CHAPTER 2
WHY ADOPT ADA 95?

SUMMARY OF ADOPTION ISSUES

PEOs and PMs now have the opportunity to adopt Ada 95. This chapter provides a rationale for why they may want to take advantage of Ada 95 features and benefits. As noted in the Introduction, subsequent chapters will address the issues associated with Ada 95 adoption and provide resolution strategies. *Each subsection in this chapter describes the reasons why PEOs and PMs would consider adopting Ada 95 from both the perspective of those using Ada 83 and those using other languages.* Tables 1 and 2 summarize the key management and technical benefits that Ada 95 may provide.

Table 1: Key Management Benefits in the Switch to Ada 95

Key Management Benefits	Why Adopt Ada 95?
Transition Risks Addressed	**Ada 95 Adoption Builds on Ada 83 Lessons Learned —** DoD and Industry have collaborated to minimize the costs and risks. DoD has "primed the pump" by sponsoring transition activities (e.g., references and guides, initial training resources, pilot project efforts). Industry has learned from the early Ada 83 transition efforts and is avoiding the same problems the second time around.
Lower Cost	**Increased Productivity and Reduced Development Costs —** Through enhanced support of software reuse and re-engineering
Shorter Schedule	**Increased Efficiency and Effectiveness across Software Life-Cycle —** Reduced recompilation during development, expanded use of reusable libraries, and the use of object-oriented programming
Risk Reduction	**Improved Real-Time Features —** Language features help to ensure that real-time projects meet their performance and hardware resource constraints.
Risk Reduction	**Maximum Use of Existing Ada 83 Code —** Extremely high percentage of Ada 83 code will compile and run "as-is" when transitioning to Ada 95. Extensions in Ada 95 can be gradually phased in to minimize impact on existing or planned development activities
Conformance to Policy	**Ada 95 Replaces Ada 83 —** Initially, policy will allow both; long-term policy will phase out use of Ada 83. Vendors will gradually shift their resources to support only the new tools.
Conformance to Standards	**Improved Portability and Interoperability —** Language features help support migration of legacy code, interoperability of client/server systems and integration with non-Ada code and COTS

Table 2: Key Technology Reasons to Switch to Ada 95

Key Technology Reasons	Why Adopt Ada 95?
Ada 95 = Ada 83 plus new technology	Ada 95 offers Ada 83's strengths plus new extensions to resolve limitations in areas such as real-time performance, object-oriented programming, interfaces to COTS and safety/security
Same Advantages as the Alternatives	Ada 95 offers many of the advantages of alternative languages (i.e., C++ and Smalltalk) with the added benefit of upward compatibility from Ada 83
Stronger Real-Time and Object-Oriented Support	Ada 95 provides much stronger support for both object-oriented development and real-time systems requirements
Improved Interfaces to Legacy Systems	Ada 95 provides improved, standardized ways to interface with other languages (e.g., FORTRAN, COBOL and C)

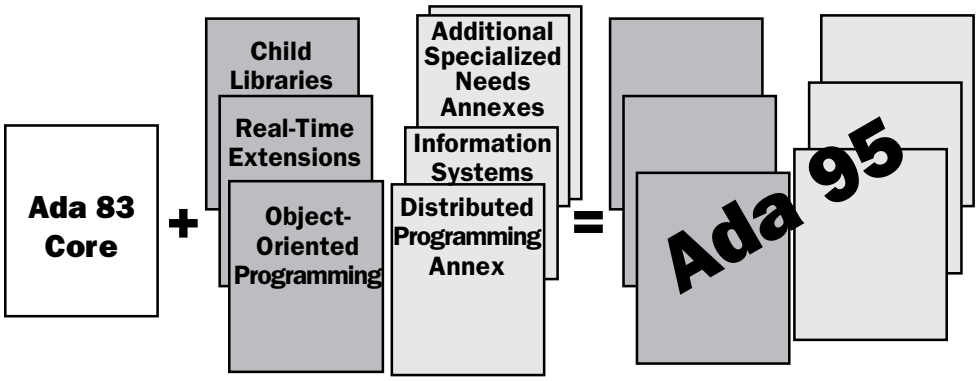
What Is Ada 95?

