "standard method" for developing object-oriented applications. What was actually delivered, however, was a notation

and a description of work products. What was not included was any proposed techniques or heuristics (How do I go about creating these work products?) nor any proposed process (How do I get from concept to code?). For example, in some 200 pages of specification, there is no guidance as to how classes should be defined. Yet establishing the right classes is crucial if object-oriented applications are to be maintainable, extensible, and reusable (see [1]).

To their credit, Messrs. Booch and Rumbaugh admitted that "originally we took credit for a little more than we should have. . . . And we finally decided we should admit that" [5].

In 1996, Ivar Jacobson joined Booch and Rumbaugh in the UML effort. Subsequent addenda to the UML specification [3, 4] focus clearly on the specification of notation and work products.

We can consider UML to be a specification for the *alphabet* of object technology communication. Or, as the authors state, "This is why we say that UML is essentially the language of blueprints for software" [3].

OBSERVATIONS

First a caveat: I am not an expert on UML; I am a practitioner. For over a year, I have presented UML-compliant object technology training. This section presents some observations based on my experiences in teaching UML and applying UML on client projects.

The UML metamodel establishes a number of object-oriented development models, each of which serves a particular purpose in the object-oriented life cycle [3, 4]. These are:

- Class model
- State model
- Use case model
- Operational model
- System composition model

The UML notation establishes the format for a number of work products (diagrams) and the notation for constructing these diagrams [3, 4]. The UML diagrams are:

- Class diagram
- Use case diagram
- Sequence diagram
- Collaboration diagram
- State diagram
- Component diagram
- Deployment diagram

One of my first observations about UML was that it is clearly biased in favor of Messrs. Booch, Rumbaugh, and Jacobson's approaches and techniques — it is not a "neutral language." The authors readily admit this: "Indeed, UML assumes a process that is use case–driven, architecture-centered, iterative, and incremental" [3].

Suppose that, for some reason, I wish to develop an object-oriented application from a data-centric perspective — or a function-centric perspective. (Perhaps I'm reengineering an existing application.) Does UML prevent me from doing that? Not at all. I would not use use cases, and perhaps I might add an entity-relationship diagram or data flow diagram, or I might add some embellishments to the class diagram.[1]

Notwithstanding the comments above, I have found a use case–driven approach for object-oriented systems development to be extremely powerful. However, for real systems, there may be hundreds of use cases or component events for use cases. I have found the use case diagram (as distinguished from a use case approach) to be cumbersome. In my experience, the "value added" by this diagram does not justify its bulk. I have found it more useful to present use cases in a tabular form, such as (from [6]):

[1]I suspect that if Messrs. Booch, Rumbaugh, and Jacobson read the above paragraph, they would argue that any software system could be specified from a use case perspective, because every software system must recognize the occurrence of events and produce responses to those events. This, indeed, is true. However, in the real world, not every software systems analyst or designer is capable of executing a use case–driven approach. Many analysts and designers have skills in data modeling or process modeling; as practitioners we must capitalize on our existing resources.

1. PAID SUBSCRIPTION REQUESTED	A. CREATE NEW SUBSCRIPTION RECORD AS REQUIRED B. CREATE NEW SUBSCRIBER RECORD AS REQUIRED B.1 CREATE OR UPDATE ADDRESS RECORD AS REQUIRED C. CREATE NEW RECIPIENT RECORD AS REQUIRED C.1 CREATE OR UPDATE ADDRESS RECORD AS REQUIRED D. ESTABLISH SUBSCRIPTION PRICING E. POST PAYMENT AS REQUIRED E.1 ISSUE INVOICE AS REQUIRED F. NOTIFY SERVICE BUREAU OF SUBSCRIPTION AS REQUIRED G. ESTABLISH EXPIRATION DATE FOR SUBSCRIPTION H. ESTABLISH EXPIRATION WARNING DATE FOR SUBSCRIPTION
2. COMPLIMENTARY SUBSCRIPTION REQUESTED	A. CREATE NEW SUBSCRIPTION RECORD AS REQUIRED B. CREATE NEW SUBSCRIBER RECORD AS REQUIRED B.1 CREATE OR UPDATE ADDRESS RECORD AS REQUIRED C. CREATE NEW RECIPIENT RECORD AS REQUIRED C.1 CREATE OR UPDATE ADDRESS RECORD AS REQUIRED
3. PAYMENT REC	A. POST PA
4.	

