USACERL TR FF-94/19
April 1994

US Army Corps
of Engineers
Construction Engineering
Research Laboratories

Ada in the PAX Environment

by
Edward J. Japel
John Murphy
Wayne Schmidt
Laura Harmet
Scott Maxwell

The Programming, Administration, and Execution (PAX) system is a mainframe-based timesharing system that provides the U.S. Army Corps of Engineers with a number of crucial budgeting, planning, and scheduling applications. Like any mainframe-based system, PAX is expensive to operate and maintain, but it may be possible to reduce costs by converting PAX applications to Ada, the computer language mandated by Congress and the Department of Defense. While offering a number of potential advantages, conversion to Ada would also pose some significant problems.

This report examines the advantages and disadvantages of converting the PAX system to Ada. Issues discussed include PAX programming and implementation practices, conversion options, availability of compatible compilers and software tools, training, and costs.

The authors recommend against converting PAX to Ada at this time due to cost and technical considerations. Recommendations also are offered for how to make the transition most effectively should policymakers decide to proceed with conversion to Ada.

USER EVALUATION OF REPORT

REFERENCE: USACERL Technical Report (TR) FF-94/19, *Ada in the PAX Environment*

Please take a few minutes to answer the questions below, tear out this sheet, and retur~ it to USACERL. As user of this report, your customer comments will provide USACERL with information essential for improving future reports.

1. Does this report satisfy a need? (Comment on purpose, related project, or other area of interest for which report will be used.)

_____

_____

_____

2. How, specifically, is the report being used? (Information source, design data or procedure, management procedure, source of ideas, etc.)

_____

_____

3. Has the information in this report led to any quantitative savings as far as manhours/contract dollars saved, operating costs avoided, efficiencies achieved, etc.? If so, please elaborate.

_____

_____

4. What is your evaluation of this report in the following areas?

   a. Presentation:_____

   b. Completeness:_____

   c. Easy to Understand:_____

   d. Easy to Implement:_____

   e. Adequate Reference Material:_____

   f. Relates to Area of Interest:_____

   g. Did the report meet your expectations?_____

   h. Does the report raise unanswered questions?_____

   i. General Comments. (Indicate what you think should be changed to make this report and future

reports of this type more responsive to your needs, more usable, improve readability, etc.)

_____

_____

_____

_____

_____

5. If you would like to be contacted by the personnel who prepared this report to raise specific questions or discuss the topic, please fill in the following information.

Name: _____

Telephone Number: _____

Organization Address: _____

_____

_____

6. Please mail the completed form to:

Department of the Army
CONSTRUCTION ENGINEERING RESEARCH LABORATORIES
ATTN: CECER-IMT
P.O. Box 9005
Champaign, IL 61826-9005

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	April 1994	Final

4. TITLE AND SUBTITLE
Ada in the PAX Environment

5. FUNDING NUMBERS
MIPR
E8792S050

6. AUTHOR(S)
Edward J. Japel, John Murphy, Wayne Schmidt, Laura Harmet, and Scott Maxwell

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
U.S. Army Construction Engineering Research Laboratories (USACERL)
P.O. Box 9005
Champaign, IL 61826-9005

8. PERFORMING ORGANIZATION REPORT NUMBER
TR FF-94/19

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Headquarters, U.S. Army Corps of Engineers (HQUSACE)
ATTN: CEMP-P
20 Massachusetts Avenue NW
Washington, DC 20314-1000

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
Copies are available from the National Technical Information Service, 5285 Port Royal Road, Springfield, VA 22161.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

The Programming, Administration, and Execution (PAX) system is a mainframe-based timesharing system that provides the U.S. Army Corps of Engineers with a number of crucial budgeting, planning, and scheduling applications. Like any mainframe-based system, PAX is expensive to operate and maintain, but it may be possible to reduce costs by converting PAX applications to Ada, the computer language mandated by Congress and the Department of Defense. While offering a number of potential advantages, conversion to Ada would also pose some significant problems.

This report examines the advantages and disadvantages of converting the PAX system to Ada. Issues discussed include PAX programming and implementation practices, conversion options, availability of compatible compilers and software tools, training, and costs.

The authors recommend against converting PAX to Ada at this time due to cost and technical considerations. Recommendations also are offered for how to make the transition most effectively should policymakers decide to proceed with conversion to Ada.

14. SUBJECT TERMS
Ada (computer program language) cost effectiveness
PAX system
software

15. NUMBER OF PAGES
26

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	SAR

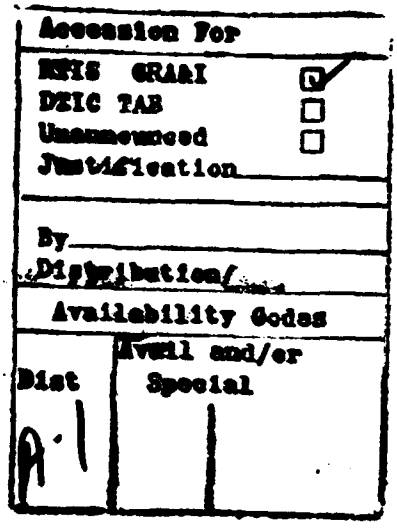
FOREWORD

This research was conducted for the Programming Branch of the Directorate of Military Programs, Headquarters, U.S. Army Corps of Engineers (HQUSACE), under military interdepartmental purchase request E8792S050, dated 25 August 1992; reimbursable Work Unit "Next Generation PAX System." The technical monitor is John Sheehey, CEMP-P.