Ada 95 is the informal name for the current Ada standard (ISO/IEC 8652:1995). It is an *incremental improvement* of the Ada 83 (ANSI/MIL-STD-1815A) programming language. It is *neither* a new language *nor* some radical redesign. When moving to Ada 95, you keep your Ada 83 software and your investment in Ada 83 training, tools, or technology. The transition is a natural evolution. As Figure 1 shows, the new language overcomes some limitations of Ada 83 and adds new features to the existing Ada 83 language.

Ada 95 Builds Incrementally on Ada 83

Ada 83 developers can build on their existing knowledge when learning Ada 95. The new parts of the language can be learned gradually and its benefits incorporated in small steps. Immediate benefits are realized by beginning to use Ada 95's simpler features while building up to using *all* the features that the language has to offer. This enables the developers to follow the KISS principle. While an incremental approach simplifies learning and lowers costs, development teams may not realize all the benefits immediately.

Figure 1: The Incremental Nature of Ada 95 — Ada 83 Plus Enhancements



Why Was Ada 95 Developed?

So, why is there an Ada 95? One reason is that users demanded it. Programming technology advances rapidly — many technologies have matured since 1983, including object-oriented programming, client/server and distributed computing, and improved real-time scheduling algorithms — just to name a few. Like most software users, Ada's customers demanded both new features and improvements to existing ones. Feedback from the users of Ada 83 was positive; their projects were successful. But people wanted features that weren't available in Ada 83.

ANSI standards must be reviewed every five years. The Ada Board, based on its perception that there were things that should be changed and added to Ada 83, recommended that the standard be revised. When the Ada revision process started in 1988, users around the world sent in more than 750 revision re-

quests. The language revision was an open, publicly reviewed process. This consensus-based approach included involving users who reviewed and commented on the changes to the language at workshops, conferences, through the Internet and, of course, as part of the formal standards committee.

Why Use Ada 95 Over Ada 83?

Ada 95 is the next version upgrade to the language. Table 3 summarizes the management benefits of the major new technical features available in Ada 95. If your system has the properties shown in the first column of the table ("Systems That Have These Characteristics"), then you are likely to benefit from Ada 95.

Features and Benefits of Ada 95

Management reasons to switch to Ada 95 include:

- *Lower Costs through Software Reuse* — Object-oriented programming and interfacing features of Ada 95 will support large-scale reuse (i.e., use of existing class libraries, etc.).
- *More Efficient Development* — New hierarchical libraries will reduce the amount of recompilation needed to develop and maintain large software systems, thus accelerating development and making your software workforce more productive. A significant portion of development time is devoted to compilation and recompilation.
- *More Efficient Software* — Several new, more efficient concurrency features will enhance real-time performance.
- *Improved Maintainability* — Both object-oriented programming and hierarchical libraries will make software changes to meet new user requirements quicker and more efficient.
- *Eventual Loss of Ada 83 Support* — Vendors will be targeting the majority of their resources to support the new versions of Ada compilers. Although support for both Ada 83 and Ada 95 will be available for a period of time, eventually only Ada 95 will be supported.

Increased Functionality

Ada 95 makes available new language features not found in Ada 83. Table 3 summarizes their benefits. These features are grouped into four major categories:

- *Object-Oriented Programming* (OOP) — The ability to develop new software components by specifying how they are similar to and different from existing components. ***OOP is a popular approach to help make software easier to modify and less costly to build and maintain.***
- *Hierarchical Library Units*: Increased Support for Programming-in-the-Large — The ability to structure large packages into groups of small related "subsystems" that are more understandable, can compile and recompile faster, and can decrease executable code size. ***The hierarchical library units result in software that can be modified more easily, greater developer productivity, and lower cost.***
- *Real-Time Enhancements* — New real-time constructs (e.g., protected types) provide more efficient support for data-oriented concurrency. More direct support for interrupting processing to respond to mode changes is provided. These real-time features decrease the risk of using Ada over assembly languages. ***Real-time enhancements also increase software efficiency.***

