2

Special Report

CMU/SEI-91-SR-4

Carnegie-Mellon University
Software Engineering Institute

AD-A253 646

A Comparison of Ada 83 and C++

Nelson H. Weiderman

June 1991

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

DTIC
ELECTE
JUL 30 1992
S B D

92 7 27 212

92-20296

Special Report
CMU/SEI-91-SR-4
June 1991

A Comparison of Ada 83 and C++

Nelson H. Weiderman
Technology Division

Approved for public release.
Distribution unlimited.

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

This technical report was prepared for the

SEI Joint Program Office
ESD/AVS
Hanscom AFB, MA 01731

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

Review and Approval

This report has been reviewed and is approved for publication.

FOR THE COMMANDER

[illegible]

John S. Herman, Capt, USAF
SEI Joint Program Office

The Software Engineering Institute is sponsored by the U.S. Department of Defense.

This report was funded by the U.S. Department of Defense.

Copyright © 1991 by Carnegie Mellon University.

This document is available through the Defense Technical Information Center. DTIC provides access to and transfer of scientific and technical information for DoD personnel, DoD contractors and potential contractors, and other U.S. Government agency personnel and their contractors. To obtain a copy, please contact DTIC directly: Defense Technical Information Center, Attn: FDRA, Cameron Station, Alexandria, VA 22304-6145.

Copies of this document are also available through the National Technical Information Service. For information on ordering, please contact NTIS directly: National Technical Information Service, U.S. Department of Commerce, Springfield, VA 22161.

Use of any trademarks in this report is not intended in any way to infringe on the rights of the trademark holder.

Table of Contents

Accession For	
NTIS GRA&I	✓
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

DTIC QUALITY INSPECTED 2

A Comparison of Ada 83 and C++

Abstract: The purpose of this report is to provide technical input to the Deputy Assistant Secretary of the Air Force for Communications, Computers, and Logistics to assist that office in preparing a business case for using Ada or C++ to develop Corporate Information Management (CIM) systems. This technical input has been gathered by using the comparison methodology of a 1985 Federal Aviation Administration (FAA) report as a model, as well as by conducting interviews with experts in Ada and C++. The conclusion of this report is that technically neither language is clearly better than the other; for government use, however, there is clear justification and rationale for using Ada rather than C++ for large complex systems with long lifetimes.

Executive Summary

The purpose of government efforts to standardize is not to ensure that everyone in government is using the most modern technology. Rather, the purpose of government standardization is to reduce cost. Using a common high order language to develop software for government systems is desirable because it increases the ability to: use different software systems together, reuse software systems and components, transport software and personnel across departments, and maintain software over long lifetimes. A common language must necessarily be a general purpose language, which may be less suitable for a given application than a language designed specifically for that application. In government, the optimal solution is rarely optimal for specific applications in terms of cost or technology.

Ada is the high order language required by DoD directive as well as by the public law. There has been a significant investment in the Ada standard by both the public and private sectors. This investment is just starting to pay off in completed weapon systems, completed command and control systems, and completed information systems. The private companies that invested in Ada are now producing production quality tools and contractors are producing high-quality software. More importantly, the use of Ada has been accompanied by a growing awareness that large systems must be designed and developed with software engineering discipline.

The C++ language is an extension of the C language and was developed at AT&T by Bjarne Stroustrup. For the most part, the new features provide support for better software engineering practices. C++ has gained rapidly in popularity since 1986, when the first reference manual was published. The rapid growth can be attributed, at least in part, to the large number of installations of C and UNIX. More than other modern programming languages, C++ has been associated with the object-oriented programming paradigm.

It is futile to try to make a comparison of general purpose computer languages on technological grounds alone. In particular, there is no clear answer to the question of whether Ada or C++ is a better programming language. The languages come from different programming cultures with different priorities. Interminable arguments result from comparing the features of one programming language with those of another. For example, Ada has an abstraction mechanism called a package for encapsulating types and procedures with some common theme. C++ has an abstraction mechanism called a class for encapsulating types and procedures in a different way. Which capability is better is still conjecture.

However, the following is clear. Both Ada and C++ are adequate for writing programs for information systems. Both Ada and C++ are better than Pascal, C, or assembly language because they are higher level languages and address some of the problems of developing large, complex, software systems with long lifetimes. Those interviewed for this study who are familiar with both Ada and C++ believe that Ada is probably the better choice for such systems. However, the choice of language is far less important overall than are the socioeconomic issues and the processes used to develop systems.

Socioeconomically, the distinctions between Ada and C++ are much clearer. The price of making Ada a real, rather than hollow, standard has already been paid. Since the draft standard was released in 1980, it has taken 11 years of considerable effort to institute the technology. There have been other costs associated with articulating what the standard means; with immature language implementations and immature tools; with convincing intransigent contractors to use Ada; with developing the training; and with developing prototype systems for proof of concept. There have also been costs associated with developing secondary standards for numerical software, as well as interface standards for database software, user interface software, and graphics software. If another language is chosen to replace or supplement Ada, many of these costs will have to be paid a second time.

Commercial de facto standards such as C++ have the advantages of widespread visibility and acceptance. The marketplace moves rapidly to ensure that C++ can work with other software systems. C++ is ahead of Ada in this regard. Much commercial investment has been made in the infrastructure for tools and training. Despite these advantages, the price of maturation and acceptance of the language within government will have to be paid again if a new language is to play a substantial role.

