SEPR and Programming Language Selection

Richard Riehle
Naval Postgraduate School

When former Assistant Secretary of Defense for Command, Control, Communications, and Intelligence Emmett Paige issued a memo in 1996 abrogating the Department of Defense's single-language policy, he included a clause designating programming language selection as a part of the Software Engineering Process Review (SEPR). The intent of his memo and its realization have not yet converged. Many readers of the memo mistakenly assumed his intent was a license to abandon Ada rather than advice to determine language selection as part of a rational evaluation step in the SEPR. This article addresses that confusion. Many of the references to Paige in this article originate in private conversations and correspondence between Paige and the author.

Former Assistant Secretary of Defense for Command, Control, Communications, and Intelligence Emmett Paige issued a memo in 1996 abrogating the Department of Defense's (DoD) single-language policy. His memo included a clause designating programming language selection as a part of the Software Engineering Process Review (SEPR). Many who read the memo mistakenly assumed his intent was a license to abandon Ada rather than advice to determine language selection as part of a rational evaluation step.

As a consequence of misinterpreting Paige's memo, the DoD is retrogressing toward the situation prior to 1983; a period sometimes described in *Tower-of-Babel* terms. Projects are adopting a language *du jour* policy that is destined to restore the havoc experienced prior to the single-language policy. For example, one DoD software group has replaced all of its Ada code with Perl. While Perl is a perfectly good scripting language, the decision to use it instead of Ada for this application represents a substantial failure of management to understand its long-range responsibilities. Program managers are once again required to cope with a multiple-language policy rather than a single-language policy.

When the Ada mandate was relaxed, some in the software community asked, "If the DoD cannot manage a single-language policy, how can it be expected to manage a multiple-language policy?" Although it was not explicitly stated in his memo, Paige did not want a return to the days of more than 400 DoD programming languages. He included the SEPR clause intending to provide guidance for this new multi-language policy.

Most DoD program managers are unprepared to make decisions regarding language choice. They are at the mercy of each contractor. In the future, we will see DoD software written in Java, C++, C#, Ada, Eiffel, Ruby, Perl, Python, Smalltalk, Fortran, COBOL, Euclid, Lisp, Prolog, Haskell, or even some proprietary languages invented by contractors.

Compilers for these languages, with the exception of Java and Ada, implement dialects that fail to correspond to a published standard. Gone are the days when a program manager would be able to insist on a validated compiler for the chosen programming language. No one even pretends that C++ compilers can be validated. In my conversations with program managers, I find that many have simply abandoned the language decision to the contractor. While this is expedient in the short term, it is likely to create major problems in the future. The Tower of Babel is slowly being rebuilt.

> *"Program managers are once again required to cope with a multiple-language policy rather than a single-language policy."*

More Mature Selection

At first, many in the software community were surprised that someone with Paige's appreciation of Ada would issue a memo abrogating its mandate. In meetings with Ada advocates subsequent to the SEPR memorandum, Paige emphasized several key points. He noted that, with more than 50 million lines of Ada code deployed in operational weapons systems, Ada had proven it could do the job it was intended to do. He also noted that, instead of unifying the software community within the DoD, Ada had become a rallying point for bickering among contractors and military officials. Its image was being tarnished through flurries of e-mail and other correspondence, often by some of the very people who were its advocates. Paige came to believe that Ada was good enough to stand on its own against alternative choices.

His decision coincided with the advent of the Ada 95 standard. In 1995, Ada changed from a military standard (MIL-STD 1815A) to an ISO/ANSI standard (ISO8652-1995), which made Ada 95 a powerful language for real-time, embedded object-oriented systems. Those who discovered that fact are enjoying the benefits of Ada while those who have chosen error-prone languages such as C++ have a long struggle ahead of them. The DoD will not be the beneficiary of that struggle.

Paige envisioned the SEPR as an essential part of any DoD software engineering effort. We know that successful software engineering projects start at a high level of abstraction. Before adopting tools, languages, and methods, we need to answer a question at the highest level of abstraction: "What problem are we trying to solve?"

