Ada in the 21st Century

Benjamin M. Brosgol
Ada Core Technologies

Although Ada does not receive the publicity of some other languages, it is alive and well in a broad range of applications that include hard real-time embedded systems, safety critical programs, desk-top graphical user interface environments, and the Internet. The first internationally standardized object-oriented language, Ada is a language of choice in many environments where reliability is critical. It continues to play a significant role in teaching and research at computer science departments around the world. The Ada infrastructure, including the validation testing for implementation conformance with the standard, has been successfully transitioned from the government to the private sector. The language is evolving smoothly to meet the demands of the 21st century.

Editor s Note:
When Donald Reifer's *Is Ada Dead or Alive Within the Weapons Systems World* was published in December 2000, we said, "Ada has been surrounded by controversy almost since its inception. In this issue we offer one perspective on the current state of Ada and how this affects technology decisions for weapons systems. An upcoming issue will provide an opposing point of view." Here is another Ada report. Letters to the editor follow.

The U.S. Department of Defense (DoD) had two major goals when it sponsored development of the Ada programming language in the early 1980s. First on the technical side, Ada was designed to meet the requirements of large-scale, reliability-critical applications (specifically, embedded real-time systems). Second, Ada was intended as a common DoD language to reverse the language proliferation trend that was making source code portability all but impossible. Ada was the product of an international effort led by a team from France and involving representatives from government, private industry, and academia.

Ada became an international standard (ISO) in the mid-1980s and was revised (with full support for object-oriented programming) in the mid-1990s. Its advocates observe that Ada has met its initially intended goals, and that it is an effective and methodology-neutral language for teaching software engineering. Ada implementations, tools, and libraries are available on a wide variety of platforms through a number of vendors. The language is especially applicable to safety-critical and high-integrity applications, and it is heavily used in industries such as transportation. Ada offers more security than languages such as C and C++, and better efficiency and a simpler run-time model than Java. Indeed, an impartial observer, the respected real-time authority E. Douglas Jensen, offered this technical assessment:

> "Ada 95, including its Real-Time Systems Annex D, has probably been the most successful real-time language in terms of both adoption and real-time technology. One reason is that Ada is unusually effective (among real-time languages and also operating systems) across the real-time computing systems spectrum, from programming-in-the-small in traditional device-level control subsystem, to programming-in-the-large in enterprise command and control systems [1]."

Nevertheless, Ada has not enjoyed the commercial success or publicity of other languages. Occasionally the question is posed whether Ada is "dead or alive." This article offers a snapshot of the Ada industry as of late 2000, concludes that Ada is indeed alive (especially in the domain for which it was initially intended), and offers some predictions on its future.

The first several sections of this article provide some perspective: reviewing the history of the DoD's Ada policy, showing how the choice of a language affects productivity, and addressing the issue of commercial penetration as a factor in choosing a language. The paper then summarizes the supply side of Ada (compiler vendors, tools, bindings, etc.), looks at how Ada is being used today, identifies the main reasons it is chosen, and describes the lessons learned from user experience. Lastly, the article discusses Ada's role in teaching and academic research.

Ada Policy in the DoD

Ada is one of a small number of programming languages that is accompanied by a comprehensive test suite that reflects a compiler's conformance with the language standard. This suite, originally called the Ada Compiler Validation Capability and more recently designated the Ada Conformance Assessment Test Suite, comprises several thousand individual tests that exercise the core language, the standard library, and the specialized needs annexes. A compiler that successfully passes all applicable tests in the current suite is labeled *validated.* Although the evolutionary nature of the test suite precludes trying to standardize the tests themselves, the test procedures became an ISO in 1999.

From 1983 until 1997, the DoD's Ada Joint Program Office promoted a stringent Ada usage and validation policy requiring Ada on weapons systems applications and specifying that any project using Ada employ a validated compiler to produce delivered code. The policy had dual consequences. The Ada mandate seeded the market encouraging hardware vendors to supply Ada implementations and spurred a furious competition among Ada compiler vendors. However, it was imposed before Ada environments had become mature. Thus early experiences were often disappointing, especially given the pent-up demand and overly ambitious expectations that often accompany a new technology. A required waiver for not using Ada—and justifying the decision on projected cost saving—was widely ignored. Moreover, if an Ada project failed, it was convenient, even if not often justified, to blame problems on Ada language or implementation.

