NOTE: This article was published in the 1997 special UML issue of Ed Yourdon's American Programmer, in response to a request by Mr. Yourdon. Given the current political-correctness climate in the software industry, we feel it necessary to include the following disclaimer:

This article is a satire and in the time-honored tradition of the pamphlet some of its comments may appear outrageous to some readers. We feel, however, that there is a need to let dissenting views be voiced "as is", without neutering by spin doctors or PR experts.

Needless to say (actually, probably needful to say) this article is its author's responsibility and does not in any way commit ISE's policy on UML. In fact we recognize the success of UML and its endorsement by many organizations; as a consequence, we are actively supporting UML use for those of our customers who have adopted the notation. In particular:

- Our flagship EiffelCase analysis and design workbench offers, starting with version 4.3, the option of a UML look-and-feel, as an alternative to its traditional BON (Business Object Notation) look-and-feel, which will of course continue to be supported.

- As a result of our cooperation with Cayenne Software, we provide a full Eiffel-UML solution thanks to our Cayenne Software's outstanding ObjectTeam UML CASE environment.

So please do not draw any hasty conclusions from this little pamphlet. But do shed a tear or two as you discover the tragic yet true adventures of poor Candide Smith, so intent to please, and so far out of his luck.

From: Candide Smith, OO-101 student
To: Professor Severa Stern
Subject: My request for a grade change

Dear Professor Stern:

The Teaching Assistant for your OO-101 class has instructed me to approach you directly about the D-minus grade I got on my term paper "An evaluation of the proposed Unified Modeling Language (UML)". I hope you will consider changing it to something better -- a
plain D perhaps? -- as it would be a real blow to my Grade Point Average, already not in too good shape after that "Fail" you gave me in your last class. (You may remember that in the final exam I wrote "there may have been other things between sliced bread and Java". I now realize how ill-advised that comment was and sincerely apologize if I hurt anyone's feelings.)

I realize of course the reason for the D-minus and appreciate your generosity in not being harsher on me. As the T.A. pointed out, there was not anything positive about the UML in my paper! Surely that cannot be right. Everyone says the UML is a breakthrough in software engineering, and who am I to question this? That's why I am not asking you to change my grade just because of the effect on my GPA, although I do hope you will appreciate that it's not nice to lose my Good Student insurance rate, not to mention girlfriends and the like. No, I admit I was wrong and I want to make up for it. There must be something good to say about the UML.

And I can assure you I'll remember the lesson: be positive. Whatever the topic, it is always possible to put a favorable spin. This morning's paper even printed an adjective that might be interpreted as not unflattering in an article about Newt Gingrich! Why then not UML?

**Be positive!**

So I have followed the T.A.'s advice. For example there might be nice things to say about the notation itself. It might be simple, useful, convincing, easy to learn. And there is indeed such a notation for object-oriented analysis and design, whose entire set of graphical symbols fits on one page yet covers all the basic O-O system description techniques, and which is particularly good at scaling up for the description of large systems: Waldén's and Nerson's Business Object Notation (BON) as described in their book [3]. Of course UML has none of these properties. In its attempt to show that it has included everyone's pet ideas, it is a chock-full of symbol after bizarre symbol. Just the "Notation Summary" takes up 60 pages and has its own table of contents! UML is in fact as complex as a big and cryptic programming language, with generous use of "$" and "#" and ".-" and "*" and "solid triangles with no tail" and rectangles and diamonds and solid lines and dotted lines and solid ellipses and dotted ellipses and arrows of all kinds and keywords such as "const" and "sorted" (not to be confused with "ordered") and different semantics for a class depending on whether its name appears in roman or italics; but at least a programming language, even the worst of languages, is executable! Here you have to learn all this monstrous complexity just to build diagrams of a possible future system.

Which brings the question of seamless development. Once you have your beautiful (or not so beautiful) diagrams, you will want to build a system, unless the budget has already run out on CASE tools (a common occurrence for companies that take the "methodology" hype too seriously, and end up with no money left for real development). But then you have to start over with a programming language to do your real job. And how do you maintain the consistency between the program and the diagrams? Waldén and Nerson convincingly discuss this goal of seamlessness (providing a single, continuous process) and reversibility (being able to go back and forth between analysis, design, and implementation). Tools such
as EiffelBench and EiffelCase support this concern, but it does not seem to be a concern at all with the UML. But of course anyone using UML must be very smart -- if only to learn all the symbols -- and so will get the analysis right the first time around, then the design right the first time around, and then will never have to change anything in the next ten years of the project's life. Or perhaps UML is for projects whose specifications never change. My grandma told me she had heard about such a project in her youth. I think she said it had something to do with computing the 6th Fibonacci number.