As a project moves forward, even under agile processes, there is a succession of reviews to decide on methods, tools, and languages. We use the plural of language because more and more contemporary projects are implemented in multi-language environments. We would not choose C++ where HTML is appropriate nor would we choose Ada where we could more effectively use MATLAB. Java would be inappropriate for high reliability mathematical applications but might be perfect for displaying some of the results of those computations. XML is of growing importance, and XML works well with both Java and Ada.

In other words, we select the appropriate language for the problem to be solved. This is analogous to selecting the right tool for the job at hand. A pipe wrench does a different job than a box

wrench. When there is a chance we might break off the head of a bolt with a long-handled wrench, we use a torque wrench. Of course, we must know something about torque wrenches before we use them.

Paige's SEPR memo suggests that programming language selection should be a carefully considered process with the decision made on the basis of criteria derived from the project requirements. Although it was noted earlier that Paige was responding to a controversy, it was that controversy that led to his realization that the DoD needed a more mature approach to programming language selection. This, in turn, led him to the decision to include programming language choice in the SEPR.

One important benefit of abrogating the mandate has been the democratization of Ada. When the mandate was in place, Ada compiler publishers had the DoD as a captive customer. They could charge whatever they wanted for their technology. Prior to the Ada 95 standard, Ada compilers and tools were priced so commercial software developers could not afford them. When that captive DoD audience diminished, many Ada compiler publishers vanished or merged. In the absence of the mandate, compilers and tools, downloadable by anyone with access to the Internet, are now free.

Programmers worldwide are now experimenting with Ada. More non-DoD developers are quietly using it. Contemporary Ada has been adopted by the United States' friends and enemies. For example, Iranian and Chinese military software engineers are now using Ada. It is not as popular as C++ or Java, but it is equal to those languages in every way and better in some respects. At this stage, the cost of Ada technology should not be any greater than for C++ technology. Ada compilers are no more difficult to create than C++ compilers. They can be hosted on any computer in existence or being planned.

Programming languages are designed according to different goals. The SEPR must consider its language choice with an understanding of those goals, along with other factors. Those other factors include mission requirements such as targeted platform, expected level of dependability, maintenance ease, compiler availability, development tools, and environments as well as others. The factors should be determined by defining the criteria appropriate to the software product requirements.

Too often, language is chosen by the programmers, the contractor, or through some *ad hoc* decision-making process that has little to do with the underlying requirements. It is not unusual to see programmers make the language decision in pursuit of career goals. Do not let the programmers decide what language will be used for a sensitive DoD project. Long after the original programmers are gone, the software will require continued maintenance. Language choice must be a management decision based on what is best for the long-term health of the final software product.

It is rarely difficult for programmers to learn the language needed for the project. Ada, well taught, is as easy to learn as any other contemporary language. Our tools, including our programming languages, must help us meet the mission requirements with minimal error and maximum reliability.

"Too often, language is chosen by the programmers, the contractor, or through some ad hoc decision-making process that has little to do with the underlying requirements."

Reliability is an essential criteria for DoD weapons systems. A pilot attempting a carrier landing will be greatly annoyed if greeted by a heads-up display that announces, "Sorry. System error occurred. Please reboot." That is the kind of thing that can happen if we make the wrong choices.

So how should the programming languages be selected during the SEPR? I believe the answer lies in ideas: context and criteria. Can criteria be defined in the context of the future product? Lloyd Mosemann, senior vice president of Corporate Development for Science Applications International Corporation and former deputy assistant secretary of the Air Force for Communications, Computers, and Logistics, often cites predictability as an essential characteristic of DoD software. Predictability is an essential property of any engineering effort. Software engineering is not any different.

