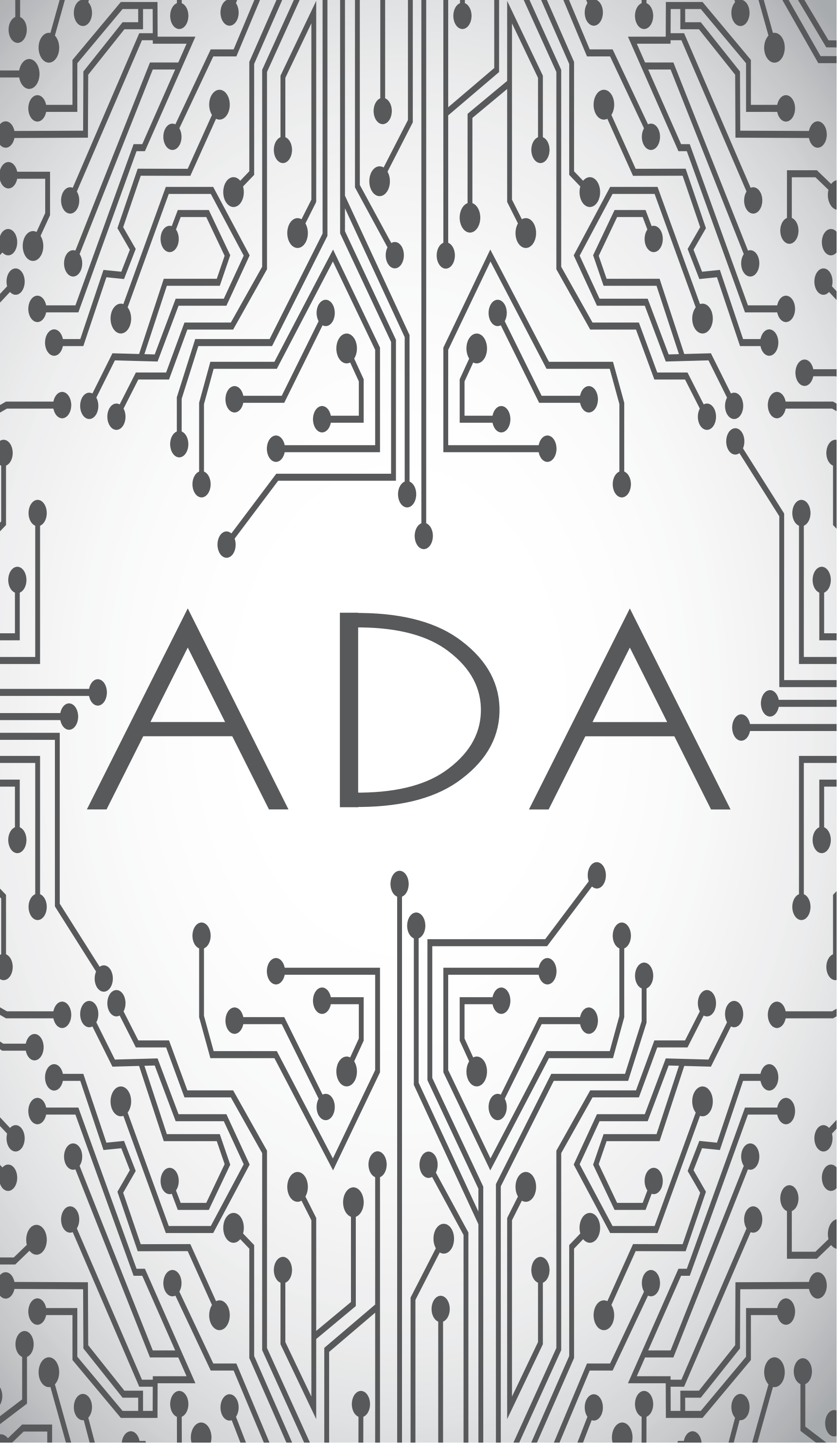

20 Years After the Mandate

By Drew Hamilton and Patrick Pape

In January 1997, the first author was seconded from the U.S. Military Faculty to serve as the chief of the Ada Joint Program Office (AJPO). The spring of 1997 was very eventful for the AJPO and DoD support of Ada. By May 1996, there were well-founded rumors that the Defense Information Systems Agency (DISA) intended to discontinue supporting the Ada support activities of the AJPO. The Computer Science and Telecommunications Board of the National Research Council made a compelling case for continued support of Ada and the AJPO in their significant work entitled "Ada and Beyond." ("Ada and Beyond" may be downloaded for free from the National Academies Press at http://www.nap.edu/download/5463#.) On April 29, 1997, Lieutenant General Emmett Paige, Jr., ended the policy mandate requiring use of the Ada programming language.