Table 3: Benefits of Using the New Features of Ada 95 in Various Systems

Systems That Have These Characteristics	May Gain These Benefits	From These New Features
	Object-Oriented Programming	
• Have designs with similar objects and variations among the objects (e.g., multiple kinds of sensors or multiple kinds of soldiers)	• Faster development time and/or lower costs due to increased reuse Greater maintainability due to the extensibility of software	
• Have program logic that depends on which variant of an object is used	• Faster development time as objects are extended rather than modified	
	Hierarchical Program Units	
• Are currently making use of "subsystems" and object-oriented concepts	• Less recompilation as software is extended rather than modified	
• Have interfaces to complex common software or COTS software bindings	• More understandable sets of software • Less executable code, as only children of needed hierarchical library units are incorporated	
	Real-Time Enhancements	
• Have tasks that share data • Already use vendor-specific extensions to meet performance goals	• Faster execution time than Ada 83 • Lower risk of exceeding CPU budget in meeting performance constraints	New Real-Time Primitives
• Are currently using rate monotonic analysis and scheduling • Have changing missions and real-time requirements	• Lower risk of not meeting task scheduling requirements • Faster execution time as task scheduling delays are minimized	Expanded Task Scheduling Features
• Have strict real-time performance requirements	• Less need to go outside Ada to meet performance requirements	
• Currently use vendor-specific real-time extensions and tasking features • Need information on the performance of real-time Ada features	• Reduced risk with new standardized interface to real-time systems services • Reduced risk with increased ability to control real-time execution	Additional Real-Time Support
	Specialized Needs Annexes	
• Are information systems efforts • Are legacy information systems, or re-engineering and migration systems	• Faster development schedule by using built-in facilities for decimal arithmetic and formatted picture IO	Information Systems
• Are a single logical program run on multiple computers connected by networks (e.g., client/server systems)	• Lower costs to create and maintain distributed object-oriented programs due to integrated language support • Lower risk due to increased checking for consistency across units	Distributed Computing
• Are migration systems, re-engineering efforts, systems interfacing to legacy code, or systems integrating COTS software	• Lower costs due to ability to use COTS and legacy software written in other languages	Interfaces to Other Systems
• Need strings, numerics packages, and/or IO streams	• Faster development time and lower costs due to reuse of common objects	Additional Standard Libraries

- *Interfacing with Other Software* — Better and more intimate interoperation with COTS and GOTS software written in other languages. ***This interfacing feature increases software interoperability.***

- *Specialized Needs Annexes* — The annexes define support for many domains. Packages to help support legacy applications in information systems. An annex providing a portable model for producing distributed Ada programs supports client/server computing and distributed computing. Real-time systems are supported by packages and commands to provide more sophisticated scheduling and concurrency needs. Additionally, auxiliary libraries support areas such as formatted IO and advanced math functions. ***Specialized needs annexes allow vendors to provide greater amounts of specialized support to their customers within the standard.***

Faster Development Of Systems

New Ada 95 features will allow programmers to be significantly more productive than their Ada 83 counterparts. Ada 95 adds new capabilities and rules to Ada to *reduce* the amount of *recompilation*. A significant portion of development time is devoted to compilation and recompilation, especially during the test phase. Large Ada 83 projects will benefit from the ability of users to modify their system by extending existing software rather than by changing its interface (interface changes have caused extra recompilation time for large Ada 83 systems). Programmers primarily exploit extensibility in Ada 95 through the use of child library units (to extend existing packages with new functionality), and through inheritance (to extend types with new values and operations/functionality). Both features extend the software without recompiling the parts of the system that depend only on the parent package. The enhanced generic capability in Ada 95 allows developers to build templates and then change their implementation without impacting users of those templates.

Why Use Ada 95 Rather Than Language X?

Once an organization has decided to consider a language change, several issues must be considered. Typically, an organization will move from its existing language to another language when the benefits of the new language clearly outweigh the effort to change the status quo (e.g., training, buying new tools, etc.). The following subsections describe some of the advantages and the disadvantages Ada 95 has, compared with other programming languages. Individually, many of these features are not unique to Ada 95, but taken together, they make Ada 95 a good alternative to other languages.

Ada 95 Provides New and Unique Features

Ada 95 will keep developers competitive. Ada 95 provides the same kinds of features found in other modern languages. These include: support for object-oriented programming, strong real-time building blocks, and the ability to more directly access the hardware system. All these features supplement the features of Ada 83: packages and generics (promoting reuse and modifiability), tasking directly integrated into the language (supporting concurrency and real-time programming), exception handling (promoting safety and reliability), and a rich set of built-in structures and types (making the software reflect the real world).

