
Programming language design sometimes seems to require clairvoyance more than any specific technical talent. By the time a language has been standardized, users have been trained, compilers have been implemented, and support tools and applica-

Benjamin M. Brosgol

tion libraries developed, years (and several hardware generations) have gone by. If the language designer was not sufficiently forward-thinking, the result of his or her labors may be obsolete by the time it is mature.

Ada

(c) The Stock Market / Chris Jones, 1992

Ada

With Ada's 10th birthday approaching, it is fair to ask whether this language has managed to avoid such a fate, and if so, how. The accompanying articles address this topic from several angles. Schonberg, Gerhardt and Hayden set the stage with a technical overview of Ada. They show Ada as a Pascal-based, strongly typed, general-purpose language—but with features going well beyond its ancestor in a number of areas including packages for system modularization, private types for data abstraction, tasking for concurrency, exceptions for dealing with run-time errors, and generics for reusability and general parameterization. The idea was to build software engineering support into the language rather than relying on programming style or operating system services.

No previous language had the ambition (some would say “temerity”) to attempt combining such a set of facilities into a single coherent whole. You will have to judge for yourself whether the melding has been successful, but the fact that later languages have borrowed and either adopted or adapted many of Ada's concepts is evidence that the language was, if anything, ahead of its time.

As many know, the Department of Defense (DoD) sponsored the development and standardization of Ada. To avoid the problems with uncontrolled proliferation of languages and dialects that were all too typical before Ada, and to facilitate programmer and source program portability, DoD policy prohibits subset and superset implementations. It also includes a mandate for defense software developers to use Ada. No such mandate is in effect at colleges and universities, where Ada must vie with other languages. In fact, it is useful to see how Ada is managing in this competition. The issue is not simply theoretical. A language ignored by the academic community will find it difficult to reach critical mass with respect to trained programmers, textbooks, and so on.

The Feldman article provides a snapshot of usage in undergraduate curricula. From the perspective of a college professor deciding on a language for an introductory computer science course, Ada comes with several handicaps. Economically, the cost of Ada compilers has historically been higher than other languages. Culturally, Ada's DoD heritage is not necessarily an advantage on college campuses. Conversely, it is widely recognized that some alternative to the traditional introductory language—Pascal—is needed, since even second-term students find Pascal is just too “small.”

The question for teachers is how to present “large” languages in appropriately small doses. Ada does not look at large as it once did, compared with extended Pascal dialects and C++. Accordingly, the Feldman article shows that Ada's use as a foundation language is increasing steadily with over 70 colleges and universities now using it for this purpose. The sidebar by McCormick relates how one university uses Ada for teaching the principles of real-time programming.


If structured programming was the watchword of a previous generation of programmers, object orientation is surely today's mantra. Rosen's article examines the technical

aspects of object orientation through a contrast between the development methods of classification (via inheritance) and composition, indicates the choices made by Ada, and discusses the benefits and disadvantages with respect to program maintenance.

Interestingly, the main concepts of object-oriented programming (OOP) are hardly new and, in fact, predate Ada. Simula offered classes, inheritance, polymorphism and dynamic binding more than 25 years ago. Their omission from Ada was by intent rather than by accident. The Ada designers had both methodological and performance concerns with the dynamic aspects of OOP and chose to provide a simpler development paradigm known as object-oriented design (OOD). The Rosen article argues that, despite all the publicity touting OOP for modern software development, Ada's simpler OOD paradigm is as effective given the framework the language supplies for software engineering support.

No programming language can remain static and still respond to the rapid pace of technological change that characterizes the computing industry. Standards organizations recognize this reality and have formal processes to ensure that languages are updated periodically, reflecting user and implementer experience and responding to new requirements. Ada is no exception. Indeed, a project known as Ada 9X is currently under way to revise the language and obtain an approved standard. The intent of Ada 9X is to provide enhancements to Ada that improve its support for its original application area (real-time systems) while also increasing its attractiveness in other domains.

Three contributions in this section focus on Ada 9X from different vantage points. Taft's article provides a technical overview of the main features of the revision, summarizing the new linguistic support for OOP, the hierarchical package model for more general modularization, and the protected type feature for efficient data-oriented synchronization. Anderson's article gives a view of 9X from her perspective as the project's manager, and Ploedereder's piece describes the consensus-building process associated with 9X.

In summary, the Ada articles in this section provide a view of what the language is about, how it is used in academia, how it relates to the concepts of object orientation, and what is in store for the future. They reveal a language that is neither the silver bullet that early hype unrealistically promoted as a cure for all software ills, nor the over-engineered niche language that its detractors unfairly present. Rather, Ada is a significant contribution to computer software technology whose role promises to grow increasingly important as the sophistication and complexity of applications increase. With a cover article entitled “Computing the Future,” the inclusion of a special Ada section in this issue of *Communications* is altogether appropriate. 

Guest Editor **Benjamin M. Brosgol**, an Ada consultant and educator, has been involved with the language's development and implementation since 1975. He supervised the “Red” language team at Intermetrics—runner-up in the Ada language design competition. He was also a founder of Alsys, Inc., where he served as Vice President/Technical Director. **Present Address:** Brosgol Consulting and Training, 79 Töbey Rd., Belmont, MA 02178, brosgol@ajpo.sei.cmu.edu