The history of the design of Ada and the selection of the "green" language in 1979 is well-documented elsewhere. Prior to the Ada Mandate it was estimated that there were more than 450 programming languages in use in the DoD. [1] It was common to develop a unique operating system and programming language for a specific system. A reasonable estimate of programming languages in use by the DoD in 1997 was less than 50. [1][2] So, on a simplistic level, it was argued that the original mission of the AJPO had been accomplished. The closure of the AJPO then proceeded rapidly.

From the AJPO perspective, this all happened quite quickly, and there were some harsh lessons to be learned.

1. The usefulness of a policy varies inversely with the size of the policy domain.
2. Money is always a factor.
3. Attempts to make software a commodity were and still are premature.

In 1997 it was projected in "CrossTalk" that Ada would still be around 20 years later, even if no new programs were written in Ada because of the critical mass achieved. [2] (Although the paper "Why Programming Languages Matter" is too old to be in the current CrossTalk online archives, it can be downloaded from http://www.drew-hamilton.com/pub/Why_Programming_Languages_Matter.pdf)

Almost 20 years later, the trends forecast in that paper have proved correct. Rather than rehash what was already written, this paper will focus on what was not foreseen in 1997.

It is very difficult to prescribe technical policies for an organization as large as the Defense Department. As noted in the Ada Information Clearing House Archives, the Ada Mandate went into effect on June 1, 1991, and read as follows: "Notwithstanding any other provisions of law, where cost effective, all Department of Defense software shall be written in the programming language Ada, in the absence of special exemption by an official designated by the Secretary of Defense." [3]

The mandate was sound as written, but its implementation varied greatly across the DoD. There were certainly cases where Ada was not the most effective choice from an engineering design perspective as well as from a cost perspective. Early Ada compilers could be extremely expensive, particularly compared to compilers written for C, Pascal, FORTRAN, etc. Further muddying the waters were development environments that, early on, were more advanced for other languages. (Hamilton recalls dealing with an Army organization seeking an Ada waiver to use Lisp simply because they wanted to use Symbolics Lisp machines for development.) Ada waiver requests were typically handled at very senior levels in the services, creating some unintended consequences. A legitimate way to obtain a waiver was to demonstrate that Ada usage was not cost effective. By focusing on upfront costs rather than downstream savings, this was often easy to do.

AdaCore (http://www.adacore.com) revolutionized the cost of Ada compilers in the '90s as their GNAT Ada compiler matured. GNAT (Gnu NYU Ada Translator) is still freely available for download.

DoD policies preferring commercial off-the-shelf (COTS) systems and components essentially eliminated the rationale for a DoD-procured compiler. In 1998, the Software Engineering Institute published a monograph on DoD COTS policies. [4]

There were certainly examples of failing DoD information systems that appeared to have successful and cheap commercial alternatives. Applying COTS to weapons systems always seemed absurd, since you cannot simply go to Wal-Mart and buy a guided missile. But 20 years later, we see networked information systems carrying more and more sensitive information, and the reality is that few commercial software products — then or now — have military-appropriate security.

One unique aspect of the Ada effort was compiler validation. DoD usage required a validated compiler, so there was little market for non-validated compilers. There were many calls for subsets and supersets, but compiler validation ensured that Ada code was always very portable and that compiling for another target architecture was generally not a problem as long as you were using validated compilers on both systems. This portability had profound implications for technical interoperability, but was generally ignored after the end of the mandate.

Computer security concerns were already surfacing in 1990s, but one thing the AJPO did not consider was the problem of rigged compilers — that is, compilers that surreptitiously create back doors in any code they generate, such as Ken Thompson demonstrated in 1984. This attack is described on stack exchange as follows:

"Re-write compiler code to contain two flaws:

"–When compiling its own binary, the compiler must compile these flaws.

"–When compiling some other preselected code (login function), it must compile some arbitrary back door.

"Thus, the compiler works normally – when it compiles a login script or similar, it can create a security backdoor, and when it compiles newer versions of itself in the future, it retains the previous flaws – and the flaws will only exist in the compiler binary so are extremely difficult to detect." [5]

Had Ada compiler validation continued, ensuring compilers did not have back doors would have been something else to consider. For more information on the DoD Ada Compiler Validation Procedures, see the 1997 ACVP posted on the Ada Information Clearinghouse. [6]

As noted in "Why Programming Languages Matter," entire classes of security vulnerabilities are eliminated when code is compiled with a validated Ada compiler. Buffer overflows, for example, are impossible in Ada. One general officer at the time remarked that this did not matter since "good programmers write good code and bad programmers write bad code." Regardless, 20 years later, the problems with unbounded buffers are well known, but buffer overflows are still at the top of most computer security vulnerability lists.