The work was performed by the Facility Management Division (FF) of the Infrastructure Laboratory (FL), U.S. Army Construction Engineering Research Laboratories (USACERL). A portion of the work was performed on contract by John Murphy and Associates, Phoenix, AZ. Alan W. Moore is Acting Chief, CECER-FF, and Dr. Michael J. O'Connor is Chief, CECER-FL. The USACERL technical editor was Gordon L. Cohen, Information Management Office.

LTC David J. Rehbein is Commander of USACERL and Dr. L.R. Shaffer is Director.

CONTENTS



1 INTRODUCTION

Background

PAX * is a mainframe-based timesharing system developed by the U.S. Army Corps of Engineers (USACE) to support the Military Construction, Army (MCA) program. Through the PAX system menu, several important applications are available to users, including the DD Form 1391 Processor, the Automated Multi-Year Plan Applications Subsystem (MYPLAN), the Economic Analysis Package (ECONPAC), the Construction Appropriation, Programming, Control, Execution System (CAPCES), and the PC Dugout communication system.

Like typical mainframe-based systems, PAX is expensive to operate and maintain. But with the adoption of Ada as a Department of Defense (DOD) standard programming language (Public Law 101-511), it may be possible to reduce long-term operating costs by porting PAX applications to Ada. PAX applications currently exist in a conglomeration of languages and styles. Object-oriented applications written in a standardized language may be significantly less expensive to maintain and extend.

As part of a separate effort to reduce operating costs on the PAX mainframe, some PAX applications are being moved from the mainframe to MS-DOS**-compatible microcomputers (Japel et al., May 1991). Consequently, it is necessary to investigate Ada tools for microcomputers as well as the PAX mainframe environment.

Any PAX system converted to Ada would have to provide the same functionality as the current system, and should be at least equally cost-effective.

Objective

The objective of this work was to examine the advantages and disadvantages of converting PAX to Ada, and offer recommendations.

Approach

A review was conducted of the characteristics and capabilities of Ada. The relative merits of the languages used in current PAX applications were compared to Ada's strengths. The current PAX system and resources were examined to identify and examine key issues that will be involved in converting PAX to Ada. The major issues identified and discussed were:

- project management and programming philosophy
- approach to conversion
- available software tools and resources

---

*PAX: Programming, Administration, and Execution System.
**DOS: Microsoft Disk Operating System.

- compiler capabilities and costs
- programmer training availability and costs.

This report is a product of the authors' programming expertise and experience in developing PAX applications. Important information and analysis support were derived from discussions with the following consultants and vendors:

- Ada Information Clearinghouse (AdaIC)
- AETECH
- Alsys, Inc.
- Electronic Data Systems Corp.
- Information Builders, Inc.
- IBM Corp.
- R.R. Software, Inc.
- Knowledge Ware, Inc.
- McClendon Automation
- Meridian Software Systems, Inc.

Scope

Although PAX is a mainframe-based system, a separate initiative is now under way to investigate cost reductions that may result from moving some PAX applications from mainframe to desktop microcomputers. To address this potential conversion issue, this report considers the costs and capabilities of Ada tools for the microcomputer programming environment as well as for the mainframe.

The information presented in this report pertains to a very fluid software environment. Prices, product offerings, and capabilities may change rapidly.

This study evaluated Ada only as it pertains to PAX applications fielded by the Directorate of Military Programs, HQUSACE.

Mode of Technology Transfer

The findings of this study will feed into the development of Ada-based versions of current PAX applications. Users may obtain a PAX logon identification (PAXID) from Mr. Michael Rice, CEMP-P, (202) 272-0577.

2 CHARACTERISTICS OF THE ADA LANGUAGE

Congress and DOD have mandated the use of Ada for all new DOD software projects, and for existing applications in which more than 30 percent of the code is under revision.

DOD developed Ada in an effort to contain rapidly escalating software costs. Ada was developed specifically for embedded systems, such as missile guidance and aircraft control systems, where reliability and maintainability are matters of life and death. Ada still strongly reflects these origins, and does not have fully developed tools to support management information systems (MIS). Consequently, Ada has not yet been widely used for traditional MIS applications such as PAX.

Ada embodies and enforces many modern software engineering principles. DOD argues that mandating the use of Ada will achieve savings through the standardization and reuse of code. Standardization of interfaces, internal documentation, and the code itself help make programming less dependent on individual styles and talents, it is argued. Furthermore, DOD believes that standardization will turn programming from an art into a discipline. Consequently, it is believed that software production will depend less on individual genius and more on rigorous methodology. It is also believed that making code, documentation, and interfaces easy to understand will save maintenance time. Increasingly, it will be possible to build software out of previously written, thoroughly debugged modules. Such modules would inherently require less maintenance than custom-built portions of a system.