Choosing Ada 95 keeps development efforts abreast of Industry. Ada 95's developers have taken lessons from the best Object-Oriented Programming Languages (OOPLs) to ensure that it will be a language for the 20th and the 21st centuries. Ada 95 doesn't just provide technical parity with other languages, in some ways it surpasses them. Ada 95 provides a rich set of OOP features and integrates them with tasking and generics. Table 4 compares features found in C++ and Smalltalk with those in Ada 95. ***Ada 95 provides***

Table 4: Comparison of the OOP Features of Ada 95 and Other Popular OOPLs

Feature	Ada 95	C++	Smalltalk
Strong Typing	✓	✓	—
Compiles Time Checking of Errors (e.g., type mismatches)	✓	✓	—
Single Inheritance	✓	✓	✓
Multiple Inheritance	✓*	✓	—
Polymorphism	✓	✓	✓
Namespace management (e.g., packages or Namespaces)	✓	✓	—
Exception Handling	✓	✓	—
Hierarchical libraries (of Namespaces)	✓	—	—
Concurrency	✓	—**	✓
Distributed Programming	✓	—**	—**
Parameterized components (e.g., generics or templates)	✓	✓	—

* The three common uses of multiple inheritance are supported in Ada 95 through a combination of existing Ada 83 and new Ada 95 facilities.

** These features may only be achieved through the addition of third-party libraries.

the power of object-oriented programming — reuse, higher productivity and lower maintenance costs.

Using a high-level language, for real-time applications, decreases technical risks, provides increased productivity, lower cost and greater system reliability. Ada 95 provides enhanced support for embedded weapons systems and other real-time projects. New real-time features include a more efficient language primitive for providing exclusive access to shared data (protected types), more flexible ways to respond to a mode change or interrupt (the new "select with abort"), and more flexible scheduling of tasks. These features enable the use of Ada 95 where assembly language or low-level languages were previously used, thus providing increased productivity and reliability while lowering cost. It also supports PEOs and PMs in the move from specialized hardware platforms (e.g., UYK-43 and UYK-44) to open systems (e.g., commercial platforms using POSIX, etc.).

Ada's unique features include* special *support for many common application domains. Ada 95 provides several "specialized needs annexes", which define additional libraries and support for particular domains such as real-time, numerical, distributed, and information systems. These capabilities will make Ada 95 the first distributed, concurrent, object-oriented programming language to become an international standard. These specialized needs annexes will increase portability for applications within specific domains by ensuring standard interfaces to needed services. However, projects will only achieve this when they use compilers that support those annexes. For maximum portability, projects should use only the features found in the core of Ada 95.

Standard interfaces to other languages are included so that Ada programs may make use of existing code written in C, COBOL, and FORTRAN. In fact, Ada 95 is intended to exist cooperatively in a multi-language environment; this makes it suitable for migrating and re-engineering legacy systems so groups can continue to use the old system as a part of the new effort. Finally, standard Ada 83 bindings exist that permit Ada to be used with SQL. Ada 95 bindings to SQL-2 are under development.

Moving from Procedural Languages: FORTRAN, C, COBOL, JOVIAL, CMS-2, etc.

Ada 95 provides all the benefits of Ada 83 — a 10-year track record of higher productivity, lower maintenance costs, and lower technical risks – because Ada 83 is the core of Ada 95.

Ada 95 is also an opportunity to adopt modern methods and processes. PEOs and PMs are migrating their organizations and re-engineering their legacy systems. These migration efforts offer an opportunity to introduce more modern software engineering technology. Migration and re-engineering efforts allow the PEO and PM an opportunity to adopt a newer programming language. Ada 95 is an important option — it brings with it the ability for an organization to improve its entire software development process. Both events can have a synergistic effect.

Adopting Ada 95 improves the probability of success because Ada 95 provides the improvements of other modern languages — with some significant differences. Once PEOs or PMs have decided to move from their existing procedural language, the choice is then, "to what new language?" If the move stems from a desire to adopt an OOPL, then Ada 95 is a good choice compared with other current alternatives: it possesses a richer set of features than Smalltalk; and it is simpler to understand, read, and maintain than C++. Its facilities for real-time support are unparalleled in any other standardized language. All these benefits are part of an ANSI and ISO standardized language. In addition, Ada represents a DOD core competency — DOD has the skills, tools, and infrastructure to support Ada. ***When PEOs and PMs consider changing from procedural programming languages, Ada 95 is the low-risk alternative.***

Choosing among Object-Oriented Languages: Ada 95, C++, Smalltalk, etc.