This study compares Ada and C++ according to six criteria categories established in a 1985 Federal Aviation Administration (FAA) study comparing five programming languages. In the categories of efficiency and life-cycle cost, the differences between Ada and C++ are insignificant, with C++ having an edge in the first and Ada in the second. In the categories of maintainability/extensibility and risk, Ada has a small advantage over C++. In the categories of capability and availability/reliability, Ada has a significant advantage over C++ at the present time. C++ had lower ratings overall, partly because it is a rather new and untested relative of Ada. When Ada 9X is introduced in a few years, a similar period of instability and immaturity of the language and its compilers can be expected.

Ada may not be an optimal programming language for information systems, but it is an adequate programming language for that purpose, and, more importantly, it is a standard, it is stable, and it is reasonably mature. C++ is also an adequate programming language, but unlike Ada it is not a well-defined standard, it is not stable, and it is not mature. Those who were consulted for this study could provide scant evidence of large systems being developed in C++ outside of AT&T and little basis for believing that C++ would reduce life-cycle costs for developing information systems. The technical problems of using Ada for information systems, such as providing interfaces to window systems and commercial-off-the-shelf software, awkward I/O, and lack of mathematics for decimal arithmetic, have been solved, albeit in a less than optimal way. For these reasons, it will be difficult to justify waivers to use C++ for large complex information systems.

1. Introduction

The purpose of this report is to provide technical input to the Deputy Assistant Secretary of the Air Force for Communications, Computers, and Logistics to assist that office in preparing a business case for using Ada[1] or C++ to develop Corporate Information Management (CIM) systems. This technical input has been gathered by using the comparison methodology of a 1985 Federal Aviation Administration (FAA) report as a model, as well as by conducting interviews with experts in Ada and C++.

1.1. Scope of This Study

Public Law 101-511, Section 8092 prescribes, "Notwithstanding any other provisions of law, after June 1, 1991, where cost effective, all Department of Defense software shall be written in the programming language Ada in the absence of special exemption by an official designated by the Secretary of Defense." The law has been interpreted to exclude commercial, off-the-shelf (COTS) software and other end-user software like spreadsheets, and "cost effective" has been interpreted to mean life-cycle costs rather than development costs.

The public law raises the status of already existing Department of Defense (DoD) policy as specified in DoD Directive 3405.1, which in Section D. 3. b states that "Ada shall be used for all other applications [i.e., other than intelligence, command and control, and embedded systems], except when the use of another approved higher order language is more cost-effective over the application's life-cycle, in keeping with the long-range goal of establishing Ada as the primary DoD higher order language (HOL)."

To determine whether to waive the requirement to use Ada in developing its information systems, DoD must consider more than the technical features of Ada and C++. Therefore, this report includes more than a comparison of language features. It covers the broader range of technical, economic, and social issues surrounding the choice of language and granting of waivers as they might influence public policy. Because Ada is the mandated language, it is assumed that there must be compelling reasons, justified by life-cycle cost, to waive the Ada mandate.

1.2. The Corporate Information Management Environment

A Plan for Corporate Information Management for the Department of Defense [17] defines information as one of four resources that must be managed by any organization (the others being capital, materiel, and labor). The plan views information management not as the automation of existing business methods but as the application of computing and communication technology in new and creative ways. The scope of the plan is limited to business

[1] In this instance, and for the remainder of this report, Ada refers to Ada 83, ANSI/MIL-STD-18154-1983.

functions, namely managing personnel, materiel, and financial resources. Command and control is not included in the initial scope, but is subject to reassessment. Embedded weapon systems are specifically excluded.

The DoD information systems inventory is written in many different languages. COBOL predominates, with estimates ranging from 50% to 80% of the total source lines. Other programming languages used include FORTRAN, C, Pascal, Basic, PL/I, and Mumps. Ada source lines constitute less than 5% of the management information system (MIS) total and possibly less than 1% of the overall total. There is little or no C++, CMS2, or JOVIAL in DoD MIS applications.

Use of COTS software is increasing in MIS development. For smaller applications such as spreadsheets, word processing, and desktop publishing, COTS is used extensively. For large applications it is still a small but growing percentage of the software. COTS is sometimes chosen and augmented or modified for DoD use. There is no DoD standard for database systems, but database products must comply with the Federal Information Processing Service (FIPS) definition of the Structured Query Language (SQL). Ada has not been a predominant language in the DoD MIS environment because Ada was designed to satisfy the requirements of embedded weapon systems and is generally perceived as not adequately addressing the problems of transaction processing and accessing large databases.

The current situation is described in the CIM plan as follows:

> Very few common information systems have been developed within the Department. Existing federal and DoD development policies have encouraged individual, non-integrated systems development efforts. Efforts to standardize systems for certain functions, such as pay and personnel, received strong emphasis in the Reform 88 initiative, but little success was achieved because the efforts focused on technical systems. Thus, in DoD today, there are 27 payroll systems, which is still a reduction from several years ago. Systems are complex and expensive, retraining costs are high, and organizational flexibility is degraded by "unique" systems [18, p.16].

The CIM plan calls for centralized control and decentralized execution. Technologies specifically mentioned in the vision of the future include heterogeneous, open system architectures, standards critical to portability and interoperability (including networking/communication standards, language standards, database standards, and standards for graphically oriented windowing), data modeling tools and methodologies, software development methodologies, and distributed systems.