The most important features of Ada are described in the sections that follow. It will be seen that these features all encourage strong modularity in both code and data.

Software Engineering in Ada

The *software engineering* approach to system development is much more structured than merely "writing programs." Software engineering heavily emphasizes correct initial system design which—when succesfully executed—avoids many problems in constructing the software. The central goal of the Ada language is to support and enforce the use of good software engineering principles. These principles include:

- formal structured analysis of requirements
- strong data typing
- explicit interfaces between modules
- modern language constructs.

Ada's enforcement of these principles makes systems developed in Ada more reliable, and easier to develop and maintain, than systems developed in some other languages. These principles also make reusing code more practical. In fact, the Army's Reusable Ada Products for Information Systems Development (RAPID) is already online, providing software developers a library of thousands of lines of reusable Ada code.

Formal Structured Analysis of Requirements

One feature of software engineering is greater emphasis on the analysis stage—that is, deciding what tasks must be accomplished and how work will be split between modules. To use Ada properly, one must conduct a formal, detailed analysis. Until all modules and their interfaces have been defined, programs cannot be compiled and linked.

Strong Data Typing

Ada variables are strongly typed. This means that the programmer can explicitly limit variables to certain ranges of values; the compiler will ensure that the variables remain in range. For example, the programmer may define a variable to represent line numbers and specify that it may only take on integer values from 1 to 100. The executable code then "watches" the variable's value to make sure that only values within the specified range are used. However, the programmer must take most of the responsibility for using this feature. Type-checking can be defeated simply by not using the data typing function—for example, by defining a line number variable as just "any old integer" and not as a special data type. In such a case, the compiler would not flag a negative line number as an error. Using strong data typing enhances a system's reliability—if the engineer uses the capability.

Strong data typing also enables the compiler to "police" the operations allowed for a particular data type. The compiler prevents errors such as adding variables that represent pounds to variables that represent inches. Subroutine parameter types are checked, too, so the compiler can ensure that only proper parameters are used by each call.

Explicit Interfaces Between Modules

Ada modules are made up of a module header and a program body. The module header describes the module's interface to the rest of the system—what types of parameters it accepts and what kind of value it returns. The program body describes the work done by the module.

In addition to specifying each parameter's type, the module header states whether the parameter is used as an input value and whether it can be modified. Ada uses this information aggressively to ensure that parameters are used properly. Compilers for older languages such as COBOL and Fortran, by contrast, may not even check whether a subroutine has been given the correct *number* of parameters. If a COBOL subroutine requiring three parameters is called with only two, the compiler simply accepts the erroneous call. Ada will not allow a module to be built with such an error.

While type checking improves system reliability and assists in debugging, it also adds significant compilation overhead—compilation is slower and requires more memory and disk space. Furthermore, it can make system modifications more difficult because it requires modules to be compiled in a particular order.

Modern Language Constructs

Ada was designed to directly implement structured design constructs, thus making a program closely match its design specifications. For example, Ada supports common loop structures, such as "while" loops, that naturally arise in structured program design.

Another structured design concept supported by Ada is modular code. Ada readily supports breaking up code into separate blocks that are often kept in separate files. These code modules may then be reused in later software development with little or no additional effort.

One of the most important of these structured design constructs is *information hiding*, a method by which programmers can limit access to certain variables. Variables defined in a block of code are local to that block of code; they cannot be accessed or modified outside the block. This feature helps reduce the number of bugs caused by improper use of variables.

Support for Object-Oriented Techniques

The Ada standard is evolving to include new computer science principles and techniques, the most significant of which is direct support for object-oriented programming. Although the current Ada standard does not directly support object-oriented programming, the new language standard—Ada 9X, due for final release in 1993—should be implemented very rapidly by compiler vendors. The Ada 9X standard will incorporate some object-oriented ideas and constructs already included in commercial languages such as C++.

Object-oriented software analysis and design are part of a new generation of systems analysis tools. They build on existing structured techniques, providing improvements where the older tools were weak.

The traditional method of analysis has been structured analysis, in which a large system is broken down into its major components. Each component is then further broken down into smaller pieces until the pieces are simple enough to understand and implement directly. The system is decomposed from the top down, focusing on the functions the system will perform, not on the data the functions will use. While this approach has advantages, one drawback is that it is difficult to identify common operations that can be reused throughout the system. Also, because the analysis is based on functions, the design changes will probably be necessary when requirements change.

In an object-oriented approach, the system is examined to identify *objects* that are controlled by or involved by the system. In this sense of the term, an object can be almost anything: a purchase-order form, an airplane, a person, or even a computer program. The definition of an object includes not only the types of information associated with that object, but also the operations that may be applied to the object. One benefit of this approach is that objects tend to undergo fewer changes than functions, because objects are based more directly on reality. In addition, more of the code can be reused because operations on objects are defined before coding begins, so common operations can easily be factored out.

Object-oriented techniques emphasize data analysis and code reuse. They also encourage system partitioning, meaning that no part of the system depends on the details of how other parts do their jobs. Consequently, each part of the system may be changed without affecting other parts, as long as the external interface remains consistent. For these reasons, PAX could benefit greatly from the application of object-oriented techniques, whether in Ada or another language.