Object-oriented?

So I have had to look elsewhere for something positive to say about the UML. I am trying really hard and I hope you will take all these efforts into account before turning in the final grades to the administration. (Remember, I am only asking for a D, although of course a C-minus would be most gratefully appreciated.) For example I have tried to see if I could characterize UML as "object-oriented"; we all know this is a great compliment. Fat chance. Of course the authors make the requisite use of "object" and "inheritance" and so on. But a mere glance at the diagrams shows UML for what it is: an extension of entity-relationship modeling. The basic examples show binary and ternary associations, such as (page 16 of [1]) associations between "flight" and "seat" and "person"; this is the exact opposite of object-oriented design, where, as we all know, the world is structured in classes, built around object types, and every operation or property belongs to one class. Surely in object-oriented design you can't have a "passenger" link that treats "seat", "flight" and "person" symmetrically! In an object-oriented system it would belong to one of the classes; that's how you obtain the consistency, simplicity, modularity and reusability of O-O architectures; look at BON or at Eiffel to enjoy the results. The authors of UML know this, of course; to understand why they call UML object-oriented we must appreciate their famous sense of humor. Obviously they meant it in jest. (Is this positive enough?)

Further evidence of the joke is provided by the frequent reference to "use cases" as a central element of the method. Back when "use case" was the buzzword of the Web-year I tried to understand what all the fuss was about, and had a hard time until I asked my grandfather, who explained it all to me: it's the new name for the top-down functional design of his adolescence. You look at what the system must do ("use cases"), and deduce the system architecture from that analysis. This is the exact opposite of object-oriented design, which consciously refuses to pay too much attention, during the early phases, to the main function or functions of the system, because they are so subject to change, because they reproduce the behavior of previous systems (those which we are trying to replace with a new system), because they lead to early commitment to the order of operation (the most volatile aspect of the software), and because they focus on superficial properties of a system -- its interface to the rest of the world -- rather than its fundamental properties. Instead, O-O design concentrates on the types of objects manipulated by the system, and defines each one of these types through the list of all applicable operations and their abstract properties -- contracts -- regardless of order of application. My grandpa added that he was glad that use cases were present in the UML, because they brought up memories of the good old days and he had never liked that object stuff anyway. (I think this is a positive comment.)
Finding a use for UML

Obviously the UML will not be useful to any software developer. Could it perhaps be useful to someone else? Project managers, perhaps? Company executives? The companies themselves? Of course not. Indeed the documents make no pretense of such usefulness, and for good reason. It is hard to imagine what benefit a business could get from pages of cryptic diagrams about hazy properties of a poorly understood system.

So I am looking further. Sometimes a proposal offers a flawed solution, but it has the merit of posing the right problem. Can we say that about the UML? I hope so, if only because it would help my GPA, and helping my GPA would help me regain my self-esteem, not to mention girlfriends and the like, but I digress. By the way, what problem if any does the UML address? I tried very hard, in the interminable documents on Rational's Web site, to find a description of the problem (which we have been told in our engineering courses should always appear in the introduction of any technical document); I am afraid I cannot really report any useful find.

Amanda to the rescue

To see what goals could have been pursued, I turned to the term paper of my friend Amanda Lucky, the one that got an A-plus: "Challenges facing the software industry". (By the way, professor, I don't mean to complain but why does Amanda always get the good topics?) That report, very nice I must confess, does a good job of describing the technical obstacles that developers must overcome. Good software must be correct; forget an assertion in a routine, and you can have a $500-million disaster as in the recent Ariane 5 explosion, resulting entirely from a misguided attempt at reusing an Ada routine in the absence of an Eiffel-like assertion mechanism (see the January 1997 issue of IEEE Computer). I don't see anything in UML that helps correctness; some lip service is paid to the notion of contract, but the authors show that they understand nothing of the idea of Design by Contract (for example, they mix up semantic requirements with simple type declarations!). Good software must be robust; how does UML help? Good software must be easy to change (Amanda calls this "extendibility"); but the heavy apparatus of UML all but guarantees that developers who have been able to produce any system description at all will be scared to death of changing anything. Good software must be reusable; nothing in UML addresses the challenges of reuse, such as standardization of interface conventions, option-operand separation, command-query separation and so on. Good software must be efficient -- oh, sorry, this is implementation-related, and we don't talk about implementation in polite company. (If UML addressed implementation it would have to address software issues; the good thing about bubbles and arrows, as opposed to programs, is that they never crash.)