1.3. Ada and C++

The languages being considered in this report are the 1983 version of the Ada language standard (ANSI/MIL-STD-1815A-1983) and the 1990 version of C++ as defined in *The Annotated C++ Reference Manual* [12]. An implementation of this definition of the language is available from AT&T as Release 2.1. A limited number of comments will be made about the proposed revision of the Ada standard, referred to as Ada 9X, or to the experimental features of C++ defined in Chapters 14 and 15 of the reference manual. Ada 9X is in the early stages of development and an implementation of the C++ extensions will be available from AT&T as Release 3. References to these proposed modifications to the languages are noted in this report.

1.4. Methodology

Information for this study has been gathered through research on studies comparing languages and through interviews of experts outside the SEI. A list of references consulted is included at the end of this report and a list of experts interviewed is given in Appendix A. One of the references that received special attention was the FAA study conducted by IBM that compared five languages in 1985 [18]. This was used not so much for its results, as for its methodology and its evaluation criteria.

The methodology for this study consists of two major components:

1. One component of the study is use of the comparison methodology of the 1985 FAA report as a model in order to perform a comparative analysis of the Ada and C++ programming languages for use in MIS. The conclusions of that report are updated appropriately in this study.
2. Another component is the interviewing of various experts in Ada and C++. Every attempt has been made to select individuals who either know both languages very well, or have a special perspective on one or both languages. Emphasis has been placed on finding people who have actually used both languages to build large, complex systems. Others were chosen for their expertise in language issues or standardization, for their knowledge of a particular application written in one of the languages, or for their knowledge of the MIS application domain.

The study represents our best technical judgement on the use of C++ and Ada in MIS at the present time.

2. Background on Ada and C++

There are several significant historical and cultural differences between Ada and C++. Arguments can be made for each end of the spectrum, depending on one's point of view and preferences. Each of the following points is meant to illustrate the dichotomy, not to suggest that either end of the spectrum is better:

- Ada had a three-year requirements phase with input from many individuals from many constituencies. C++ never had a formal requirements phase. It was designed so that software developers at AT&T could program in a high order language similar to C.
- Ada was not constrained by any other programming language, although it was based loosely on Pascal. C++ was designed to be an upwardly compatible extension to C.
- Ada had a three-year design phase with input from many individuals from many constituencies. C++ never had a paper design. The design, documentation, and implementation went on simultaneously.
- Ada has been stable and tightly controlled. C++ is a dynamic language that has evolved, and continues to evolve, according to the needs and problems of users.
- Ada emphasizes support for development efforts with teams of programmers, each writing small sections of code. C++ emphasizes increasing the amount of code that can be handled by a single developer.
- Ada places functions such as tasking, I/O, consistency checking, and library control within the language. C++ places all these functions outside the language and under the control of separately provided tools.

These historical and cultural factors make it especially difficult to compare the languages on an equal basis. The languages address different constituencies with different perspectives and needs. There have been attempts to compare languages by feature—the compendium of papers collected by Feuer and Gehani [14], which compare Ada, C, and Pascal, and attempts to compare Ada to C plus UNIX [16]—but these studies are not scientific and are not very satisfying.

2.1. Ada History and Design Goals

The genesis of Ada can be traced to early 1975 when a working group on high order languages (HOLWG) was chartered by the DoD to investigate establishing a minimal number of common languages for use in embedded systems. After developing initial sets of requirements, the group found that no existing language satisfied the requirements well enough to be adopted as a common language. Five sets of requirements were written, culminating in a Steelman requirement in 1978, at which time the language designs of four contractors were evaluated to determine which design best met the Steelman requirement. The winning contractor, Cii Honeywell Bull, produced a 1980 language description, which was modified before becoming the current standard in 1983. The first Ada compiler was validated in 1983.

The Ada 9X Project was initiated in October 1988 and has recently completed a two-year requirements gathering process. The goal is to complete a revision of Ada 83 by 1993, the ANSI deadline for restandardization. Among the design goals for Ada 9X are:

- A conservative tradeoff between user needs and the impact on existing Ada applications and tools.
- Maximum upward compatibility.
- More precise language definition.
- Convenient interfaces to external systems, other languages, and other standards.
- Simplification and unification of language rules.

The Reference Manual for the Ada Programming Language [24] cites three overriding design goals: program reliability and maintenance, programming as a human activity, and efficiency. Emphasis was placed on program readability over ease of program writing. It was a design goal to avoid error-prone notations and encoded forms in favor of English-like constructs. The idea of development from independently produced software components was also central to the design. Language constructs were examined in light of implementation techniques available at the time and rejected if they led to inefficient use of storage or execution time.

2.2. C++ History and Design Goals

The C++ language is a superset of the C language developed at Bell Labs by Bjarne Stroustrup. C++ was first released at AT&T in the summer of 1983. Release 1.0 of the language was specified in Stroustrup's *The C++ Programming Language* [23] published in 1986. Release 1.1 added pointers to class members and the protected keyword. Release 1.2 added the ability to use unsigned integers and unsigned long integers to distinguish one overloaded function from another. It was with the AT&T cfront preprocessor for Release 1.2 that C++ grew in popularity.

The current language definition is the February 1990 definition, which was chosen by the American National Standards Institute (ANSI) to serve as a starting point for the formal standardization of C++. This definition is described in *The Annotated C++ Reference Manual*, by Ellis and Stroustrup [12]. The 1990 software release, Release 2.0, fixed problems and introduced new features, including multiple inheritance, type-safe linkage, abstract classes, and refined mechanisms for overload resolution. The current version of C++, Release 2.1, lists two features as "experimental": templates (a form of generics) and exception handling. Currently being tested by AT&T, these features will be available in some form with Release 3 of the language.