Comparison of Current PAX Languages With Ada

The following paragraphs compare the relative merits of the languages currently used in PAX applications to Ada.

COBOL

COBOL is an old programming language, originally released in 1960. Although it has been improved over the past three decades, its basic structure remains the same. For many years COBOL was the government standard, but it has been abandoned in favor of Ada.

COBOL is verbose and clumsy. COBOL programs tend to be larger than those written in other languages, and it cannot exploit newer programming techniques such as strong data typing and complete support for separately compiled modules.

Another drawback of COBOL is that it does not support functions–subroutines that return values. Code to calculate the square root of a number, for example, must be rewritten every time the calculation is required–even within a single program. This makes programming highly cumbersome, and often obscures *what* is being done because of *how* it must be done. COBOL has long been a standard on mainframes, but has little support available on microcomputers.

On the positive side, many programmers are familiar with COBOL, and the language does support some structured programming techniques. Also, FOCUS HLI supports the use of COBOL programs to access data.

C

C is in effect the universal programming language of the commercial world, used for everything from operating systems to communications programs to spreadsheets. Historically, it has been used mainly on microcomputers and minicomputers, but it is being used more often on mainframes. C fully supports structured programming techniques such as data partitioning, strong data typing, and separately compiled modules. However, Ada essentially forces programmers to use these techniques, while C does not. In this important sense, C offers the programmer more flexibility than Ada.

C is a highly portable language, available for nearly all computers. Programs written in C tend to be fast and small.

Commercial C libraries that perform a wealth of functions are readily available. These commercially maintained libraries reduce development time and maintenance costs. However, FOCUS HLI does not support C.

C suffers an undeserved reputation for being cryptic. It is true that poorly written C programs can be extremely difficult to understand, but a poorly written program in *any* language can be extremely difficult to understand. Nevertheless, the power of the C language can be a double-edged sword for inexperienced programmers. Ada is less terse, and has more controls to protect the novice.

C++

C++ has all the capabilities of C, supporting both the C language and C libraries, but adds several important capabilities:

- direct support for object-oriented programming
- additional checking of module interfaces
- extra support for strong data typing.

C++ is one of the newest commercial languages, and is largely responsible for the widespread interest in object-oriented programming. While Ada does not currently support object-oriented programming, the Ada standard is being revised to include some objected-oriented support. In addition, it is important to note that DOD is now considering whether it should find a place for C++.

Because C++ is new, there are relatively few people who know how to use its object-oriented features. This is not a serious drawback, however, because there is a natural migration path from C to C++: the former is a subset of the latter. Consequently, new C++ features can be incorporated into C programs as a programmer's knowledge of C++ and object-oriented techniques develops. Ada, by contrast, does not offer a gradual migration path from any other language.

C++ is not yet available for the VM-CMS mainframe environment, but a version is reportedly close to release by WATCOM.

3 KEY ISSUES IN CONVERTING PAX TO ADA

In considering the conversion of PAX to Ada, five key issues must be examined:

- project management and programming philosophy
- approach to conversion
- available software tools and resources
- compiler capabilities and costs
- programmer training availability and costs.

Project Management and Programming Philosophy

Ada not only encourages a structured engineering approach to programming—it essentially *demands* one. Ada protocols are inflexible. Its use for PAX would require a change of culture, both for project managers and programmers. At present, PAX is fairly open, supporting many approaches to software solutions. Ada stresses consistency in software construction, encouraging:

- more program documentation up front
- more complete and detailed system specifications up front
- more formal increments of software revisions (*ad hoc* fixes are discouraged).

The Ada approach offers several benefits, including:

- more reliable, modular software
- more thorough design specifications and program documentation
- easier task partitioning (especially useful for multiple programmers)
- modern language constructs that allow program design to closely match the specifications.

Because of Ada's unusually rigorous engineering approach, programmers and project managers require classroom instruction to learn how to effectively exploit the language. The authors estimate that a PAX programmer would require 3 to 6 months to become proficient in Ada.

Approach to Conversion

If PAX is to be converted into Ada, several approaches are possible. The system may be converted step by step or all at once. Also, applications may either be redesigned or translated (using code translators) directly from one language to another.

Converting PAX to Ada all at once would require many people, and the conversion would be risky because it is reasonably expected that technical problems, such as incomplete code translations or misuse of language features by inexperienced Ada programmers, could affect all stages of the process.

Implementing the conversion piece by piece would require fewer resources at any one time, but would considerably slow the conversion. The benefits of a piecemeal conversion would be that the correctness of the design can be checked at each step, and that technical problems can be discovered and corrected early.

The other aspect of conversion approach that needs to be considered carefully is whether to redesign applications, as opposed to merely translating existing code. One of DOD's main goals in using Ada is to reduce maintenance costs, but applications would have to be wholly redesigned before such savings could be achieved because the existing applications were not designed to exploit Ada's capabilities. Code translators offer a faster approach to producing Ada programs, but translators cannot fully use Ada's strengths, nor can they improve system design or efficiency. Furthermore, translations are not always correct or complete, and particularly in such cases, the translated product may require more maintenance than the original.