The validation policy also had mixed effects. On the positive side, it forced compiler vendors to focus on implementing the full language versus a subset. This was instrumental in realizing a high

degree of portability for Ada source code across a wide variety of platforms—a benefit that should not be underestimated. Ada has had an excellent track record for source code portability. In some areas such as task scheduling, it is significantly more portable than Java despite the latter's claim of "write once, run anywhere."

At least in early Ada industry, achieving necessary validation sometimes led to distorted priorities as compiler vendors paid less attention to other important traits such as efficient run-time performance and support for external tools. By the mid-1990s the DoD took a new look at compiler validation. They recognized its importance in preventing the proliferation of subsets and dialects—real problems with earlier languages. But they perceived this was no longer a critical risk since the range of language choices was much narrower.

The DoD ultimately rescinded both the Ada mandate and the validation requirement in April 1997. Ironically, Ada was required before it was ready, and then the requirement was removed once implementations had matured. Concluding that the Ada language no longer required external support, the DoD subsequently dissolved the Ada Joint Program Office. Ada infrastructure elements—maintenance of the validation test suite and oversight on the language's evolution—were taken over by the Ada Resource Association, a vendor trade group described below.

Programming Language Effects

A programming language is not a magic bullet that will prevent or cure software ills. It should be regarded as one element of technology used in a software project that is complemented by additional factors: tool support, people issues like developers' talents and ability to work on a team, and the efficacy of the development process. Indeed, it is sometimes argued that the choice of a specific language is one of the less critical factors in terms of effect on project outcome.

A good language will not turn unskilled developers into software wizards, but it can leverage the talents of a skilled team making them more productive. Sometimes an inspection of technical features makes this obvious. For example, Ada allows the integer value 2^{63}-1 to be expressed in hexadecimal format as 16#7FFF_FFFF_FFFF_FFFF#, which is easy to read with the underscore character serving to separate the 4-digit groups. In contrast, C (and also C++ and Java) require the same value to be written much less clearly as 0x7FFFFFFFFFFFFFFFL. The absence of a separator character in the number is of no consequence to a compiler but is an invitation for human error.

It is difficult to conduct comparative quantitative studies of programming languages in an objective manner, especially for large projects. However, an impartial and thorough analysis [2] documented a significant difference in productivity between Ada and C on the components of the Verdix VADS product line. This development comprised roughly the same amount of code in the two languages. Zeigler's study considered all the relevant factors (such as the effect of programmer skills) and concluded that Ada performed approximately twice as well as C (i.e., costs for Ada were half that for C). Some of the reasons cited were Ada's additional compile-time checking, its higher-level features, and the Ada culture that encouraged up-front design. Interestingly, the study observed that using C++ rather than C would not change the underlying result, since bug rates in C++ were higher than in C. It concluded:

> "Our data indicates that Ada has saved us millions of development dollars. For every development dollar, we could make a case for another three dollars for customer support, sales, marketing, and administration costs ...."

This result should not be surprising since Ada was specifically designed to save life-cycle costs through software engineering support. Quantitative data provides evidence that this goal has been met, even if percentage of improvement varies per project.

Commercial Penetration

For a variety of reasons Ada has not had the commercial penetration of other languages. This raises an issue: How important is a language's popularity when an organization needs to choose a technology for future software development? A recent article [3] argued that commercial success should be a guiding factor in language choice for large-scale DoD weapons systems. We believe this is the wrong way to look at the issue for several reasons:

- Commercial success is almost always more strongly influenced by marketing factors than by technical characteristics. From a software engineering perspective, Algol 60 and Pascal were better languages than Fortran and BASIC but did not achieve the latter pair's high market share. An enterprise needs to look beyond the popularity of a language and make sure that it satisfies the technical requirements of the application.
- In the commercial software market the way to succeed is to release products that are good enough, at a price that customers will accept, in a timely fashion. In a market-oriented software industry, reliability is not the most important criterion. This is reasonable for many kinds of applications; if a Web browser locks up or a word processor crashes, the damage is relatively localized. But weapons systems do not have such luxuries.
- Times change and what is popular one day may be a nostalgic memory the next. If commercial penetration had been the deciding factor in earlier times, then weapons systems would have chosen COBOL during the 1960s and 1970s, and BASIC in the 1980s.
- Relative market share data can lead to the erroneous conclusion that non-market-leading technology has failed. This is partly due to the media's tendency to portray commercial competition as a sports event with only one winner. In fact, a number of technologies that do not receive much publicity today are indeed financially successful: Examples are OS/2 in the operating systems arena and PL/I and Pascal in the language area. Judging from the infrequent press coverage these technologies receive, one would deduce that they are all but extinct. Yet in fact they are still in heavy use.
- The market dynamics of a commercially popular product sometimes conflict with the requirements of a major long-term project such as a weapons system. To stay ahead of the competition, a tool vendor needs to upgrade and enhance its product at regular intervals. However, developers cannot afford the risks that disruptions of the underlying tool base would bring. They instead need to baseline a particular version for use

throughout the project (or at least until a major upgrade). Support for older versions is not generally a high-priority item for vendors whose products have a high market share.

Perhaps most importantly, commercial penetration itself is not what is important, but rather the effects it seems to bring—an adequate and varied supply of compilers and tools[1], and a pool of programmers already skilled in the language. Let's look at how Ada fares in regard to these factors.

The Commercial Players

Principal Ada compiler vendors are the members of the Ada Resource Association trade group: Ada Core Technologies, Aonix, Averstar, DDC-I, Green Hills Software, OC Systems, and Rational Software Corp. Several smaller companies such as Irvine Compiler Corp. also sell Ada compilers into specialized markets. All of these companies are established players in the field. The oldest were founded in the early 1980s, and the youngest was formed in the mid-1990s. The companies vary in their business models, with different mixes of off-the-shelf products, contract work, and support/consulting services.

Although the number of Ada compiler vendors is smaller today than it was 10 years ago, this is a common phenomenon throughout private industry as mergers and acquisitions have become standard business practice. Moreover, there is not a large number of compiler vendors for more widely used languages.

The Ada compiler industry today can be characterized as fairly stable with competition on the most popular platforms. The size of the Ada market is difficult to quantify with any precision. Based on an informal analysis by the Ada Resource Association and public data provided by some vendors, the worldwide market is about $80 million or so annually. This figure has been fairly steady during recent years and is not likely to change significantly in the near term.

Ada is available across a wide range of platforms. A partial list of Ada '95 compiler host environments includes Compaq/Digital Alpha (OpenVMS, UNIX), Concurrent/PowerMax, HP9000/HP-UX, IBM 390/MVS, Intel x86 (Windows-NT/9X/2000, Linux, OS/2, UNIX), PowerMac/Tenon, PowerPC/AIX, RS6000/AIX, SGI/IRIX, Siemens-Nixdorf/SINIX, SPARC/Solaris, and Vax/OpenVMS.

Target environments include all the above hosts plus ADI-SHARC/EONIC, ADI-21020/Bare, HP7xx/HP-RT, IBM390/CICS, i960/HAOS, Intel/ETS Intel/RTLinux, the Java Virtual Machine, M68k/VXWorks, MIPS/VXWorks, Nighthawk/6800, PowerPC/Bare, PowerPC/LynxOS, PowerPC/VXWorks, and also optimized ANSI C.

Several compiler companies use automated code generator technologies and layered run-time systems with a clear interface to target-dependent components. This makes it relatively easy to produce Ada compilers targeted to new chips and operating systems/real-time kernels. Implementing Ada on a new target is roughly comparable in effort to implementing C++ (except for needed concurrency support in the run-time system since C++ lacks this facility).

Ada compilers are supplemented by an assortment of productivity-enhancement tools, component libraries, and bindings to popular software systems. The compiler developers supply some of these: source-level debuggers, graphical user interface (GUI)-based tool environments, browsers, tools for memory monitoring, GUI builders, and target OS interfaces. Third-party developers provide others. These products include source analyzers (DCS, McCabe & Associates, Vector Software), formal verification tools (Praxis Critical Systems), automated design tools (TNI), interfaces to X-Windows (TopGraph'X), Ada support for CORBA (TopGraph'X, Objective Interface Systems), real-time embedded graphics (DCS), a *thick* binding to the Windows API (RR Software), and many others.