Simplicity and runtime efficiency are two important design goals of both C and C++. Features that would incur runtime or memory overheads were avoided. "C was used as a base language for C++ because it (1) is versatile, terse, and low-level; (2) is adequate for most system programming tasks; (3) runs everywhere and on everything; and (4) fits into the UNIX programming environment" [23, p.4.].

2.3. Standardization and Validation

Language standardization and language validation are separate issues. Standardization is a rather long process that results in a common understanding of the syntax and semantics of a programming language. Standardization generally takes many years of effort by language experts and reviewers. Complex languages require delicate negotiations on fine points and highly legalistic interpretations. Language standards may aspire to mathematical formality, but the state of the practice is still to express them in natural language with its attendant ambiguity.

Language validation is the testing process that attempts to demonstrate the conformance of a compiler or interpreter with the language standard. In other words, the validation test suite is designed to demonstrate that for every program written in the language, the execution of the program conforms to the specifications in the standard. In spite of these aspirations, a test suite cannot guarantee absolute conformance. It can only provide a reasonable degree of confidence that a compiler or interpreter conforms reasonably to the standard.

The Ada standard was approved February 17, 1983. At that time, it became ANSI/MIL-STD 1815A-1983. Subsequently, it became an International Standards Organization (ISO) standard as well. Interpretations of the standard are made by an international committee that was originally under the auspices of the U.S. Department of Defense and called the Language Maintenance Committee, but now falls under the jurisdiction of ISO and is called Ada Rapporteur Group.

The mechanism for ensuring that Ada compilers conform to the standard is the Ada Compiler Validation Capability (ACVC), a suite of approximately 4000 test programs. This test suite has been updated periodically, but is now frozen at Release 1.11 and will be updated in conjunction with the development of the new standard (Ada 9X). It is unlikely that there will be a new Ada standard before 1993 or 1994.

The ANSI C standard was approved in December 1989 and ISO approval of the C standard is in progress. Through the National Institute for Standards and Technology (NIST), ANSI C will become a U.S. Federal Government standard called FIPS 160 effective September 30, 1991. A year later, it will become the mandatory standard for C. (FIPS standards already exist for Ada, FORTRAN, COBOL, and Pascal.) An ANSI committee (X3J16) has been meeting for approximately a year to standardize C++. Once a draft standard has been agreed to, it is distributed for balloting. Three ballots are required, each taking up to a year. It is thus unrealistic to think that there will be a C++ standard for several years.

Furthermore, there are those from the C community who believe that it will be quite difficult, if not impossible, to come to closure on a formal C++ standard. The first reason for this is that C++ is still evolving—new features are still being added to the language. The second reason is that because of the significantly greater complexity of C++, and the intricate, subtle interactions of features, convergence may not be possible.

For these reasons, as well as the fact that the commercial marketplace does not want a divergence of C++ standards and implementations, it seems likely that C++ will rely more on *de facto standards* than *de jure* standards. The AT&T *cfront* preprocessor has thus far been the de facto standard. In fact, the vendors strive to make their compilers "bug compatible" with cfront (meaning that compatibility and standardization come at the price of having the same errors in all implementations). This may be a less than optimal solution, but it does promote a certain level of portability among C++ implementations.

At least three companies sell validation test suites for C: Plum-Hall, Perennial, and Ace. Each of these suites contains several thousand test programs. NIST has chosen the Perennial suite for official distribution within the government to test conformance to FIPS 160. NIST will be initiating their testing service to validate C compilers on January 1, 1992. Katheryn Miles, specialist in the NIST validation group, expects that the validation suite for C will be "fairly fluid" for the next 2-3 years.

Perennial currently is distributing a test suite based on the UNIX Systems Laboratories (USL) C++ Language System, Release 2.1. The C++ Language System includes not only the cfront preprocessor, but also libraries like iostream, task, and complex, as well as the C compiler and other tools.

3. The FAA Language Selection Analysis Report

The FAA contracted with IBM in middle 1980s to study a number of high order programming languages for possible use on the Advanced Automation System (AAS) Program. The resulting report provides the rationale for the study and recommendations for the high order language (HOL) to be used. It also provides the analyses and recommendations for the use of assembly language.

Only those parts of the study that are relevant to a comparison of Ada and C++ are reviewed here.

3.1. Methodology

The first stage of the study was to identify candidate HOLs. Based on the FAA requirements, five languages were chosen: Ada, Pascal, C, JOVIAL J73, and FORTRAN. The second stage of the study was the evaluation of these five languages using comparative analysis and relative benchmarking. The third stage of the study was to draw conclusions based on the evaluation results and to make recommendations as to which candidate language was most suited for AAS.

To compare the five languages, the FAA study had two separate components: comparative analysis, which is conceptual and quantitative in nature, and relative benchmarking, which is empirical and experiential in nature. The comparative analysis (after which the current study is modeled) involved analysis of forty-eight criteria by a group of twelve experts, while the second component involved running two programs written in each language and comparing compilation times, execution times, and various measures of space requirements. The assessments in the comparative analysis were refined using a Delphi process (to approach consensus, experts are allowed to revise their original assessments after seeing the overall results).