Ada 95 builds upon the lessons learned from other OOPLs; therefore, it improves the probability of success when adopting a new programming language. With the large numbers of OOPLs that have emerged in the past few years, choosing the right one can be difficult. In addition to evaluating the technical merits of the language (see Table 4), managers should be aware of other impacts of choosing a language such as:

- *Vendor Independence* (*through use of a standardized language*) — Ada 95 has become an international standard ahead of all other OOPLs; this ensures that all compilers, from all vendors, provide the same language facilities. Projects making use of only core Ada 95 features will find this to be true for *all* compilers. Those making use of features from the annexes will find it true for those compilers that support those annexes. ***Standardization enhances portability — there are fewer incompatibilities when moving from Vendor A's Ada to Vendor B's.***

- *Lower Development Cost and Maintenance Cost* — PEOs must consider the trade-off of development costs *now* versus maintenance costs *later*. Ada 95 provides balanced support to both a fast "time to market" (through Ada's large set of standard libraries and good reuse support), and long-lived projects (common standards to lower maintenance costs and a common language to facilitate a pool of skilled software engineers). In balancing these demands, Ada 95 is as strong as C++. It does not try to replace rapid prototyping technology, however. These reuse benefits accrue to projects that use existing components. PEOs and PMs should remember that projects that create reusable components are making an investment and must be prepared for that extra cost. ***Ada 95's features, such as high readability and hierarchical library units, supplement Ada's OOP features in supporting both quick time to market and lower maintenance costs.***

- *Ability to Trade Performance for Power/Productivity* — Ada 95 is based on the philosophy of providing developers with building blocks. This enables them to know the costs associated with any language features

they use. ***Ada 95 was designed to enable the technical staff to use the right tool for the right job.***

- *Acceptable Training Costs* — Training costs are related to the time required to learn and use the language. ***Ada 95 is more expressive (powerful) than many OOPLs and also provides this expressive power in ways that may make learning Ada 95 no more difficult than learning C++***. Although there is only limited experience through tutorials and industry classes, early Ada 95 training costs seem comparable to those of other languages.

See Table 4 for a detailed comparison of the features found in Ada 95, C++, and Smalltalk.

Using with 4GLs: dBASE, Focus, etc.

Ada 95 co-exists with and supplements 4GLs, allowing PEOs and PMs to gain the power of both technologies and lowering the risk for information systems to move to Ada 95. There are some instances when a project may want to choose a procedural 3GL, such as Ada 95, over a non-procedural 4GL, especially when the 4GL doesn't provide enough functionality to supply special, application-specific behaviors. In this case, Ada 95 may either replace the use of a 4GL or be used to implement application-specific routines that are called from the generated 4GL code (e.g., actions when a user presses a button on a graphical user interface [GUI] screen). In other applications, Ada 95 could be used jointly with 4GLs — especially since Ada 95 has added new features that allow it to be called by other programming languages as well as call on them. One common strategy will be to use the 4GL for the GUI and database access components and have Ada 95 code serve as the application specific functionality.

Trial Ada 95 Technology Available

Low-risk, low-cost options are available to introduce Ada 95 into a project. Inexpensive sources of Ada 95 technology are available, allowing the PEO or PM the opportunity to try the technology before undertaking full-scale adoption. Appendix A lists a number of ways to get information and products including: a *free* compiler for Ada 95, called GNAT, which is useful for training and pilot project efforts; free software components; and example programs, booklets, articles and tutorials. The availability of this technology lowers the cost of evaluating and adopting Ada 95, compared to Ada 83's costs. Team members will be able to find out quickly and easily about the new Ada 95 features and see how to apply them.

Development teams can assess the cost/benefit trade-off for each new feature and use only those that benefit their project. Teams will not pay a cost for Ada 95 features they don't use. Users may adopt Ada 95 incrementally, using only a few of the new features at one time. The language designers made sure that systems that don't use a new Ada 95 feature will *not* suffer any performance cost (either size or speed). This means that a new feature will affect a development effort only if it is used. For example, if a project does not make use of OOP features, then the project will not suffer any execution time penalty in speed or size.