Can DoD program managers give guidance and direction about programming language choice? Should they simply provide context and criteria and let the software contractor choose the programming language? I believe the program manager, representing the DoD, should have a role in the programming language decision. The sad fact is that I see many contractors making the programming language choice on the basis of convenience, résumé building, and other factors that have little to do with product quality. Some consultants make recommendations based on poorly chosen criteria. Worst of all, language choices are made on the basis of what is currently popular rather than on what is best for a particular project.

Language Options

Because so many languages are available, this article focuses on object-oriented programming (OOP) languages. There are many OOP language choices available, including Smalltalk, C++, C#, Java, Ada, Eiffel, Modula-3, and Object COBOL. This list could be longer, especially if it included some of the excellent new languages, such as Ruby, or discussed the value of functional languages such as Objective CAML and Haskell.

Each language has benefits in given application domains. Some are better for one domain than another. Advocates will make the case that a favorite from the list is great for every kind of application, but that claim must be supported by the criteria declared for the targeted domain.

The language decision must recognize the difference between two issues: *expressibility* and *expressiveness*. Nearly every programming idea can be expressed in your favorite language. *Expressiveness* is about how well a language maps its solution space to the problem space. The ability to express an idea is called *expressibility*. The ease of expressing that idea is called expressiveness. For example, we might be able to compute Bessel functions in Lisp, but Fortran better expresses mathematical functions than Lisp. Lisp is more expressive of ideas related to artificial intelligence.

A DoD contractor for whom I once worked was assigned to create materials management software for a specialized Navy environment. At that time, the contractor was a Fortran programming shop. Some members of our team, comfortable with Fortran, insisted we could do the entire project in Fortran. From the perspective of *expressibility*, they were right. Others on the team made the case for COBOL. The COBOL advocates correct-

ly pointed out that COBOL was designed to express exactly this kind of application. The Fortran advocates correctly pointed out that they could do anything in Fortran that the other group could do in COBOL. The COBOL advocates won the day. The deciding factor was one of expressiveness over *expressibility*. Although Fortran could express the required programming solutions, management decided that COBOL was more expressive of those solutions.

Expressiveness is one of the earliest issues to consider before choosing other criteria. However, one needs to exercise some care about this. Some special purpose languages are expressive for specific applications, but targeted so narrowly they fail to meet other requirements. Also, many expressive languages are proprietary, useful only on one operating system, or poorly supported. For example, Visual BASIC is popular for programming on Microsoft platforms but is nonportable when considering other environments.

The program manager and the contractor together formulate the relevant criteria. Are the applications computational/numerical? Is nonportable software OK? Must we be able to deploy on multiple operating systems? Will this application require a lot of string manipulation? Is there an embedded real-time requirement? Must we interface with other languages? Is the application short-lived or long-lived? What is the cost of a software failure? Do we expect a lot of enhancements during the life cycle of this product? Is this a graphics product? What are the human-machine interface requirements? What are the software architecture considerations? Can we get efficient code from the available compilers? Is there a technology transition cost? The list of possible questions continues.

As you develop criteria, try to include a numerical weight and rating. This is a good place to use a spreadsheet listing languages and criteria with weights for the various criteria (see Table 1). Have more than one person assigning the scores on separate versions of the spreadsheet. You will be surprised by the disparity of viewpoints. Gather those involved in the scoring process and encourage a discussion that excavates biases, predilections, and perversions that may have influenced each person's scoring. The final score on the spreadsheet should be one of your indicators, but not the only indicator.

Language Criteria

Consider Table 1 in which sample criteria are weighted in favor of a safety-critical weapon systems. The entries will vary according to each kind of system. To avoid introducing too much bias into the chart, and to acknowledge that projects will evaluate criteria differently, I have masked the names of the languages. Your evaluation would insert the languages of interest in the spreadsheet.