Instead, the UML documents seem to have just one aim: again and again to convince the reader that they are not omitting any of the buzzwords, as in this extract, page 31 of [2], which tries to explain that patterns are interesting but covered by the authors' own concept of "interaction diagram", whatever that is (I couldn't figure it out):
The interesting aspect of many patterns is their dynamic behavior. We can certainly show this with interaction diagrams, but only in the context of a specific model; it is harder to keep the "patternness" of the patterns open. Finally, patterns are templates, in a sense, in which classes, associations, attributes and operations might all be mapped into different names which keeping the essence of the pattern; we need a way to clearly parameterize the patterns. Nevertheless we feel that we have enough facilities to capture patterns within the UML by expressing them in terms of interactions.

Frankly, who is interested in such mumbo-jumbo? How can anyone believe that it has anything to do (even "in a sense") with the software industry? And I haven't even quoted the stuff about the "metamodel". Maybe you could ask Amanda to devote her next term paper to "patternness". Talk about challenges.

A friend who was at an object-oriented conference recently told me of all the UML jokes that O-O experts -- some of the most respected people in the field -- were trading during the breaks and in the back of the rooms. He couldn't believe the contrast between the public hype and the private disparagement. But CEOs and other decision makers only hear the hype; they are told that UML is object-oriented, or more often that object-oriented means UML. When it fails to help their software development, UML will give a bad name to the whole O-O field. Given the marketing blitz around it, UML, by not delivering on its promises, has the potential of delaying the progress of object technology by ten years.

The answer at last?

So, professor, what can I say, what can I say? I went to the T.A. and asked her if "Rational's home page has really nice colors" would do. But she said no, I have to find something of substance. I tried "at least in the latest revision they gave up calling their stuff a method, which shows that they have some sense of ridicule". It didn't pass muster either. So I looked and looked, and finally -- I am so excited -- I found it! Yes! UML has a purpose after all!

The reason I did not discover it earlier is that it only appears in the conclusion. A strange place to describe what one was trying to write about all along, but better than no place at all. Of course what I found is not a technical goal (UML, as I said already, serves no software-related purpose whatsoever, and that's fine with me -- some people have better things to do with their lives than trying to improve software technology). It is not a managerial goal, at least not for people managing software projects. It is not a business goal, at least not for any business using UML. But it is a goal.

When I finally discovered that goal, I was almost driven to tears by the authors' noble and generous spirit. It would be unfair to say that the piles of UML documentation are empty of any substance, when in fact they contain one genuine idea. I found it in the very last paragraph of the very last report describing the 0.91 revision [2]. There it was in all its openness, explaining everything I had theretofore misunderstood in my youthful naïveté.
How could I criticize the method for not helping software developers or managers, when it does not care about software development at all, but only about developing a market for consultants and trainers? Everything started to make sense: the complexity and bizarreness of the notation, which I had foolishly taken for a deficiency, are in fact among its most attractive qualities, since they make for endless business opportunities, for Rational and perhaps even for others as well; so does its lukewarm adoption of object-oriented ideas, since it means a consultant does not even have to like or understand object technology to profit from UML.

My long search had not been in vain. It had led me to a full appreciation of the UML, this admirable self-feeding machine, devoted from A to Z to the creation of a new market, free of any of the difficulties associated with the unpleasant business of software development: UML books! UML courses! Courses on the books! Books on the courses! Books on the books! Introductory courses to prepare for the advanced courses! Courses for those who teach the courses! Revisions! UML journals! Conferences! Workshops! Tutorials! Standards! Committees! T-shirts! The more you think about the possibilities, the more dazzling they look. And for any reader brave or bored enough to read the documents to the end, the grand scheme is all there, laid out in the final paragraph.

Everything was coming into place. With the air of inevitability that reveals the genuine masterpiece, in all the glory of the document's inimitable style, the last six lines suddenly gave sense to the hundreds of vacuous pages before them:

There are several public courses, and we know of several other books that are being written that address different aspects of the UML, all based on our previous preliminary publications. We hope for widespread tool support, training courses, and consultant usage in the future. We strongly encourage this kind of support [How good of you to encourage support for your own products! What selflessness!] and we will work with authors, trainers and consultants to ensure that their questions are addressed so that there will be wide-spread and consistent support for the UML.

Hoping for your favorable attention to my request,

Yours respectfully,

Candide Smith

References