Stability of the Ada 95 Definition

The international standardization of Ada 95 has produced careful language scrutiny and review, which ensures that the language definition is stable and mature. In addition, the presence of an international standard increases portability and supports DOD's efforts to make use of commercial standards.

Ada 95 became the first internationally standardized object-oriented programming language on February 15, 1995. It has been sanctioned as meeting FIPS standard 119-1 and is undergoing ANSI standardization. *There will be no DOD specific standard.* This supports DOD's quest to move toward greater use of commercial standards maintained by outside standards bodies. It also makes Ada 95 more attractive to commercial firms. ***Ada 95 is a stable, international standard that will provide PEOs and PMs with greater probability of success on their projects.***

DOD's Policy on Ada 95

Ada 95 is the current version of Ada. Therefore, *no special permission is needed to use Ada 95.*

In addition, Emmett Paige, Jr., the Assistant Secretary of Defense for Command, Control, Communications and Intelligence [ASD(C3I)] issued a memo, Figure 2, *encouraging the early adoption and use of Ada 9X.*

Figure 2: Early Use of Ada 9X Memorandum

March 9, 1994

ASSISTANT SECRETARY OF DEFENSE

COMMAND, CONTROL, COMMUNICATIONS AND INTELLIGENCE

MEMORANDUM FOR SECRETARIES OF THE MILITARY DEPARTMENTS, DIRECTORS OF THE DEFENSE AGENCIES

SUBJECT: Early Use of Ada 9X

Revision of ANSI/MIL-STD-1815A (Ada 83) has progressed to the point that it is nearly certain that the new version, referred to as Ada 9X, will be approved by national and international standards bodies during 1994. To facilitate transition to this new standard, use of Ada 9X prior to final approval of the standard and/or availability of validated Ada 9X compilers is encouraged for programs described below.

Unvalidated Ada 9X compilers may be used for:

- Research and development programs (6.1, 6.2, and 6.3A appropriations),
- Proof of concept prototypes, so long as any subsequent system is delivered using validated Ada 9X compilers, and
- System development programs, so long as the systems are delivered using validated Ada 9X compilers, in accordance with the validation procedures issued by the Ada Joint Program Office.

Early use of Ada 9X provides access to the language's many enhancements, including full support for object-oriented programming, enhancements for realtime programming, and interfacing to other languages. It will permit programs to take advantage of these improvements while the final steps of the standardization process proceed.

The decision by program managers to use unvalidated Ada 9X compilers incurs risk that must be managed accordingly. Where early use of Ada 9X is not pursued, Ada 83 (ANSI/MIL-STD-1815A) is required in accordance with current policy.

(Signed)
Emmett Paige, Jr.

cc: DDR&E

The DOD policy on Ada allows the use of either Ada 83 or Ada 95. After a transition period, Ada 95 usage will be required. This approach parallels the way tool vendors support new versions of their products and phase out support for the older versions over time.

Early Results of Using Ada 95

Production projects are under development using Ada 95. Lessons learned will help future projects adopting Ada 95. Early results are encouraging. In the area of real-time systems, Ada 95 concurrency features yield large efficiency gains over Ada 83 tasking. In the area of OOP, early users found that they gained the promised benefits of reuse and productivity, with suprisingly low overhead.

General information concerning representative efforts is provided in Table 5. Additional information may be obtained from Tri-Ada 95 Conference Proceedings and the people listed in the table.

Support for Migration Systems and Multi-Language Development

Ada 95 improves the probability of success to projects taht must support multiple programming languages. Both cost and schedule will benefit by the improved ability to interface with legacy software, COTS, GOTS and other "class libraries" of reusable components — those written in Ada and those written in other languages.

Many software projects of the 1990s are either migraiton systems effors (which consolidate several systems into one) or re-engineering efforts (which take legacy code and obtain a more maintainable modern system while preserving the work invested in the legacy code). Both types of projects are similar because they require that the new code smoothly integrate with the existing legacy software. Multi-language development has always been a difficult task, but Ada 95 has several new features that make this simpler than it has been before.