Software Tools and Resources

CASE Tools

Computer-Aided Software Engineering (CASE) tools help streamline and automate the design process. Programming in Ada requires conscientious attention to detailed specifications, making CASE tools particularly useful. Several CASE tools are available for Ada, including an upcoming release of KnowledgeWare, and Hierarchical Object Oriented Design (HOOD).

KnowledgeWare is working on an Ada code generator, in response to the Government's ICASE (interactive CASE) procurement. This new release will not affect the Planning and Analysis workstations: the workstations will continue to support KnowledgeWare's information engineering approach. Additions to the Design workstation will support Ada language constructs. A new Construction workstation (CWS-Ada) will generate Ada code, but the software will not translate existing COBOL program designs into Ada code.

Although KnowledgeWare will support code reuse via Ada packages, the developer has not finished working out the details of this support.

KnowledgeWare will support the Ada compilers by AETECH and Alsys. KnowledgeWare is targeting the UNIX and POSIX operating systems, as well as SQL* (in support of the ICASE procurement).

All current KnowledgeWare workstations would have to be upgraded to use these forthcoming Ada features. Depending on the status of current maintenance contracts, it may prove more cost-effective to buy new workstations. KnowledgeWare's prototyping tools probably will not be suitable for the current PAX environment because they are oriented toward full-screen applications and SQL. For these reasons, KnowledgeWare is not well suited to Ada and the PAX environment.

One good candidate for evaluation is Hierarchical Object Oriented Design (HOOD), marketed by Meridian. HOOD was developed by the European Space Agency for requirement analysis and structured design. Among HOOD's tools are data-flow diagrams, and entity-relation diagrams which construct visual models of relationships among objects in the system so programmers may examine them more easily. The DOS version of HOOD costs $795; the Microsoft Windows version costs $995.

*SQL: Structured Query Language.

Database and Software Interfaces

PAX is now a relatively open system that relies on many commercially available tools to reduce software development costs. One of the most heavily used tools is the FOCUS database management system (DBMS). On the microcomputer, many C-oriented tools are used but, unfortunately, Ada cannot use them. In fact, there are no Ada interfaces to most tools used in the PAX environment. This lack of interfaces is one of the major drawbacks to using Ada with PAX.

Database Interfaces. Nearly all PAX mainframe applications use FOCUS databases, but there is no Ada binding to FOCUS. That is, Ada programs cannot use FOCUS programs or tools. CAPCES uses FOCUS 4GL (FOCEXEC) to update and report from its databases, so CAPCES is a very poor candidate for implementation in Ada. It would require a thorough redesign, which would be very difficult and seriously inconvenience users.

The DD 1391 Processor uses the Host Language Interface (HLI)—another FOCUS resource—for its database accesses. If a binding can be developed to replace the HLI interface, the work of converting the DD 1391 Processor could be reduced to translating the COBOL programs to Ada. This alone, however, would help neither with maintenance nor operating costs. To exploit Ada's capabilities, at least a partial redesign would be required, plus extra work to try to minimize operating costs in the new language. Even if an Ada implementation were made to work, the new system would not be portable to other Army operating environments because of its dependence on FOCUS.

If a binding cannot be developed to replace HLI, or if the PAX team decides to switch to an SQL-based DBMS, then CAPCES, the DD 1391 Processor, and all other PAX applications will have to be completely redesigned because of the unavoidable changes to database interfaces and design. Such a change would also increase operating costs because the applications are currently designed to take advantage of FOCUS's hierarchical structure. The sole benefit would be that the resulting applications would be more portable, because Ada and SQL are Army standards.

The attempted conversion would run up significant costs because the converted code would inevitably contain bugs of its own. This would also irritate and inconvenience the user community. Trying to redesign all PAX applications while learning Ada would overload system maintainers for years. Migrating to Ada in stages would make the conversion more feasible, but would also dramatically lengthen the process. The PAX team would have to set the overall objectives and develop plans for each application to best meet those objectives as resources permit.

Languages and Commercial Library Interfaces. PAX uses several languages and commercial libraries to simplify application development and improve the user interface. The principal languages used by PAX are Fortran, COBOL, REXX, and C. Commercial libraries include the MicroFocus COBOL Runtime Library and C-Scape. Because PAX software uses widely supported commercial interface standards, it is easy to mix languages and libraries in one application.

Ada is far more restricted. Validated Ada compilers are not required to have any interfaces to external software. In fact, Ada vendors generally do not support commercial interface standards for the PAX environment. Consequently, there are no Ada interfaces to most software and tools used in the PAX environment.

There are several sources of reusable Ada code, some of which are publicized in the *AdaIC Newsletter*. However, because these typically are not commercial products, support is a major concern. Problems include finding a package of reusable Ada code that does what is needed, learning whether the package works in the PAX environment, debugging the software, and obtaining documentation.

Compiler Capabilities and Costs

Most Ada compilers are designed for UNIX or POSIX workstations and embedded systems. Few vendors offer validated Ada compilers for VM/CMS* mainframes or DOS-compatible microcomputers. Compiler vendors are discussed in this chapter, along with prices and any distinctive features.

