# The Future of Object Technology

Bertrand Meyer, Eiffelsoft

In the future, object technology will not be confined to a niche. Objects will be pervasive; very little serious software will not be object-oriented at least in some way in 1998 and beyond.

Object technology isn't a matter of fashion. It's simply that no one really knows how to tackle the kind of sophisticated systems that our users now want, without using object technology.

It's also that no one has found anything better. Since object technology came into prominence a decade ago, pundits have at various times predicted its demise, with the inevitable periodic announcements of the so-called "Object Winter," an allusion to the "AI Winter" that froze the spread of artificial intelligence in the late seventies.

But winter has not come. And all the signs indicate that spring will continue.

## ALL THINGS OO

In every area of software technology, OO ideas are at the fore; this is true in programming languages, analysis and design methodology, and in databases, graphics, formal approaches, networking, concurrency, distributed computation, and even Web development.

Areas that had traditionally resisted the influx of OO ideas are no longer immune. Indeed, 1997 may be known in history as the year during which object technology finally reached the world of embedded and real-time systems. Even the scientific computing field is becoming increasingly OO, both with the spread of OO languages and libraries and with the increased influence of object ideas on new versions of traditional languages, most notably Fortran.

Areas that had traditionally resisted the influx of OO ideas are no longer immune.

Ian Graham recently noted in the *Journal of Object-Oriented Programming* that some authors have been citing "componentware" as what will come after objects. But of course object technology has been component-savvy all along. When a headline announces that a technology goes "beyond objects," it's usually to highlight techniques, such as patterns and frameworks, that fit nicely with the rest of the object-oriented approach.

So the main concern of any object enthusiast should not be whether object technology will be around in the future, but whether the OO concepts can avoid the kind of dilution that have plagued structured techniques. If everything is advertised as object-oriented, the burden is on the buyer to ascertain what is OO and what is not.

## OO DATABASES

The future of object-oriented databases is an interesting topic for speculation. It is remarkable to see how relational database vendors—Oracle in particular—managed since 1986 to stifle the growth of OO databases through preemptive announcements, repeatedly convincing customers that their product "is going to be OO next year anyway, on top of offering all the relational facilities."

Ten years later the products did begin to match the announcements, although OODB experts will still tell you that the offerings from the main relational vendors, in spite of their advertised object facilities, are still far from the real thing. In the meantime, however, none of the half-dozen companies that in 1986 hoped to become "Object Oracle" did.

Some applications do require true OODB facilities in fields such as finance, CAD/CAM, simulation, and graphical information systems. As they become more and more ambitious, the market for OO databases will continue to grow, but it will remain a fraction of the traditional database market.

## JAVA AND UML

At the moment, much of the buzz is about Sun Microsystems' Java and Rational's Unified Modeling Language (UML); I don't think either will matter very much in a few years.

The limitations of Java as a language are becoming evident to many people, and its most significant contributions (multithreading, dynamic loading, and Web interfaces) are being transferred to other languages. It is also doubtful whether Java's byte code will, as Sun hopes, become a universal portability vehicle. The technical obstacles (did we hear "performance"?) and the political

**Editor: Bertrand Meyer, EiffelSoft, ISE Bldg., 2nd Fl., 270 Storke Rd., Goleta, CA 93117; voice (805) 685-6869; ot-column@eiffel.com**

ones (did we hear "Microsoft"?) are formidable.

Meanwhile, UML's current success is due to the same factors that have attracted people to such proposals as the Capability Maturity Model and ISO 9000. For decision makers, especially those who are not themselves software professionals and are dismayed by the difficulty of managing software projects and predicting costs and development times, the prospect of industrializing software development through standards is soothing. Unfortunately, UML is too complex to be of much help in practice. An analysis-design method must be simple and easy to learn, so as to facilitate the development process without interposing a formidable wall of new notations and concepts.

This complexity is all the more remarkable when we realize that UML is only a tool for analysis and design. In the end, you still have to write the program.

In many respects, the UML idea goes against the *seamless* nature of object technology, which is designed to reduce the conceptual distance between the various phases of the process, not introduce notations removed from the programming language. So, although UML will be successful at first, because it has the right endorsements, it will be of little use to the actual process of developing software.

It is indeed remarkable to see how few of the officially sanctioned, management-oriented approaches to software engineering have succeeded on a wide scale. At the recent Tools Pacific Conference in Melbourne, two of the invited speakers, Jim Coplien and Doug Schmidt, questioned the value of ISO 9000 and the Capability Maturity Model; this probably would not have happened a few years ago.

One area in which it is risky to make a prediction is that of communication mechanisms. At the moment it seems that both Corba and DCOM are carving themselves respectable niches. Whether this trend continues, or DCOM obliterates Corba, depends in part on whether Unix will find new life or whether Windows will reduce it to boutique status. The signs of a Unix resuscitation are not yet here, but it is too early to call in the undertakers.

Perhaps most importantly, reusability will become more and more the way of life in our industry. As many Object Technology columns of 1997 explained, it is not enough to rush to reuse. The components must be certifiably good, requiring application of the Design by Contract idea and a serious qualification process, based both on tools (for validation and verification) and on people (for deep scrutiny).

If reuse becomes a way of life—and the chances are good that it will—the next few years will see as many advances as those of the past decade, profoundly affecting the software field and providing object-oriented enthusiasts with many new sources of excitement. ❖