The availability of Ada Core Technologies' GNAT Ada compiler under the General Public License (GPL) of the Free Software Foundation has inspired the development of a variety of tools and bindings that are also available under the GPL. A number of these are available though the Ada Power Web site (see On-Line Resources). Examples include a COM/DCOM/COM+ framework and bindings to the Windows API.

A wealth of libraries is available in other languages: mathematical algorithms in Fortran, low-level communications functions in C, network-ready classes in Java, etc. This has been cited as an Ada weakness, but in fact one of Ada's unique strengths is the ability to interface with software in other languages in a standard and straightforward manner. Tools for several languages are available that automatically generate the appropriate glue specifications that provide the Ada interface to foreign components. If you need to mix C and Fortran, this is easier in an Ada environment than in either a C or a Fortran compilation system.

Ada is unique among programming languages in having a standardized interface for tool developers: the Ada Semantic Interface Specification (ASIS). This high-level interface contains relevant information about an Ada program in a format that is convenient for processing. Several of the tools previously mentioned work from an ASIS version of the source program. Moreover, most Ada 95-compiler vendors have made an effort to integrate their development environments with non-Ada tools such as configuration management systems. This results in Ada development environments that are as comprehensive as other languages with higher commercial penetration.

Another side effect of a language's market share is the supply of skilled programmers. It is a tautology to observe that the more popular the language, the higher the number of programmers who know it. How significant is this issue for a project that will continue over many years with a large team that produces perhaps a million lines of code?

On one hand it cannot be denied that starting with a team already familiar with the programming language will save some up-front costs. But this is largely a red herring issue. Most large projects need to reserve time for new team members to assimilate possibly unfamiliar technology. Any programmer who is skilled in C or C++ can come up to speed in Ada through a variety of approaches, including on-line tutorials, books, or professional courses. A professional programmer in any language can become Ada-proficient in a five-day course. In the process he or she will also learn software development techniques (package design, use of tasking features, and reuse through generics) that carry over to other languages.

While Ada may not have the commercial penetration of

other languages, this should not be a major factor in choosing it for a large, long-lived project. Indeed, basing a decision on commercial popularity may even increase some risks due to the mismatch between market dynamics of the software industry and the requirements of enterprise-critical software. Ada tools and environments are also mature, competitive, widely available, and the apparent training gap in the supply of professional programmers is a problem that is easily addressed.

Ada Usage

Ada applications range from hard real-time processing to commercial desktop tools. Ada users include small start-ups, large established firms, all points in between, and encompass government agencies, the private sector, and academia around the globe.

The language has shown particular strength in the safety-sensitive domain, especially in the transportation industry. Boeing used Ada heavily for their 777 aircraft and continues to require it for all software of the highest safety criticality certification levels—DO178b levels A and B. Ada has been used on subway systems, including the London Jubilee, and the recent extension of the Paris Metro, and for work on the Carnarsie line of the New York City subway. It has been used on the French TGV trains and metrorail systems in Europe, Asia, and Latin America. It has also been used in commercial shipboard control.

Other domain uses include the TV and entertainment industry, medical computing, communications network switches, and financial and information systems. It has been used in industrial control, including safety-critical nuclear reactor shutdown, and in desktop software. It has of course been used traditionally for military programs in the United States and allied countries. Ada has also seen heavy use by nonmilitary governmental agencies such as NASA and the European Space Agency. There are hundreds of millions of lines of Ada code in operation today worldwide and in outer space.

It is beyond the scope of this article to document even a small fraction of these applications in much detail. Nevertheless, several consistent themes emerge among the reasons that Ada has been chosen, and from the experience and lessons learned that have ensued. Further information may be found through links at Professor Michael Feldman's Web site (of the George Washington University), and the sampling of success stories on the Ada Resource Association's Web site (see On-Line Resources).

Why Ada?

Since its inception, the decision to adopt Ada has been a conscious choice commercially, and also for military applications after the Ada mandate was eliminated. In all cases the main reasons cited are the same: Ada was deemed to be a more reliable language than the alternatives, with run-time performance meeting the efficiency requirements of the application, and sufficiently mature compilers and support tools.