Dr. John W. McCormack's analysis of a 1997 Communications of the ACM article entitled "My Hairiest Bug War Stories" points out that of the 17 software bugs enumerated, an Ada compiler would have detected 15. [7] The software engineering literature is full of papers that suggest ways to manage security flaws that simply do not exist in Ada.

Much has been written about the technical merits of Ada. But it is important to remember why the Ada Mandate came about. "Why Programming Languages Matter" stated that Ada had achieved critical mass in DoD with an approximately 33.5 percent share of DoD weapons systems and an approximately 22 percent share of DoD automated information systems. [1] The percentage of DoD software that is still in Ada is unknown but likely in decline, particularly in information systems.

A survey of programming languages in current use is beyond the scope of this retrospective paper. The Tiobe index (http://www.tiobe.com/tiobe-index/) is another measurement of programming language use not confined to just DoD systems. The index shows Ada usage declining, currently ranking thirtieth with a usage rate of 0.655 percent. It is important to recognize that the Tiobe index measures much more than just DoD usage, but the trend seems clear. A rolling five-year history of the Tiobe index is shown in Figure 1. [8]

Ada is still here almost 20 years after the DoD ended support for the Ada Programming Language. The August 2006 issue of "Crosstalk" was entirely devoted to Ada2005. [9] Ada 2012 is an International Organization for Standardization and an International Electrotechnical Commission Standard (ISO/IEC 8652:2012). The Ada 2012 ISO/IEC standard was approved on Feb. 1, 2016. Reports of Ada's demise would seem to be premature.

The latest Ada Language Reference Manual is available for download from many sources, including http://www.ada-auth.org/standards/12rm/RM-Final.pdf. The Association for Computing Machinery has a special interest group dedicated to the Ada language (SIGAda, http://www.sigada.org/index.html). SIGAda through ACM publishes Ada Letters and conducts an annual conference entitled "High Integrity Language Technology" (HILT, http://www.sigada.org/conf/hilt2016/). Ada still commands greater interest internationally than domestically. Ada-Europe is one particularly active Ada group (http://www.ada-europe.org).

The SIGAda focus seems to be the current direction of Ada usage — employment in high integrity applications. It is hard to imagine a DoD weapons system that does not require high integrity software, but it is unlikely that the DoD will mandate a programming language anytime soon. In addition to AdaCore, commercial Ada compilers are available from several companies, including: DDC-I, Green Hill Software, Irvine Computer, Corp., OC Systems, Atego, RR Software, and PTC. While some companies are only offering legacy support, several companies are offering current compilers targeting to high-integrity applications. Dr. Martin Carlisle and the Department of Computer Science at the United States Air Force Academy developed A# as a port of Ada to Microsoft.NET (http://asharp.martincarlisle.com). [9]

TIOBE Index Very Long Term History
Programming Languages Ranked by Usage

Programming Language	2016	2011	2006	2001	1996	1991	1986
Java	1	1	1	3	17	-	-
C	2	2	2	1	1	1	1
C++	3	3	3	2	2	2	5
C#	4	5	6	11	-	-	-
Python	5	6	7	25	23	-	-
PHP	6	4	4	8	-	-	-
JavaScript	7	9	8	7	21	-	-
Visual Basic .NET	8	29	-	-	-	-	-
Perl	9	8	5	4	3	-	-
Ruby	10	10	21	32	-	-	-
Ada	27	16	16	17	7	4	2
Lisp	28	12	12	14	6	7	3
Pascal	62	13	17	15	4	3	7

Software engineering has changed a lot in the past 20 years. Where previously there were many calls, especially from government, for software reuse, now reusable components are commonplace. This reuse resulted in different problems, like determining who originally wrote which component. The DoD still has problems fielding secure, software-intensive systems, and a new programming language mandate is unlikely to resolve those challenges.

So looking into 2017, do programming languages matter? We believe the answer to this question is "yes," programming languages do still matter. While there are common and widespread security issues with current languages, each language finds a niche where it performs better than other languages for a specific application. Different projects have different requirements, and performance is almost always an issue with real-time system development. An engineering design team must consider both the speed of a system and its robustness. This is a classic trade-off in the world of programming languages that is not likely to be overcome anytime soon.

We live in a world that has multiple programming languages currently being used. Some languages are more conducive to portability, like Java, which might explain why it is the most popular in the current landscape. For rapid prototyping and a wealth of existing and easily integrated libraries, Python is a good choice. For compiled languages, C gives a strong middle ground where you have object-oriented programming, with enough control at the lower levels to get the behavior you want from the system without having to manually configure all aspects of the code. In many cases, such as the work discussed in [10], efforts are being made to create processes for quickly and efficiently increasing the reliability and robustness of software developed in particular languages. Post-development checking and enhancement is a practice often seen when developers try to minimize the shortcomings of using a particular language.