		Language					
Criteria	**Weight**	**A**	**B**	**C**	**D**	**E**	**F**
Object-Oriented Programming (OOP)	3	3	3	4	5	4	5
Built-in Concurrency	5	5	0	3	2	0	0
Safety-Critical Features	5	5	3	3	4	3	1
Ease of Learning	3	3	2	4	4	4	5
Java Virtual Machine	2	3	0	5	2	0	0
Portability	4	4	3	4	4	3	4
Relevant Component Libraries	4	4	4	4	4	4	2
Open Source Compilers	2	4	4	2	2	2	5
Pre-, Post-, and Invariant Assertions	3	3	1	4	5	2	2
Type Safety	4	5	3	4	4	4	2
Development Tools	3	3	4	5	5	3	2
Language Maturity	4	4	4	3	4	1	1
Market Penetration	3	2	5	5	1	0	1
Microsoft Windows	2	4	5	1	4	3	3
Linux	4	4	3	1	4	1	5
Other Unix	4	4	3	1	4	1	5
Generic Templates	4	5	5	0	4	0	0
ISO/ANSI Standard Compliance	5	5	4	0	0	0	0
Interoperability	4	5	4	3	3	2	1
Raw Totals:		75	60	56	65	37	44
Weighted Totals:		279	214	192	230	128	146

Table 1: *Language Criteria Spreadsheet*

Let me emphasize that Table 1 will look different after you have combined the scores from more than one person. Where the example shows a score for OOP of five for Language D and four for Language E, someone else might score these differently. Also, someone might take issue with a score of one for Language E on Microsoft Windows. Even if no one argues about zero for built-in concurrency, they might argue that external (operating system-based) concurrency is a close enough equivalent.

Some Language Choices

Some of the OOP language choices were named earlier. The following sections contain a few pros and cons of some languages. I have chosen to mention Smalltalk, C++, Ada, Java, Eiffel, and Object COBOL. If there were enough space, I would have included some other favorites such as Modula-3, Haskell, and Ruby. Also, logic languages such as Prolog often have an important place in DoD applications. While this is all a matter of opinion, it is opinion derived from experience as well as study of the given examples.

Smalltalk

This is still the gold standard for OOP. Whenever someone writes about OOP, they compare their favorite language to Smalltalk. It is fun to program in Smalltalk. Although it has fallen out of popularity during the past several years, it comes with a powerful development environment, a large selection of libraries, and a small but powerful collection of features for building interactive applications. It is not a type-based language and does less compile-time checking than other languages. It is excellent for applications where dynamic binding is beneficial. It is not appropriate for safety-critical or embedded weapon systems applications. Smalltalk is portable enough for most situations. I personally like Smalltalk, but must realistically acknowledge its limitations.

C++

This language gained a large following during the 1990s. It is losing some ground to Java. C++ has both severe critics and committed advocates. It is a general purpose OOP language that became an ISO standard in the late 1990s. Few compilers support the full ISO standard so develop-

ers often design around a subset of the C++ language to achieve portability. The language has a type system built on predefined primitive types and designer-defined classes. A well designed class is supposed to behave like a predefined type.

The C++ type system has weaknesses. Among these are the notion of structural equivalence, the potential for unruly pointers, and the excessive reliance on predefined types as primitives. Typecasting in C++ is fraught with potential dangers. C++ has borrowed a number of features from Ada, including genericity, exception handling, and dynamic memory allocators.

It is a satisfactory language for noncritical software such as windowing applications and graphics. It is a poor choice for any kind of safety-critical application. It is probably a terrible choice for weapon systems, radar, space applications, or flight-control software. Validation suites for C++ do exist. Unlike Ada, C++ is held to a lower standard and no program manager or contractor ever suggests requiring a validated compiler.

Ada

The ISO 1995 Ada standard is a great improvement over the original 1983 standard. The primary goal of Ada is to maximize the amount of error detection a compiler can perform as early in the development process as possible. This has led to the strongest type-safety model of any contemporary language supplemented with a set of rules for scope and visibility that prevents name collisions, structural equivalence problems, dangling pointers, pointers to nonexistent data, along with many other problems. Ada supports several levels of object technology, including inheritance, polymorphism, dynamic binding, and genericity. However, OOP is optional and can be avoided when inappropriate for a particular application. Ada supports built-in concurrency and embedded real-time systems.