As already noted, Ada 95 defines standard mechanisms for interfacing with other programming languages. In addition, the standard defines additional support packages to simplify interfacing with C, FORTRAN and COBOL. These support packages provide a portable, efficient and standard interface to legacy software. Built-in support from the language lowers the risk present in the new development effort and enhances characteristics such as portability. It also allows Ada 95 programs to use proven building blocks written in other languages as part of the Ada 95 application. This "peaceful coexistence" feature allows PEOs and PMs to make use of libraries of COTS or existing code. This reduces costs and speeds development.

PEOs and PMs should be aware that while the standard allows interfacing to any language, it does not require that all languages be supported by all compilers. Projects should be sure to evaluate each compiler's support for interfacing to their particular programming languages as a part of the compiler selection process.

Additionally, Ada 95 has the ability to be called by code written in other languages. This "call-in" capability allows Ada 95 code to be used to incrementally replace functionality present in a legacy system under maintenance, with minimal risk and disruption.

Table 5: Representative Efforts

Project	Description
Joint Advanced Strike Technology (JAST)	A joint U.S. Air Force, U.S. Marine Corps, and U.S. Navy embedded weapons system to support new aircraft. Captain Jules Bartow bartowj@ntrprs.jast.mil.
Software Activity Support System (SASS)	Redesign of C++ version of SASS using Ada 95. SASS is a part of the GCCS/Joint Maritime Command Information System (JMCIS) that provides automated system integration. For information contact Doug Lange at dlange@nosc.mil.
Airfields	A Defense Information System Agency Global Command and Control System (GCCS) application characterized by a graphical user interface and relational database executing in a networked environment. For information contact Velma Blue at bluev@cc.ims.disa.mil.
Army Common Operating Environment (COE)	Common support modules for the Army Tactical Command and Control System (ATCCS) and GCCS. For information contact Stan Levine at (908) 532-2608.
Patriot Fire Control System	A re-engineering of Patriot System originally developed using JOVIAL and assembly code. For information contact Chris Garity at chris@inmet.com.
Advanced Combat Direction System (Block 1)	A software re-engineering effort to modernize software originally developed in CMS-2/ULTRA-32 and deployed on all US Aircraft Carriers. For information contact Fred Benson at fbenson@nando.net.
Ada Embedded Computer Software Support Effort (AECSS)	A U. S. Air Force effort designed to identify and develop technology to support distributed multiprocessor software. Systems software and tools have been implemented that allow an Ada application developer to create programs that can be distributed across one or more processors in one or more chassis. For information contact Patrick Rogers at progers@acm.org.
Distributed Fighter Aircraft Simulation	A U. S. Air Force-distributed aircraft simulation that executes on several microprocessors in a single VME chassis For information contact Marc Pitarys at pitarysmj@aa.wpafb.af.mil..
Ada 95 Booch Components	A re-engineering of the C++ Booch components (12,000 lines C++) using Ada 95 (10,500 lines Ada 95). For information contact Dave Weller at dweller@starbase.neosoft.com.
Generic Ada Reusable Library for Interpartition Communication (GARLIC)	A software component implementing interpartition communication for distributed programs. For information contact L. Pautet at pautet@inf.enst.fr.
Onboard Satellite Control System	A Space AB real-time embedded onboard satellite attitude control system. For information contact A. Carlsson at Saab Ericsson Space AB, S-405 15 Goteborg, Sweden, +46 31 35 43 67.
Model-Based Software Engineering Architecture	An SEI model-based architecture transition from Ada 83 to Ada 95. For information contact A. Gargaro at gargaro@sw-eng.falls-church.va.us.
Performance Monitoring System	A MITRE Corp., Ada 83 to Ada 95 transition of performance monitoring software of a demultiplexing system. For information, contact K. Warner at (202) 651-2241.

CONCLUSION

Whether moving from Ada 83 or from another language, Ada 95 can provide significant benefits to help PEOs and PMs minimize budget and schedule disruptions and other risks. Ada 95 adoption can help a software team to provide more capabilities for every dollar spent. However, to exploit these opportunities, the process of adopting Ada 95 — as with the adoption of any new technology — *must* be managed in order to minimize transition-related risks. The next chapters will help PEOs and PMs understand the adoption process, assess and manage the risks, and successfully transition to Ada 95 wherever it is appropriate and effective to do so.