In some cases, bad experience with buggy or nonportable software written in other languages made Ada an attractive choice. In other cases, particular Ada functionality guided the decision (e.g., concurrency support, low-level and real-time features, or interfacing facility). Its status as an internationally standard language, backed by a validation suite measuring a compiler's compliance, has also been a relevant factor.

These reasons, especially Ada's security and focus on checks at all levels, have been convincing factors in the safety-critical and high-integrity software domain. A number of applications such as the French TGV rail system have found that Ada and formal methods make a happy marriage. Several Ada compiler vendors directly support safety certification through certified run-time kernels and other means. Ada is at the forefront of safety-critical technology with its Ravenscar profile [4], a reduced set of tasking features whose implementation can be certified against the highest levels of safety criticality. This profile strikes a delicate balance. It is restricted enough to allow a simple, efficient, and certifiable implementation, yet powerful enough to express common real-time idioms such as periodic and event-driven activities.

The safety-critical market is a small but important sector, and one where Ada continues to see strong interest.

Experience and Lessons Learned

Users have reported that expected Ada benefits have largely been realized: fewer bugs, easier maintenance, and higher portability than other languages. If there has been any disappointment, it has generally not been with the language or the compilers and tools, but rather with Ada's slow rate of commercial penetration and the resulting smaller supply of Ada-knowledgeable developers.

Projects that have chosen Ada tend to continue using the language as their systems evolve. This is neither surprising nor unique to Ada; if a technology works, there is no compelling reason to spend time and money converting to something else. A corollary is that many Ada 83 projects are continuing with Ada 83—using a baselined compiler or an Ada 95 compiler with an Ada 83 option—rather than moving to Ada 95, although Ada 95 is generally used for major upgrades.

Ada and Other Technologies

User experience shows that Ada fits well with modern technologies. In distributed applications and components Ada is supported for CORBA, and also for frameworks such as Microsoft's COM, DCOM, and COM+. In the free software/open source community, the GNU Visual Debugger, a new tool that will be part of the standard GNOME desktop environment, has an Ada graphical toolkit as its basis. In the safety and security domain, a seminal report on the relationship between Ada language features and techniques for integrity assurance identifies how the level of certification can affect the choice of features, and vice versa.

Ada historically and presently continues to influence many other technologies. Its exception handling, generics, and packages affect the design of exception handling (in C++, Eiffel and Java), and templates and namespaces in C++, respectively. Ada's real-time features directly influenced the real-time extensions proposed for Java. The Ravenscar profile influenced the Java real-time core High-Integrity profile. Ada's picture string localization affected choices made in COBOL, and the structured Ada syntax influenced PL/SQL.

Ada and Academia

A combination of factors affects the selection of a programming language for teaching and research. Several are technical:

Pedagogy. Does the language reflect sound software engineering practice (encapsulation, abstraction, object orientation, genericity, etc.)?

Teachability. Can the language be partitioned such that simple concepts can be covered first and more advanced topics later, with a minimum of forward references? Are textbooks and supporting educational material available?

> "Although the DoD removed the Ada mandate in 1997, the Ada market has been fairly stable in recent years. Projects using Ada have tended to stay with the language as their systems evolve. Ada works, and the job of converting to another language is not worth the cost."

Applicability. Does the language reflect the state of the art in the software industry and facilitate interoperability with elements such as modern windowing systems, network software, etc.?

Generality. Does the language span a variety of domains (such as systems programming, real-time applications, enterprise software, etc.)? Does it fit in with current research areas (formal models, proofs of correctness, parallelism, and distributed computing)?

Portability. Are programs easily ported across different hardware platforms (for example UNIX, Linux, and Windows)?

Tool Quality. Are compilers and supporting tools available that are applicable in a teaching/research setting (for example, with good diagnostic messages, an easy-to-use interactive development environment, and access to the source code of the components)?

Other factors are nontechnical:

Marketability. Will learning the language increase students' employment prospects?

Price. Are low-cost or free compilers and environments available?

Against this backdrop it is useful to look at Ada's history in academia. During the requirements phase and the development and review of the preliminary designs from 1977 to 1983, the academic community was heavily represented. Consultants from computer science departments at major universities in the United States and abroad helped shape the final design. There was some hope in the DoD that Ada would replace Pascal as the dominant language for teaching introductory programming.