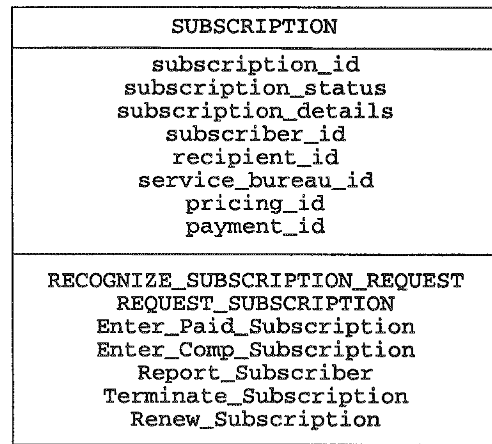
Another observation about the UML diagrams: the inclusion of a state diagram in UML reflects, in my view, the predominantly engineering/real-time backgrounds of Messrs. Booch, Rumbaugh, and Jacobson. To make UML more robust, in addition to the state diagram, there should be a data structure diagram and a process diagram. Or perhaps the state diagram should not be part of UML at all. Instead, the UML specification might simply say that "additional models and diagrams may be used to supplement the core models presented in this specification. . . ." The purpose of all these diagrams is to communicate ideas; no specification standard should be so rigid that it precludes the use of any mechanism that enhances communication.

The UML class diagram has served as the foundation for creating object-oriented analysis and design models. When walking through a proposed analysis or design, it is the class diagram that serves as our focal point. The class diagram, however, presents static information — classes, relationships between classes, and information about classes. When validating a use case, however, we need dynamic information about how objects collaborate to perform the work of the use case. For this, I have found the sequence diagram to be extraordinarily useful. The sequence diagram presents information that cannot be easily represented in any other way. The class diagram together with sequence diagrams form the core of our object-oriented modeling work products.

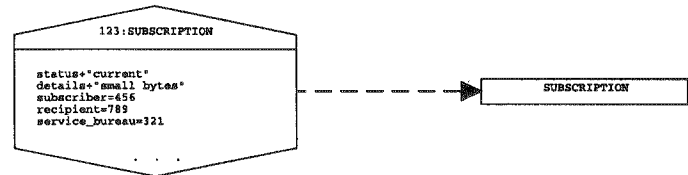
Although information about how objects collaborate is important, I have not found the collaboration diagram to be worth the trouble of creating it. Typically, in a walkthrough meeting, we focus on the class diagram and one or more sequence diagrams. To add collaboration diagrams becomes unwieldy.

Information about object collaborations is important, however, so I overlay this information on the class diagram rather than create separate collaboration diagrams. An automated tool should allow me to manage and view these overlays.

This brings up an important observation about the class diagram. It does not offer me the flexibility needed to distinguish between classes and instances of classes (objects) on the diagram. A class is represented as:

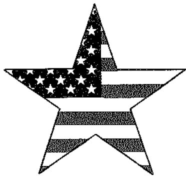


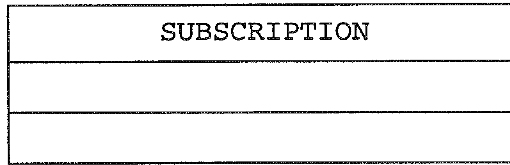
Should I wish specifically to identify an instance of that class (as I would when presenting an object collaboration), the UML notation would be:[2]



As you can imagine, for a complex model, the class diagram can become cumbersome. To solve this problem, I advise students and clients to consider:

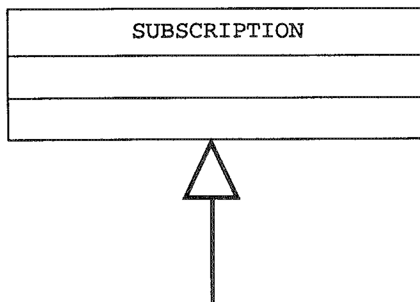
[2]The object symbol, on the left, is described by Mr. Booch as a "polygonal cloud."



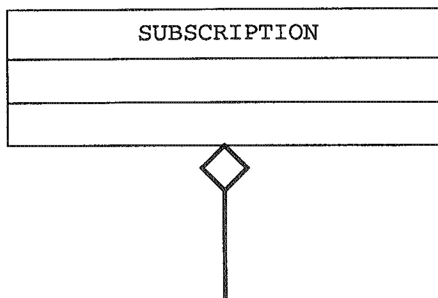


as representing either a class or some unspecified instance of the class. This solves the "wasted space" problem but now muddles together the distinction between a class and instances of that class.