For the purpose of the current study, only Ada and C will be considered. The rationale for choosing Ada and C as candidate HOLs was described in the FAA report as follows [pp. 9, 10]:

> Ada is the new ANSI/MIL-standard language. It was originally designed under DoD sponsorship for use in military embedded computer systems, but is gaining widespread acceptance as offering excellent support for modern software engineering principles. Ada represents the state of the art in programming language design and compiler technology, incorporating such features as abstract data types, concurrent processing, and exception handling. Ada's chief disadvantages are its newness, its apparent complexity, and the current scarcity and weak performance of implemented compilers for it.
>
> C is a language made popular by the advent of the UNIX operating system, with which it is closely associated. It is possible to code very efficient programs in C, since very few restrictions are imposed on the programmer. The syntax is terse

and occasionally enigmatic, causing some problems of readability. A well developed set of programming support tools has grown up around C and UNIX.

The relative benchmarking activity involved the coding of two benchmark programs in all five languages. The two programs were the Dhrystone benchmark [25] and the Average Salary Program [22]. The FAA study describes the Dhrystone as a "complex series of sophisticated data processing manipulations," and the Average Salary program as "a simple series of straightforward numerical calculations."

The performance results of the relative benchmarking activity are not particularly relevant here and will not be considered further.

3.2. Evaluation Criteria

For the comparative analysis section of the FAA study, the evaluation criteria were organized into six major categories. These 6 categories were further broken down into individual criteria within these 6 categories, for a total of 48 evaluation crit^eria [p. 28 ff]. The six major categories and their descriptions (taken from the study) are as follows [pp. 21, 22]:

1. Capability: facets of the implementation language relevant to programming or software engineering.
2. Efficiency: factors relevant to optimization of generated code and runtime utilization of resources (processor time memory space, etc.).
3. Availability/Reliability: factors influencing day-to-day safety of operational systems.
4. Maintainability/Extensibility: factors influencing long-term viability of operational systems.
5. Life Cycle Cost: elements of budgetable cost associated with or impacted by the implementation language.
6. Risk: areas of uncertainty or concern associated with or impacted by the implementation language.

Additionally, four "critical requirements" were identified in the FAA study [p. 16]. They are:

1. Commercial Support/Availability of Compilers
2. Availability of S/370 Hosted Compilers
3. Expectation of Continued Long Term Support
4. Suitability to the AAS Application(s)

3.3. Study Results

In the comparative analysis activity, a set of relative importance weights was established for each of the criteria from the opinions of twelve experts. Based on these weights and the scores of the candidates for each criteria, relative figures of merit were calculated, resulting in the following ranking of the candidate HOLs:

Ada	76.7
Pascal	63.3
C	59.5
JOVIAL J73	55.5
FORTRAN	47.0

The breakdown of these scores by the 6 evaluation categories is shown in Table III [p. 33] of the FAA report. Ada ranked highest in criteria categories 1, 3, 4, and 5. It ranked lowest in category 2 (Efficiency) and in the middle in category 6 (Risk).

The more important result of benchmarking was the analysis of the conversion effort, which tended to confirm the comparative analysis activity.

The conclusion of the FAA study [p. 12] was that the use of Ada was "in the ultimate best interest of the AAS program and its goals, and that warrants coping with the temporary risks/problems that loom large in the near term in order to reap the significant benefits/payoffs over the long term."

4. Deviations from the FAA Baseline

This section evaluates Ada and C++ by comparing the situation in 1991 with the situation in 1985 when Ada and C were compared by the FAA. For each of the 48 criteria categories, an evaluation is made as to whether Ada in 1991 ranks higher or lower than it did in 1985 and whether C++ in 1991 ranks higher or lower than C in 1985.

Each of the following subsections presents the description of the subcategory (taken from the original FAA report), the FAA rating of Ada and C, and the SEI rating of Ada and C++. There is a minimum of analysis or second guessing of the original evaluations. Following the scoring is a description of the difference between the earlier and current ratings. At the end of each of the six broad criteria categories is a summary of how the language or compilers have changed in the intervening years and an evaluation of how Ada and C++ compare in that dimension today. Finally, an overall summary is given.

The reader is cautioned that the scores from both the original FAA study and the current study are best technical judgements at a specific point in time, given limited time for investigation. In some cases, these scores may be assigned using incomplete knowledge or inconsistent data. For example, the compilers and benchmark tests used by IBM in the FAA study may not have been the best available in 1985. The SEI study was less systematic than the FAA study due to the lack of a consensus-building process. Some of the categories deserve ranges of values depending on special circumstances. The individual numbers and the summary numbers should not be assumed to be precise. Nevertheless, the conclusions reached can be given some weight as a whole, despite the possible individual anomalies in the 48 criteria categories.

4.1. Capability

Capability refers to the facets of the implementation language relevant to programming or software engineering. Because Ada has not changed at all since 1985, its rating relative to the 1985 baseline should be unchanged. Because C++ is a superset of C, new capabilities have been added and their relative value are considered. Fourteen subcategories are considered individually.

4.1.1. Capability scores

1. Data typing -- "Extent to which the language provides for explicit typing of data and enforces data typing consistency, in the sense that the type of a data item determines the values it may assume and the operations to which it may be subjected" [p. 22].