On the technical side, Ada offered significant benefits such as language support for encapsulation, concurrency, and genericity, and a standard input-output library. However, several factors prevented Ada from realizing widespread penetration. The compilers available for Ada 83 tended to be expensive. Although the major compiler vendors gave academic discounts, the price still tended to be too high for most university budgets. Also, Ada was not widely supported on one of the primary machines then used in academia, the Macintosh.

Ada 83 did not support subprograms as data objects or parameters, making it complicated to express applications such as mathematical integration and awkward (and nonportable) to realize callback. As the 1980s drew to a close a language named C++ began to attract attention as a way to bring object-oriented programming (OOP) into the mainstream. Ada 83 supplied the major elements of object orientation but intentionally, in the interest of avoiding the need for implicit storage reclamation, fell short of full support.

Realizing the importance of penetrating academia, the Ada 95 effort attempted to address these issues. Complete support for OOP was provided; whatever can be done in languages such as C++, Java, and Eiffel can be done in Ada 95. Subprograms in Ada 95 are first class data objects, and the common callback idiom is easily and portably expressed. Most importantly, the Ada 95 project sponsored the development of the original GNAT compiler at New York University, based on the Free Software Foundation's General Public License. Thus an open-source Ada 95 compiler, free in both pricing and usage senses, was available when the language was standardized in late 1994.

Ada 95 has addressed the necessary technical issues for success in academia. It reflects sound methodology and, unlike Java, supports several traditional approaches and not just OOP. Indeed, it was the first widely-used language to be designed based on software engineering principles. The language can be effectively taught in several tiers: introductory concepts (the "Pascal subset" of data types and data structures, subprograms/algorithms); encapsulation; abstraction (generics); concurrency; and OOP. A number of high-quality books are available, including several targeted at universities. A CD-ROM with Ada resources is distributed annually by the SIGAda technical society. Ada is an ISO and a highly portable language. Free or low-cost (but high-quality) compiler implementations are readily available on platforms common in the academic community.

These traits have made Ada 95 an attractive option at a large number of colleges and universities around the world, and usage in computer science programs appears to be fairly stable. According to data gathered by Professor Michael Feldman (See On-Line Resources), approximately 150 institutions worldwide cover Ada in their computer science curricula, around 75 percent introduce Ada in a foundation course, and the remaining 25 percent cover it in an upper-level course. These figures have been consistent since 1997.

Of course, criteria beyond technical features play a role in language selection. The most obvious factor in the past 10 years has been the rise in usage of C++, and more recently Java. Universities often try to ensure that their offerings are relevant in the job market and thus will tend to teach subjects and technologies that reflect trends in the broader computer industry. This is not a new phenomenon; in the 1970s Fortran was heavily used for teaching introductory programming in universities even though Algol 60 was arguably the better choice pedagogically. It is thus not surprising now to see heavy adoption of widely-used languages such as C, C++, and Java in computer science curricula. Nonetheless, considering the substantially greater publicity that these languages receive, Ada continues to be a language of choice for teaching, research, or both at many colleges and universities.

Experience teaching Ada has been positive. In an article [5] comparing the performance of students in different introductory computer science courses at the U.S. Military Academy, the authors observed that students did better with Ada than with

Pascal, and pointed out that this experience was duplicated at the U.S. Air Force Academy. Another article [6] relating the author's experience conducting a real-time embedded systems lab course reported that the students were far more successful in Ada than in C.

Beyond its advantages as a teaching language, Ada is also proving a useful vehicle for research. Here are some examples:

- *Ada and Real-Time Systems*: Florida State University, York University (U.K.), the Technical University of Madrid (Spain), and the Naval Postgraduate School.
- *Ada and Distributed Technology*: University of Brest (France).
- *Formal Methods and Ada for Safety Critical Software*: Uppsala University (Sweden).
- *Tools and Components*: U.S. Air Force Academy

In summary, although Ada is not about to overtake better-known languages in academia, it has a strong and energetic following and is not about to disappear. Ada educators have been regularly providing experience reports at the annual Association for Computer Machinery Computer Science Education conference. Computer science faculty members are aware of Ada and its role in the curriculum.