The referenced article focuses on creating more scenarios where open-source software can be used to complement an existing body of work. For languages that are not particularly portable or that do not have a great selection of existing open-source libraries, a focus on enhancing what open-source content is available can give developers more freedom to choose a language for the benefits that it has, while still attempting to minimize its shortcomings. The ability of an engineering design team to select a language that best fits their project while utilizing techniques for mitigating its shortcomings is an indication that even in the current software landscape, programming languages still matter.

Conclusion

If you consider the 1975 formation of the DoD High Order Language Working Group to be the beginning of the DoD's Ada effort, then the effort spanned 22 years. [11] By the program's end in 1997, Ada 95 had been fielded and the number of programming languages in use was estimated to be less than 50. It is not clear if anyone is counting anymore. During a seven-year project (2003–2010), the first author conducted software vulnerability analysis for the Missile Defense Agency. We encountered software written in mainstream, supported languages: Ada, C, C++, C#, FORTRAN, and Java.

The DoD Ada effort, Ada 95 in particular, certainly solved a lot of technical interoperability problems between programs adhering to the Ada Mandate. Unfortunately, no programming language could solve the proprietary and acquisition challenges that bedevil interoperability in addition to very technical challenges.

The era of building software-intensive systems with proprietary operating systems and propriety programming languages ended many years ago. The DoD Ada effort helped to end it, but even without it, other trends would have ended the practice eventually.

If the interoperability and, particularly, the cybersecurity challenges of the 21st century had been foreseen in the mid-'90s, perhaps DoD policymakers would have looked at Ada in a different light. Ada changed the conversation about defense software engineering and promoted correctness, reliability, security, interoperability and architecture, among other contributions. The DoD investment in Ada advanced compiler technology and programming language design. In retrospect, it is hard to dispute that DoD made a sound investment.

REFERENCES

1. Computer Science and Telecommunications Board, National Research Council. (1997.) "Ada and Beyond, Software Policies for the Department of Defense." National Academy Press, Washington, D.C. Available online at http://www.nap.edu/download/5463#.
2. Hamilton, J.A., Jr. (December 1997.) "Why Programming Languages Matter." Crosstalk: The Journal of Defense Software Engineering. Vol. 10, No. 12, p 4–6. Available online at http://www.drew-hamilton.com/pub/Why_Programming_Languages_Matter.pdf
3. Ada Information Clearinghouse, "The Congressional Ada Mandate." Downloaded from http://archive.adaic.com/pol-hist/policy/mandate.txt Accessed Sept. 7, 2016.
4. Oberndorf, P. & Carney, D. (September 1998.) "A Summary of DoD COTS-Related Polices." SEI Monographs on the Use of Commercial Software in Government Systems. Available online at http://www.sei.cmu.edu/library/assets/dodcotspolicies.pdf
5. "Ken Thompson Hack," Downloaded from http://programmers.stackexchange.com/questions/184874/is-ken-thompsons-compiler-hack-still-a-threat Accessed Sept. 6, 2016.
6. (18 Nov. 1997.) "Ada Compiler Validation Procedures." Ada Joint Program Office, Version 5.0. Downloaded from the Ada Information Clearinghouse, http://archive.adaic.com/compilers/val-proc/1222val.html.
7. McCormack, J. W. (29 March 1997.) "Ada Kills Hairy Bugs." Ada Home, The Home of the Brave Ada Programmers. Downloaded from http://www.adahome.com/articles/1997-05/am_bugs.html Accessed Sept. 6 2016.
8. Tiobe Index, "Very Long Term History." Downloaded from http://www.tiobe.com/tiobe-index/ Accessed Sept. 6 2016.
9. Crosstalk: The Journal of Defense Software Engineering. Vol. 19, No. 8. Available online at http://static1.1.sqspcdn.com/static/f/702523/9277154/1288925958213/200608-0-Issue.pdf?token=55AQAO6AKygfHRhdhMKYpAU8HI8%3D
10. Pape, P. A., & Hamilton, J. A., Jr. (Jan./Feb. 2016.) "Better Reliability Verification in Open-Source Software Using Efficient Test Cases." Crosstalk: The Journal of Defense Software Engineering. Vol. 29, No. 1. Available online at http://static1.1.sqspcdn.com/static/f/702523/26767145/1451886697337/201601-0-Issue.pdf?token=B1bglcNUe0f0f1G4RtR87ltierl%3D
11. Whitaker, W. A. (1996.) "Ada–the Project: the DoD High Order Language Working Group." History of programming languages–II. (Bergin, T.J., Jr. and Gibson, R.G., Jr. ed.) ACM, New York. p. 172–232.