Mainframe Ada Compilers

IBM and Alsys are the principal vendors of Ada compilers for the VM/CMS mainframe environment. Both vendors bundle tools to help develop programs. Alsys also provides this development environment for DOS-compatible microcomputers, but IBM does not.

IBM. IBM's Ada compiler is Ada/370, Release 3. This compiler includes a runtime package, which the compiler itself requires to operate. The compiler includes tools for program development, including a Make utility and a source-code reformatter.

Ada/370 includes interfaces for Fortran and C. There is also an optional SQL interface called SQL Module Processor for DB2. This product is designed to work with DB2, Version 2, Releases 2 and 3, under the MVS mainframe operating system, but does not work with SQL/DS under VM/CMS. If IBM does not consider the market large enough to justify the cost of developing a SQL/DS version, one would have to be developed for PAX on a contract basis.

Pricing for the Ada compiler and runtime package depends on the machine group number. Customers may pay either a one-time charge or a monthly lease, as shown in Table 1.

Table 1

IBM Ada Products

Monthly Lease (GSA Prices)	Ada/370	Runtime	Total
Group 30 (9370)	$1,280	$256	$1,536
Group 40	$2,910	$582	$3,492
Group 50	$3,635	$727	$4,362
One-Time Charge (GSA pricing is not available)			
Group 30 (9370)	$64,510	$12,900	$77,410
Group 40	$146,650	$29,330	$175,980
Group 50	$183,200	$36,640	$219,840

Source: IBM Corp.

*VM/CMS: Virtual Machine/Conversational Monitor System.

Alsys. The Alsys environment for VM/CMS includes a Make utility, a source-code reformatter, and a cross-reference utility. An assembler interface is also provided.

Alsys has a binding available for the ORACLE DBMS, but not for IBM's SQL/DS.

A runtime library is included in the price of the Alsys compiler. Customers may choose to pay a onetime charge with a yearly support fee, or a monthly license fee with an initial fee of $8,400. Prices are shown in Table 2.

Ada Compilers for Microcomputers

Vendors currently offer two basic types of microcomputer Ada compilers: compilers designed for 32-bit machines and those designed for 16-bit machines. While the 16-bit compilers will run on any microcomputer, the 32-bit compilers require an 80386 processor or higher. The 32-bit compilers are newer, and have more features than their 16-bit counterparts. Also, the 32-bit compilers can produce either 32-bit or 16-bit code, while the 16-bit compilers can only generate 16-bit code.

For serious development, the best approach is to use a 32-bit compiler to generate 16-bit code. This requires developers to use 80386-class microcomputers but end users may use any DOS-compatible machine. This approach makes more compiler features available to the developer.

Development tools should be part of any compiler purchase. These tools are generally bundled with mainframe compilers, but may be priced separately for microcomputer products. Recommended tools include:

- a debugger, preferably a source-level debugger
- a Make utility option to help with recompiling programs
- a cross-reference facility to help locate the use of variables and subroutines
- a source-code reformatter to standardize the way code looks, and to help locate syntax errors.

Table 2

Alsys Ada Development Environment

Monthly Lease Initial fee: $ 8,400 Minimum term: 12 months	Compiler	Yearly Support Fees
Group 30	$1500	included
Group 40	$2500	included
Group 50	$3500	included
One-Time Charge		
Group 30	$51,500	$10,300
Group 40	$81,500	$16,300
Group 50	$105,500	$21,100

Source: Alsys.

Generally, prices for microcomputer-specific development tools consist of the purchase price and a support fee. Some vendors bundle tools with the compiler in special packages, while others offer tools separately.

AETECH. AETECH's Ada compiler is called IntegrAda. The AETECH environment is probably the most developer-friendly of all the Ada compilers for microcomputers. The AETECH compiler includes an interface to assembler, and the company will release a C interface soon.

The AETECH Programmer's Deluxe Package (32-bit version) includes the following:

- compiler
- tools for design, source code generation, editing, analysis, configuration control, and documentation of Ada source code
- assembler
- AdaGraphics
- training and reference module
- hypertext-based Ada reference manual
- first year's maintenance fee.

The deluxe package costs $3,350. The AdaScope debugger costs an additional $895, and yearly maintenance fees are approximately $635.

Alsys. The Alsys Ada compiler for microcomputers, FirstAda, comes with an interface to the 32-bit Phar Lap Assembler and Metaware's Hi-C.

FirstAda is a complete software development environment, containing:

- Alsys compilation system
- debugger
- cross-reference generator
- Make utility
- reformatting tool
- complete documentation.

The product license fee is $2,995. Quantity discounts are available. The fee for upgrades and standard support is $860 per year, and is mandatory the first year.

Janus/Ada. The Janus compiler includes an interface to the Microsoft assembler. The Janus/Ada development system includes:

- compiler
- runtime libraries
- debugger
- programming support environment, including an Ada-aware editor
- assembler
- reformatting tool
- Make utility
- profiler (an optimization tool)
- screen-handling utility
- Ada tutorial.

The price for the complete package is $500. The product support fee is $100 per year. Site and network licenses are also available.