FAA Score:	Ada 5	C 3	[p. 76]
SEI Score:	Ada 5	C++ 4	

C was downgraded in the FAA report for lack of fixed point types and enumeration types. C provides strong data typing consistency only by means of a separate "program checker" tool called Lint. C and C++ have enumeration types [23, p.64], but without the functions of Ada's enumeration types. C++ does not have fixed point types as part of the language, but it does provide static type checking for the proper use of arguments in functions [11, p.11].

2. Data structures -- "Extent to which the language provides for the introduction and manipulation of composites of scalar data items, such as array, record, and pointer structures" [p. 22].

FAA Score:	Ada 5	C 3	[p. 77]
SEI Score:	Ada 4	C++ 4	

C was downgraded for inferior variant record data structures, and lack of assignments and comparisons of composite objects/variables such as strings. A user of C++ can construct objects whose size is not determined at compile time by taking control of allocation and deallocation [23, p.165]. Assignment of classes is permitted in C++, but if x and y are objects of the same class, x=y by default means a bitwise copy of y into x. This has the unfortunate side effect of invoking constructors and destructors [23, p.157]. Such anomalous behavior can be avoided, but it is a potential pitfall. An Ada advantage is the automatic discrimination of variant records, and a C++ advantage is the versatility of constructors and destructors for data objects.

3. Data abstraction -- "Extent to which the language provides for the introduction and manipulation of new programmer-defined data types" [p. 22].

FAA Score:	Ada 5	C 2	[p. 78]
SEI Score:	Ada 4	C++ 4	

C was downgraded because it is essentially limited to syntactic (re)naming of predefined data types and data structures, with building-block capabilities within these limitations. Unlike Ada, neither C nor C++ allows restrictions to the range of a type. The C++ extensions to C in this category are the most significant enhancement. A class in C++ is a user-defined data type that provides data hiding, guaranteed initialization of data, implicit type conversion for user-defined types, dynamic typing, user-controlled memory management, and mechanisms for overloading operators [23, p.iii]. Ada has been downgraded slightly by the SEI for lacking a mechanism like classes that permits programming by specialization and extension.

4. Control structures -- "Extent to which the language provides an adequate set of the structured programming control structures, such as SEQUENCE, IFTHENELSE, DOWHILE, DOUNTIL, and CASE" [p. 22].

FAA Score:	Ada 4	C 5	[p. 78]
SEI Score:	Ada 4	C++ 4	

Ada was downgraded by the FAA report because of a lack of a DOUNTIL. The control structures of C++ are essentially the same as C. Neither C nor C++ allows loops to step through enumerated types.

5. Procedural abstraction -- "Extent to which the language provides for introduction and invocation of programmer-defined subroutines, procedures, or functions" [p. 22].

FAA Score:	Ada 4	C 3	[p. 78]
SEI Score:	Ada 4	C++ 4	

Ada received a higher score primarily for generic procedures and functions. C++ has no analogous feature in Release 2.1, but will have a feature called a template, which introduces the macro expansion side of generics in Release 3. C++ and C allow procedure parameters which were explicitly excluded as unsafe in the Steelman requirement for Ada. With inheritance, procedure parameters, and polymorphism, C++ solves some of the same problems that Ada generics solve.

6. Interface checking -- "Extent to which the language provides for and requires that compilers automatically perform consistency checking of interfaces between the invoking programs and the invoked ones, including assurance of both static (compile-time checkable) properties such as data type and dynamic (possibly requiring run-time tests) properties such as restricted range of values" [p. 22].

FAA Score:	Ada 5	C 3	[p. 79]
SEI Score:	Ada 5	C++ 4	

C provides checking within and among separately compiled programs using Lint, but interface checking is not enforced as it is in the Ada language system. In C++ consistency of separately compiled files is the responsibility of the programmer with the help of separately provided tools [23, p.104].

7. Input/output -- "Extent to which the language (and compiler and run-time support system) provides an appropriate variety of facilities for input and output of data to/from a program and for input/output device control" [p. 22].

FAA Score:	Ada 4	C 3	[p. 80]
SEI Score:	Ada 4	C++ 4	

Ada was deemed to have "extensive, but novel," capabilities (via pre-defined packages and representation specifications,) to interface to existing I/O capabilities within the context of the language. C was said to provide "nominal" I/O. C++ continues the C tradition of keeping I/O facilities in separate libraries that can be redefined. C++ overloads the operators "<<" and ">>" to mean "put to" and "get from" respectively. For example, if x is an integer with value 123, the statement

```
cerr << "x = " << x << "\n"
```

prints the string "x = 123" with a carriage return to the standard error output stream [23, p.226]. As it is defined in the Ada language reference manual (LRM), Ada's I/O is awkward to use for MIS, but specialized packages can be provided.

8. Segmentation/Modularization -- "Extent to which the language provides for partitioning of the program into comprehensible units" [p. 22].

FAA Score:	Ada 5	C 3	[p. 79]
SEI Score:	Ada 5	C++ 4	

Ada is rated as being superior to C in the FAA study because of its package concept for grouping related data and routines, and its distinction between specifications (the client-visible interface) and body (the implementation). The consistent syntactic organization of source text partitioned into declarative and executable statements is also noted for Ada. The C++ class provides functions similar to the Ada package, but in a somewhat different way. While C++ organizes around object definitions, Ada organizes around groups of programmer-defined objects and operations. Both C and C++ use header files for inter-module consistency, but rely on separate tools such as the make tool in UNIX for building consistent systems.