There are other places in the UML class diagram where the distinction between a class and instances of that class are muddled. For example,

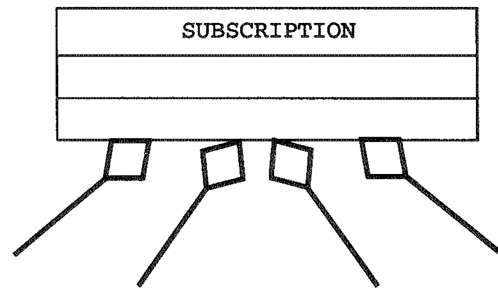


shows an inheritance relationship. These are relationships that exist between *classes*. However,

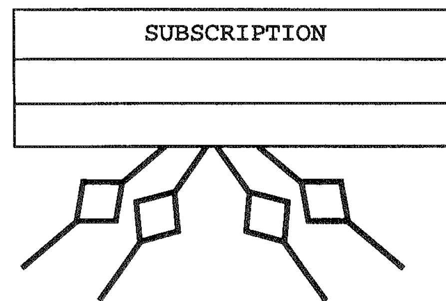


shows an aggregation relationship. These are relationships that exist between *instances* of the class. This lack of distinction has serious pedagogical consequences. Students for whom object concepts are new typically find it difficult to distinguish between the concept of a class and the concept of an instance of that class. Visual cues would help enormously in "internalizing" this distinction. I have struggled with this problem but have not found any satisfactory solution short of revising the notation.

While on the subject of notation, a typical aggregation would be represented as:



In reality, the area around the class border can get very crowded and easily becomes illegible. It would be convenient instead to show the aggregation as:



Unfortunately, the ♦ symbol is symmetric, making it unclear as to which class is the "whole" and which class is the "part."

So far I've not made any distinction between applying UML for object-oriented analysis versus object-oriented design. We use the OOA model to communicate with users and sponsors; we use the OOD model to communicate with developers and implementers. Fortunately, UML can be used for both models; the UML standard does not attempt to identify an analysis notation as distinct from a design notation. What the standard does not state, however, are the *heuristics* for how the application of UML should differ for an OOA model versus an OOD model. Because the audience for OOA models may not be technically sophisticated, I have found it useful to use only a subset of UML for these models. My core work products are the class and sequence diagrams together with use cases. Also, the notation on these diagrams is restricted to only that which captures essential application domain requirements. Any references to implementation technology are relegated to the OOD models.

OOD models, which are the blueprints for the implementers, capture much more technical

information. For these I use the full UML notation and the component and deployment diagrams.

Another early observation I made about UML was that it clearly evolved from pencil and paper techniques. The UML diagrams can all be created with pencil and paper as well as with an automated tool. But would we ever apply UML with pencil and paper alone? I think not; automation is such an integral part of our workplace that even the simplest of documents or memoranda are created with word-processing tools. In addition, the movement of documents from paper to silicon has made possible documents with hyperlinks and embedded multimedia; such documents can *only* exist in electronic forms. We gave up paper to achieve an enormously more powerful document.

Given that, in practice, UML would be used with an automated tool, it might have been possible to integrate more tightly the various models and diagrams. The multitude of UML diagrams used now (at least seven) could well have been consolidated into a single model with "views" or "layers," which could be selected electronically. For example, the case study models that my coauthor and I developed for our book [6] would typically not be printed on paper. In practice, such a model would be used only in its electronic form with one or more layers displayed, depending on the information that is of interest.

A final observation about UML. Though I might prefer that UML were less biased and more robust, I can overcome its imperfections and build my systems. Going back to our Roman alphabet analogy: clearly the Roman alphabet is biased towards the Romance languages. However, the Roman alphabet is versatile enough that it can be used to represent a diversity of languages. The Roman alphabet is even used to write Vietnamese — embellishments are simply added to convey features of Vietnamese that the basic Roman alphabet does not capture.

RECOMMENDATIONS

The following recommendations are based on a good deal of practical experience in teaching and applying object technology with UML. I hope you find them of value.

Understand the Technology

Understand that the concepts of object technology are independent of Booch, OMT, OOSE, UML, or any of the other 50+ notations. If you understand what is a class, object, abstraction, inheritance, encapsulation, and so on, then it doesn't really matter which notation you use.

Project teams that bicker incessantly about clouds versus boxes versus rounded rectangles have far deeper problems than just notation.

Understand the Difference Between Notation, Method, and Process

UML specifies *only* the "alphabet" for communicating object-oriented concepts. UML gives no guidance about how to create the various work products (i.e., heuristics).