Meridian. Meridian's compiler, OpenAda, includes an interface to Microsoft C 6.0. Meridian has a database binding to Oracle, and is developing a binding to Paradox. The complete system includes:

- compiler
- Ada compiler environment
- optimizer
- debugger
- cross-reference tool
- Make utility.

The basic price is $849, and the maintenance fee is $200 per year.

There is also a version of OpenAda for developing programs to run under the Microsoft Windows operating environment. It includes a binding to Windows. The price is $795, and the annual maintenance fee is $200.

Programmer Training

Several programmer training options are available from Ada vendors. Janus includes an online training module with its compiler. AETECH offers two different training modules: one for $195, and a more extensive course for $250. The Alsys training module is priced at $750.

Through Meridian, a five-day onsite training course for up to 12 people is available for $10,000. Individuals can attend the course for $1,500 each. Many other groups also offer Ada training, with prices comparable to Meridian's. Information on training in Ada is available through the Ada Information Clearinghouse (AdaIC). The AdaIC telephone number is 703-685-1477.

Ada's strong software engineering orientation and advanced features make the language difficult to learn, so classroom training is recommended for anyone developing PAX applications in Ada.

4 DISCUSSION

Ada encourages the use of software engineering principles that can make applications more reliable and easier to maintain, but converting PAX to Ada would pose several complex technical challenges. Converting PAX to use Ada would cut away the underpinnings of existing PAX applications, because Ada offers no interfaces to PAX tools (e.g., FOCUS). PAX systems would have to be fundamentally redesigned to operate under Ada and exploit the benefits of the Ada language. Redesigning PAX to such an extent would be very costly, but the advantages would be questionable at best.

The costs of converting PAX to Ada would be very high. The most obvious costs would include:

- Ada training for all PAX programmers, system designers, and maintainers
- respecification and redesign of all current applications
- coding, compiling, and testing converted programs
- converting PAX data files for use in an Ada environment
- compiler license and maintenance fees
- fees for any other software tools or resources required.

Some of these conversion costs (e.g., software license and maintenance fees) are fairly straightforward to quantify (see Chapter 3). Others are more difficult. For example, without running benchmarks there is no way to quantify Ada runtime costs and compilation costs in PAX. These costs may be significantly higher than for the current PAX system.

The authors polled industry experts and consultants for cost data pertaining to converting large systems to Ada, but no concrete data are available. The people redesigning systems for Ada on this scale are not breaking down their costs in a way that allow the computation of solid cost figures. Nevertheless, it seems clear that the indirect or intangible costs of redesigning PAX for Ada would be enormous. Such indirect costs could include system downtime, unavailability of data, reduced programmer and administrator productivity during the conversion, etc.

The authors estimate that conversion costs would be likely to push the payback period for Ada conversion beyond 6 years—assuming that the Ada implementation operates at least as efficiently as the present PAX system.

A quicker way to cut PAX system costs would be to move as much processing as possible to microcomputers. But even for microcomputer applications, Ada cannot be recommended above C or C++. As discussed in Chapter 2 under "Comparison of Current PAX Languages With Ada," C and C++ are more economical in the PAX microcomputer environment because they interface easily with commercial tools. In addition, C expertise is readily available. Finally, C and C++ executables are faster and smaller than Ada executables, which would probably lead to lower operating and computing costs.

Ada's most commonly cited advantages actually are not exclusive to Ada. The most significant benefits that would be realized by a conversion to Ada are actually attributable to the object-oriented approach—the particular language employed is irrelevant. Object-oriented design concepts lend themselves well to any object-oriented language (and, with a little difficulty, even to non-object-oriented languages). Similarly, the faster development times and lower maintenance costs typical of object-oriented approaches are also language-independent. From this perspective, Ada is no better than C++ or Object-Oriented Pascal—and Ada's other drawbacks weigh heavily against it.

Some cases are documented in which code translated into Ada has led to impressive monetary savings. See *Ada Transition Research Project* (Hobbs et al 1990; Hobbs et al 1992) for a particularly good example. However, in such cases, the monetary savings typically result not from Ada but from interactivity: instead of requiring an operator to haul magnetic tapes halfway across the state, for example, the redesigned systems intelligently exploit the power of desktop machines to do most of the necessary processing. Ada itself is not responsible for the savings. The same interactive program written in C++, for example, would have the same results—perhaps better results, given C++'s typically faster and smaller executables.

Ada's two most commonly cited strengths are its built-in encouragement of preplanning and its strong type checking. Yet a sufficiently determined programmer—or even a merely lazy one—can easily defeat or ignore Ada's encouragement of good programming practices. Writing code that takes advantage of Ada's better features requires discipline. And the same discipline applied to code written in other languages leads to an equally good product. In short, programmer discipline is the key to better, less expensive applications—not Ada.

5 SUMMARY AND RECOMMENDATIONS

Summary

The main advantages of converting the current PAX system to Ada include the following:

- Ada is the DOD standard, so no waivers are required to use it.
- Ada is a portable, highly standardized language. DOD has an ongoing validation program to ensure that compilers observe the current language standard.
- Coding in Ada does not lock developers into a vendor-specific environment. Ada is available on a growing number of platforms.
- Several sources of reusable Ada code and other potentially helpful utilities are available.
- Ada readily supports structured design techniques, and strongly encourages the use of software engineering techniques.