9. Parameterization -- "Extent to which it is possible to write fully parameterized programs (i.e., general purpose) programs" [p. 22].

FAA Score:	Ada 5	C 3	[p. 80]
SEI Score:	Ada 5	C++ 4	

The FAA report cites Ada for its generic facilities and C for being much more "low-level." C++ introduces much greater parameterization capability with classes, constructors, and destructors. Further parameterization will be introduced with templates in Release 3.

10. Encapsulation -- "Extent to which the language provides information hiding mechanisms and enforces access rights to data" [p. 22].

FAA Score:	Ada 5	C 3	[p. 80]
SEI Score:	Ada 5	C++ 4	

C is cited in the FAA study for some limited forms of encapsulation protection using its "internal" rather than its "external" declarations. Ada is cited for its excellent scoping rules and its limitation of access to data within a package. Ada is also cited for its "private" and "limited private" data types, which provide additional distinctions regarding access rights. In C++ the "public" label separates a class into two parts: public and private. The names in the private part can be used only by member functions. The public part constitutes the interface to objects of the class [23, p.136]. Protected names are available to class members and inherited members of inherited classes.

11. Concurrency abstraction (multi-tasking) -- "Extent to which the language and the run-time support system provide for logically or physically concurrent execution of multiple processes/tasks, for communication and synchronization between such processes/tasks, and for introduction and manipulation of programmer-defined processes/tasks" [p. 22].

FAA Score:	Ada 5	C 2	[p. 81]
SEI Score:	Ada 4	C++ 3	

Cited in the FAA report are the Ada tasking model and the Concurrent C enhancement to C. Ada includes tasking in the language and C++, like C, excludes it. Just as there is a concurrent implementation of C called Concurrent C, there is a Concurrent C++. Although tasking may have limited value in MIS applications and strong arguments can be made for keeping it outside the language, Ada ranks higher for having the features standard in the language.

12. Exception handling -- "Extent to which the language provides mechanisms for detection, processing, and initiation of 'unexpected' conditions or events, both language-defined and programmer-defined (such as PL/I's 'on units')" [p. 23].

FAA Score:	Ada 5	C 1	[p. 81]
SEI Score:	Ada 5	C++ 2	

Ada is cited for its exception mechanism, which handles language and programmer-defined conditions. C must be programmed manually to handle exceptional conditions. No exception facility exists in C++ Release 2.1, but one is planned for Release 3.

13. Macro capability -- "Extent to which the language or associated programming support environment tools provide a macro/template/model capability by which programmers can introduce and promote common usage of abbreviations/shorthands for existing facilities or extensions to the set of facilities in the base language" [p. 23].

FAA Score:	Ada 4	C 4	[p. 81]
SEI Score:	Ada 4	C++ 4	

The FAA report cites C as a powerful macro preprocessor tool. Ada is cited for meeting needs for which classical preprocessors have been used, as well as for providing generics, derived types, and the inline pragma. C++ has the same preprocessor capability as C. In addition, it has inheritance and polymorphism, which obviate some of the need for generics; in Release 3, C++ will have the template feature, which gives the macro capability of generics.

14. Library utility capability -- "Extent to which the language or its associated programming support environment tools make provision for utilization of a library of application oriented programs and data, and for augmenting such a library" [p. 23].

FAA Score:	Ada 5	C 3	[p. 82]
SEI Score:	Ada 5	C++ 3	

In C the library support is beyond the scope of the language and handled by separate tools. Considerable manual effort is required in C or C++ to achieve the same function that is provided by the Ada library support.

4.1.2. Capability summary

Although the Ada language has not changed since 1985, technology has advanced so that Ada received lower ratings in some categories of capability. C++ has provided marked improvements over C in some of the 14 categories of capability. It is primarily the introduction of the class concept in C++ that has facilitated data abstraction, information hiding, and encapsulation. In addition to the listed categories of capability, C++ and Ada both permit overloading of functions and operators, while C++ alone provides inheritance (the ability to derive a new user-defined type from an old one, making changes only as needed), and polymorphism (the ability to redefine a function of a base class in a derived class). The potential value of both inheritance and polymorphism is much more debatable than the other features. Their use and abuse is not yet well understood.

In terms of language capability, Ada had a significant advantage over C. The gap between Ada and C++ is judged to be narrower, and when the tool set available with C++ is considered, the capabilities are quite comparable. However, because of the incorporation of enforced consistency checking and enforced library capabilities within the language (rather than as separate tools), Ada retains a significant advantage in this broad category.

4.2. Efficiency

Efficiency refers to factors relevant to optimization of generated code and runtime utilization of resources (processor time, memory space, etc.). In this category, we are considering language implementations (compilers) in general, rather than the languages themselves. The natural consequence of making any technical assessment of available compilers is that there will be exceptions on either side of the norm.

In the efficiency category, there have been substantial changes since the 1985 study in Ada implementations. C++ implementations are in general newer than Ada implementations. It must be remembered that the C++ preprocessors (as opposed to native C++ compilers) generate C code for C implementations. Both the preprocessor and the C compiler are responsible (in part) for efficiency. This category has eight subcategories.

4.2.1. Efficiency scores

1. Optimization techniques -- "Extent of utilization within the compilers of various techniques to improve the efficiency of the generated object code, such as: (1) in-line expansion of subroutine calls, (2) loop optimization (i.e., movement of non-varying code out of loops, (3) common subexpression elimination, (4) register allocation" [p. 23].