Conclusion

Although Ada does not receive the publicity of other languages, it continues to play an important role in the software industry, both directly (through actual usage) and indirectly (through effects on other technologies). Ada has proved to be particularly strong in long-lived, reliability-intensive applications. This is hardly a surprise since the language was initially designed for this area. Although the DoD removed the Ada mandate in 1997, the Ada market has been fairly stable in recent years. Projects using Ada have tended to stay with the language as their systems evolve. Ada works, and the job of converting to another language is not worth the cost.

As we look toward the future, there are reasons for optimism. Arguably the period when Ada was most at risk was just after the Ada Joint Program Office's closing. There was uncertainty concerning the support of the necessary infrastructure for Ada's continued evolution. But the Ada Resource Association has successfully taken over this support role, in a smooth transition of responsibilities from a government organization to the private sector.

The Ada standard continues to evolve in an orderly fashion. The Ada Rapporteur Group, a collection of language experts in ISO's Ada Working Group, is considering a number of proposed extensions, including a mechanism that will make it easier for Ada to interface with Java classes.

The key to significant new growth for Ada is expansion in academia. Ada is well poised to make new inroads with documented advantages as a language for teaching software engineering, a track record of success as a vehicle for research (especially in the real-time domain), and with an open source Ada compiler technology available. In short, Ada has fulfilled the goals that the DoD had established for it at the outset of the project almost 25 years ago. It promises to continue that fulfillment in both the near term and long range.u

References

1. Bollella, G., et al., *The Real-Time Specification for Java*; Addison-Wesley, 2000.
2. Zeigler, S.F., Comparing Development Costs of C and Ada, March 1995 [www.adaic.com/docs/reports/cada/cada_art.html].
3. Reifer, D., et al., Is Ada Dead or Alive within the Weapon System World?, CROSSTALK, December 2000.
4. Burns, A. The Ravenscar Profile; *Ada Letters*, Vol. XIX, No. 4 (1999), pp. 49-52. Available as www.cs.york.ac.uk/rts/papers/p.ps
5. Hamilton, J.A., Jr., J.L. Murtagh, J.L., Zoller R.G. Programming Language Impacts on Learning, *Ada Letters,* Vol. XX, No. 3, Sept 2000, p. 18.
6. McCormick, J., Software Engineering Education: On the Right Track with Ada, *Ada Letters*, Vol. XX, No. 3, Sept 2000, pp. 47-48.

Note

1 A choice of suppliers is not always a consequence of commercial success; an obvious illustration is a proprietary but widespread product such as Microsoft's Windows.

Additional Readings

- N. Audsley, *Ada Yearbook Millennium Edition,* Ada Language U.K. Ltd; York U.K., 2000.
- Language Impacts on Learning*, Ada Letters*, Vol. XX, No. 3, Sept. 2000, p. 18

On-Line Resources

- ACM SIGAda technical society: www.acm.org/sigada
- Ada Resource Association: www.adaresource.org
- David Botton's Ada Power site: www.adapower.org
- Professor Michael Feldman's summary of Ada usage: www.seas.gwu.edu/~mfeldman/ada-project-summary.html
- Professor Michael Feldman's summary of Ada in academia: www.seas.gwu.edu/~mfeldman/ada-foundation.html
- Usenet newsgroup: comp.lang.ada

About the Author

Benjamin M. Brosgol is a senior member of the technical staff of Ada Core Technologies Inc. with more than 25 years' experience in the software industry. He has been involved with Ada since its inception, as a language designer, educator, implementer and user; he was the principal author of the Information Systems Annex in the Ada 95 standard. Brosgol is currently chair of the Association for Computing Machinery's Special Interest Group on Ada. He has presented Ada papers and related technologies at conferences both in the United States and abroad. He was awarded a Certificate of Distinguished Service and a Certificate of Appreciation by the Department of Defense for contributions to the Ada language effort. He has a doctorate in applied mathematics from Harvard University and a bachelor's degree in mathematics from Amherst College.

Benjamin M. Brosgol
Ada Core Technologies
79 Tobey Road
Belmont, Mass. 02478
brosgol@gnat.com