The main disadvantages of converting PAX to Ada include the following:

- Ada currently has no interfaces to most PAX tools and languages.
- Ada has no interface to FOCUS, which PAX heavily relies on.
- Few Ada compilers will work in the PAX mainframe environment. Vendor support for the ones that do work may be limited, because VM/CMS users are not the vendors' primary market.
- Substantial time and resources would be required to locate, evaluate, and employ reusable Ada code.
- Without running benchmarks, there is no way to quantify Ada runtime costs and compilation costs in PAX. These costs may be significantly higher than the current PAX system.
- No PAX programmers currently know Ada. Training PAX programmers, designers, and maintainers in Ada would be very expensive and time-consuming. Hiring Ada programmers without PAX expertise would be no less expensive or time-consuming.

Recommendations

For reasons of cost and technical considerations, it is recommended that PAX not be converted to Ada at this time.

If PAX managers ultimately decide to move to an Ada environment, or if they are so required by higher headquarters, the following preparation tasks are recommended:

1. Gain Ada expertise using microcomputers, which will enable programmers to learn the language with minimum expense. Using relatively small applications, the PAX team can compare redesign and code-translation approaches.

a. Redesign a small PAX microcomputer application in Ada to estimate the costs of writing, testing, and maintaining Ada programs. Relative program size and ease of application distribution should also be evaluated. A good candidate for this redesign test would be PC Dugout, because of its small size and lack of external interfaces.

b. Acquire a COBOL-to-Ada translator and convert an existing COBOL microcomputer application into Ada. PC ISCE would be a strong candidate for this approach. This conversion will enable the PAX team to evaluate the feasibility of straight COBOL-to-Ada code translation.

c. Acquire a Fortran-to-Ada translator and convert an existing PC Fortran application, such as PC-ECONPACK, into Ada. This will allow an evaluation of Fortran code translation and will develop more Ada expertise within PAX.

2. Gain Ada expertise using the mainframe. Try to find or build a FOCUS interface to an Ada compiler. Then develop a benchmark to compare Ada operating costs to the current system. This step will help identify the feasibility of an Ada interface to FOCUS, the operating costs of the Ada translation, and the similarities and differences between microcomputer and mainframe Ada environments.

3. After using the results of the two preceding tasks to assess more accurately the costs, benefits, and difficulties of converting to Ada, develop a final conversion plan and identify the best approaches and tools.

a. Determine the optimal hardware, software, and DBMS platforms to support PAX applications in Ada.

b. Determine the applicability of distributed database technology in a new PAX system.

c. Identify functions that could be moved to the microcomputer and those that must remain on the mainframe.

d. Specify the interface between microcomputer and mainframe functions.

e. Determine the conversion approach for each PAX subsystem—that is, decide whether a given subsystem should be rewritten in Ada, code-translated into Ada, or left alone?

f. Develop a conversion schedule, along with a list of the resources the conversion will require.

REFERENCES

Hobbs, Reginald L., Joseph J. Nealon, and Richard Wassmuth, *Ada Transition Research Project (Phase I)*, ASQB-GI-91-005 (U.S. Army Institute for Research in Management Information, Communications, and Computer Sciences [AIRMICS], December 1990).

Hobbs, Reginald L., John R. Mitchell, Glenn E. Racine, and Richard Wassmuth, *Ada Transition Research Project (Phase II)*, ASQB-GI-92-004 (AIRMICS, May 1992).

Japel, Edward J., John R. Murphy, Wayne J. Schmidt, and Connie L. Raaymakers, *PC Dugout Micro-Mainframe Integration System*, TR P-91/34/ADA237480 (USACERL, May 1991).

Public Law 101-511, *Department of Defense Appropriations Act, 1991* (5 November 1990).

ABBREVIATIONS AND ACRONYMS

AdaIC	Information Clearinghouse
CAPCES	Construction Appropriation, Programming, Control, Execution System
CASE	Computer-Aided Software Engineering
CWS–Ada	Construction workstation–Ada
DBMS	database management system
DOD	Department of Defense
ECONPACK	Economic Analysis Package
HLI	Host Language Interface
HOOD	Hierarchical Object Oriented Design
MCA	Military Construction, Army
MIS	management information system
MS–DOS	Microsoft Disk Operating System
MYPLAN	Multi-Year Plan Applications Subsystem
PAX	Programming, Administration, and Execution
PAXID	PAX logon identification
RAPID	Reusable Ada Products for Information Systems Development
USACE	U.S. Army Corps of Engineers
VM/CMS	Virtual Machine/Conversational Monitor System

DISTRIBUTION

Chief of Engineers
ATTN: CEHEC-IM-LH (2)
ATTN: CEHEC-IM-LP (2)
ATTN: CERD-L
ATTN: CEMP-P

Fort Belvoir, VA 22060
ATTN: CECC-R

Defense Technical Info. Center 22304
ATTN: DTIC-FAB (2)

9
3/94

☆ U.S. GOVERNMENT PRINTING OFFICE: 1994—3510-S/00016