It is still the best choice for DoD software such as safety-critical, real-time weapon systems and flight-control software. For interoperability, Ada 95 is probably the most hospitable language of all. It directly interfaces with several legacy languages as well as with C++ and XML. Ada compilers used for DoD software are held to a higher standard than any other language. That is, the compilers for Ada are always required to pass the validation suite before they can be used in DoD applications. Safety-critical applications continue to be best served by Ada.

Java

Not since PL/I has a language been introduced into the marketplace with so much hyperbole. Java is a language that promises to become better with time. At the elementary level, it is comparatively easy to learn. It is, in many respects, safer than C++. Indirection in Java does not use computational pointers, thereby reducing some of the risks encountered with C++. It also has automatic garbage collection, which some see as a blessing and others as a curse. Java consists of three basic parts: the language, the libraries, and the Java Virtual Machine (JVM). The most important contribution of Java is not the language. The language actually contributes very little new and actually represents a step backward in some respects. Rather, the important thing about Java is the libraries and the JVM. In particular, the JVM permits a high level of portability for compiled applets.

Java includes a capability for concurrency but falls short of the concurrency model of Ada. Java provides none of the deterministic behavior expected for a hard, real-time software environment. Like Smalltalk, Java is fun for programming, largely because it appeals to the instant gratification that entices so many of us. The language includes a disclaimer that it is not intended for safety-critical applications. Do not use it for weapon systems or applications where high reliability is required. Java is not yet a standard. It seems to be an evolving product that will probably stabilize in the next couple of years.

"With the abrogation of the DoD's single language policy, we need to take care not to fall into the trap of avoiding Ada or becoming too attached to it."

Eiffel

This is a relative newcomer, but older than Java. It is everything one could get from Java, except the bytecode. Grady Booch, chief scientist at Rational Software Corporation, in an off-the-cuff remark at a software conference said of Eiffel, "Eiffel is what C++ could have been if C++ had not been dependent on C."

The language permits a stronger typing model than Smalltalk but still emphasizes a pure OOP approach. For applications programming it is a better choice than C++ because it permits a more natural form of expressiveness than C++. When considering type safety, Eiffel is probably more reliable than C++. Eiffel includes a powerful development environment along with a full library of generic reusable components. Eiffel does not enjoy the large audience of users it deserves. Most Eiffel compilers are C Path, meaning they generate intermediate C code. Eiffel also has a built-in capability for programming with assertions. Assertions are pre-, post-, and invariant conditions that can be applied at many levels within an Eiffel module. The designer of the Eiffel language, Dr. Bertrand Meyer, calls the use of this feature *design-by-contract*. None of the other languages mentioned so far are as robust in this regard. Even Ada, which supports a kind of range constraint assertion, does not yet support assertions such as those found in Eiffel.

Eiffel is still not my first choice for safety-critical weapons systems, but it is probably a better choice than either C++ or Java. There is no ISO standard for Eiffel, but there is an international body called Network Information and Control Exchange that oversees its progress. A program manager once told me, "The only two languages I would consider are Ada and Eiffel." If you have not yet looked at Eiffel, you might consider doing so.

Object COBOL

If you are currently programming in COBOL, Object COBOL makes sense. Many COBOL shops are making the mistake of converting to Java, or worse, C++. The syntax of Object COBOL will be familiar to your programmers. They can learn OOP on the job using this familiar syntax. Everything you liked about COBOL is still there, but you can enjoy inheritance, information hiding, encapsulation, polymorphism, dynamic binding, and everything else expected from an OOP language. Contemporary COBOL is a language with expressive power required for business and business-like applications. It includes features that will make your current COBOL programmers and systems analysts more productive than they would be after retraining in some other language. Object COBOL can

improve communications between clients and systems analysts as well as between systems analysts and programmers. Object COBOL is not appropriate for safety-critical weapons systems, but it is an excellent step into the future if you are already in a COBOL programming environment.