Nor does UML specify a software development process; that is, how activities that result in the creation of work products are to be performed. If the history of engineering teaches us anything, we should not expect to see agreement on a standardized software development process anytime soon. Mature engineering disciplines such as architecture or civil engineering have enormous variations in process depending on the size and scope of the project, the culture of the organization, and numerous other factors.

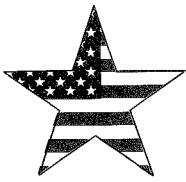
Understand What's the *Really* Hard Problem

Contrary to what compiler vendors may believe, the really hard part of building an object-oriented system is *not* the coding. The really hard problem is discovering what are the "right" objects in the first place. If you find the wrong objects, you risk impacting or destroying maintainability, reuse, extensibility, and so forth.

I fear that one of the reasons software methodologists focus so heavily on notation is that they don't have a lot to offer on solving the really hard problem.

Apply UML with Common Sense

- ★ The class diagram and sequence diagram are core work products; expect that they will form the basis for your development.
- ★ Consider use cases to be of enormous value, but replace use case diagrams with a more manageable work product (such as a table).



★ Expect the collaboration diagram to be of limited usefulness and consider integrating collaboration information as an overlay on the class diagram.

★ Recognize that state diagrams (part of UML) as well as data structure and process diagrams (not part of UML) may be useful adjuncts to the core work products. The usefulness of these supplementary work products will depend on a number of technical and nontechnical issues. The goal, however, is always to facilitate communication.

★ Apply UML differently for analysis and design activities. Expect that for analysis a subset of the UML notation will be applicable. Expect that for design more technical detail will be incorporated into the work products and additional diagrams will be used (e.g., the component and deployment diagrams).

★ Expect to extend the class diagram notation with supplements, overlays, and/or embellishments as circumstances dictate and as may be necessary to facilitate communication.

CONCLUSIONS

If you took all the software methodologists in the world and lined them up head-to-toe, you still wouldn't reach a conclusion — or a delivered system. If the brief history of software engineering is any indication, no standard will ever capture perfection, or universal acceptance.

Considering that Messrs. Booch, Rumbaugh, and Jacobson represent the principal object-oriented techniques in use today, I feel that UML, for better or worse, is the best hope we have of establishing an industry-wide object-oriented notation and work product standard.

Such a standard is essential if we expect tool vendors to produce object technology support tools of substance. Such a standard is essential if, in fact, we actually plan to reuse analysis and design work products! Such a standard is essential if we expect to see books and training materials that do more than simply regurgitate the same old ideas — only with different notations.

If you're committed to moving into object technology, UML is neither a hindrance nor an advantage. If you're looking for excuses *not* to move into objects, then UML is as good an excuse as anything else.

POSTSCRIPT

The reader may be interested in knowing that, after standardizing the Roman alphabet, Emperor Boocificus and his two faithful servants Rumbaticus and Jacobicus went on to devise a system for writing numbers. We know this system today as the Roman numerals.

REFERENCES

1 Argila, C. "Finding and Keeping Good Objects." *American Programmer*, Vol. 7, no. 10 (October 1994), pp. 36–43.

2 Booch, G., and J. Rumbaugh. *Unified Method (Version 0.8)*. Santa Clara, CA: Rational Software Corporation, 1995.

3 Booch, G., J. Rumbaugh, and I. Jacobson. *Unified Modeling Language (Version 0.9) Addendum*. Santa Clara, CA: Rational Software Corporation, 1996.

4 Booch, G., J. Rumbaugh, and I. Jacobson. *Unified Modeling Language (Version 0.91) Addendum*. Santa Clara, CA: Rational Software Corporation, 1996.

5 Meyer, B. "Gurus Share Insights on Objects." *IEEE Computer*, Vol. 29 (June 1996), pp. 95–98.

6 Yourdon, E., and C. Argila. *Case Studies in Object-Oriented Analysis and Design*. Upper Saddle River, NJ: Prentice Hall, 1996.

Carl Argila has worked for over 30 years as a practicing software engineer. Dr. Argila is president of aLigra Systems, Inc., a consultancy specializing in object-oriented training and mentoring. He is coauthor of Case Studies in Object-Oriented Analysis and Design *(Prentice Hall, Yourdon Press, 1996).*

Dr. Argila may be reached at aLigra Systems, 240 N. Jones Blvd., No. 205, Las Vegas, NV 89107-1450 (800/903-6903; fax 800/929-6903; e-mail: carl@acm.org; Web site: http://acm.org/~aligra). ★