Summary

The programming language selection process for DoD software systems is too often made on the basis of inadequate criteria. With the abrogation of the DoD's single language policy, take care not to fall into the trap of avoiding Ada or becoming too attached to it. It is important to recognize the strengths and weaknesses of more popular languages such as C++ and Java, and understand when to choose them and when to reject them.

Paige's SEPR memorandum suggests a direction without specifying too many details. As we implement his suggestions, we must do so on the basis of carefully defined criteria and with sufficient knowledge to understand the contribution of each of the alternatives in terms of those criteria. Also, selecting the language is not sufficient. We must insist that the compiler for the language we choose conforms to the highest possible set of standards available. Unlike commercial software, the safety of our uniformed personnel and the success of our military missions depend of the reliability of the choices we make.◆

About the Author

Richard Riehle is a visiting professor at the Naval Postgraduate School. He also owns AdaWorks, a small company dedicated to Ada consulting and training. His book "Ada Distilled" may be downloaded free from <www.adaic.org>.

Naval Postgraduate School
Computer Science Department
Spanagel Hall
Monterey, CA 93943
E-mail: richard@adaworks.com or rdriehle@nps.navy.mil.

WEB SITES

Data and Analysis Center for Software

http://dacs.dtic.mil

The Data and Analysis Center for Software (DACS), a Department of Defense (DoD) Information Analysis Center, is designated as the DoD software information clearinghouse, serving as an authoritative source for state-of-the-art software information and providing technical support to the software community. From its home page, the user can access more than 30 specific technical topic areas related to software engineering and software technology, including programming languages. The DACS offers a wide-variety of technical services designed to support the development, testing, validation, and transitioning of software engineering technology.

JOVIAL Lives

www.jovial.hill.af.mil

JOVIAL Lives is the official home page of the U.S. Air Force JOVIAL program office, whose mission is to provide current and future customers with superior service, support, and distribution of the best JOVIAL compilers available and JOVIAL/MIL-STD-1750A Integrated Tool Sets (ITS), which are software support tools used for development and maintenance of MIL-STD-1750A target applications. In compliance with the Air Force's policy of software reuse, this office provides a cost-effective way to maintain and modernize existing quality software products.

Ada Power

www.adapower.com

Ada Power developer resources and tools Web site was formed to contribute back to the Ada community and to help advocate this powerful language. The site features examples of Ada source code, including illustrating various features of the language and programming techniques, various interfaces to popular operating systems, various algorithms, a collection of packages for reuse in Ada programs, inplementation articles, numerous Ada links, and more.

Computing Research Association

www.cra.org

The Computing Research Association (CRA) includes more than 200 North American academic departments of computer science, computer engineering, and related fields; laboratories and centers in industry, government, and academia engaging in basic computing research; and affiliated professional societies. CRA's mission is to strengthen research and education in the computing fields, expand opportunities for women and minorities, and improve public and policy-maker understanding of the importance of computing and computing research in our society.

Ada Information Clearinghouse

www.adaic.org

The Ada Information Clearinghouse (AdaIC) was formed to ensure continued success of Ada users and promote Ada use in the software industry. The AdaIC has served a community of software engineers, managers, and programmers for more than 15 years. The Web site provides articles on Ada applications, databases of available compilers, current job offerings, and more. The AdaIC is managed by the Ada Resource Association, a group of software tool vendors that supports the use of Ada for excellence in software engineering.

IEEE Computer Society

www.computer.org

With nearly 100,000 members, the International Electrical and Electronics Engineers Computer Society (IEEE-CS) is the world's leading organization of computer professionals. Founded in 1946, it is the largest of the IEEE's 36 societies. The IEEE-CS's vision is to be the leading provider of technical information and services to the world's computing professionals.The Society is dedicated to advancing the theory, practice, and application of computer and information processing technology.