FAA Score:	Ada 3	C 3	[p. 81]
SEI Score:	Ada 4	C++ 3	

With respect to Ada and C, the FAA report cites excellent potential for providing good levels of optimization, but this had yet to be demonstrated in 1985. There had been very little optimization of Ada at the time because compiler vendors were struggling to pass the validation tests to get their certificates. In the last several years in particular, significant attention has been paid to optimization. There is increasing evidence from a number of sources indicating that optimized Ada can be as fast or faster than optimized C.

Optimization in C compilers is well developed because the compilers are quite mature. C++ preprocessors generate C code, so there are two major levels of optimization possible, at both the preprocessor and compiler levels. C++ compilers will have to introduce new optimization techniques for the new features of C++. Because of the relative immaturity and fluidity of C++, the current level of optimization available in both the preprocessors and the compilers is thought to be lower than it is for Ada.

2. "Object code and load module size (i.e., storage required for generated object code and link-edited load modules)" [p. 23].

FAA Score:	Ada 2	C 4	[p. 83]
SEI Score:	Ada 3	C++ 4	

The 1985 FAA comment is that Ada's additional capabilities should produce significantly larger modules. In fact, early Ada compilers produced huge load modules for the shortest of programs. All the runtime code was loaded, whether needed or not. All procedures were loaded, whether called or not. Many I/O and library routines were loaded, whether needed or not. Load module sizes were on the order of several hundred thousand bytes. The situation today is much more reasonable. Ada compilers load only what is needed through intelligent linking. Runtime kernels can be smaller than 10K bytes. As is the case for time efficiency, C++ space efficiency is determined by the C compilers for the C core of the language and the preprocessor or the C++ compilers for the extensions. Again, because of the immaturity and fluidity of C++, the space optimization techniques are under current development and still improving. C++ has the edge here because there are fewer features in the language requiring runtime support.

3. Instruction path length -- "Number of machine instruction cycles required to execute a function" [p. 23].

FAA Score:	Ada 2	C 4	[p. 83]
SEI Score:	Ada 3	C++ 3	

In 1985, the FAA rated C high because of the compromise it achieves between simplicity and capability. Ada is cited for slower potential execution speed because of its rich set of features and its safety/consistency checking. The overhead of such a language can be considerable, but can be mitigated by optimization. As C++ adds new features to the base C language, it is expected that the problem will become as great for C++ as it is currently for Ada.

4. "Locality of reference for instructions and data (i.e., key drivers of working set size and paging rate)" [p. 24].

FAA Score:	Ada 3	C 4	[p. 83]
SEI Score:	Ada 3	C++ 3	

No reasons were provided in the FAA report for the 1985 scoring. No reasons can be advanced for believing that there would be any significant differences between Ada and C++ in this regard at the present.

5. "Data representation (i.e., storage required for various data types and data structures" [p. 24].

FAA Score:	Ada 5	C 3	[p. 83]
SEI Score:	Ada 4	C++ 4	

The FAA report cites the power of explicitly representing data in each of the candidate languages and thus permitting the language to represent the data most accurately. No reasons can be advanced for believing that there would be any significant differences between Ada and C++ in this regard at the present.

6. "Subroutine invocation overhead (i.e., out-of-line calling sequence and parameter passage)" [p. 24].

FAA Score:	Ada 2	C 3	[p. 84]
SEI Score:	Ada 3	C++ 3	

The FAA report rated Ada low because of the additional overhead involved in its parameterization, strong typing, and runtime checking. C++ introduces significantly more complexity to parameter passing as compared to C. There are special rules for passing vectors, a facility for passing unchecked arguments, and a facility for specifying default arguments [23, p.117]. Class member arguments can cause the creation of temporaries, invoking constructors and destructors for the class [10, p.172]. C++ has more modes of parameter passing than Ada (including pointers). While they may provide some efficiency, these modes also require greater degrees of understanding and discipline.

7. "Context switching overhead (i.e., multi-tasking)" [p. 24].

FAA Score:	Ada 1	C 4	[p. 84]
SEI Score:	Ada 3	C++ 4	

Ada was rated low in 1985 because of concerns regarding the efficiency of its high-level tasking model. C, like C++, should receive a rating only in the context of a typical operating system environment. The concerns about context switching (particularly the Ada rendezvous) were warranted in 1985. At the time, context switching was extremely costly (on the order of 1 millisecond for many processors). Today, the benchmark context switch times are about an order of magnitude faster. Context switching in Ada is still a concern, but today it is much less of a concern than it was in 1985.

8. "Overhead of establishing/saving/restoring the necessary run-time environment" [p. 24].

FAA Score:	Ada 2	C 4	[p. 84]
SEI Score:	Ada 3	C++ 3	

The FAA rationale for this scoring is that the state information for Ada is greater than the state information for C. No reasons can be advanced at the present for believing that there would be any significant differences between Ada and C++ in this regard.

4.2.2. Efficiency summary

In 1985, the FAA report rated C as significantly more efficient than Ada. Because of advances in Ada compiler technology in the past six years, the runtime efficiency of Ada compilers has improved significantly. It is also the case that significant complexities have been added to the C++ superset of C that have provided new challenges for C++ preprocessor or compiler developers. In the long term, there is no reason to expect significant differences in efficiency between Ada and C++ programs. C++ is given a slight advantage in this category because of less complexity and reliance on efficient C compilers.

4.3. Availability/Reliability

Availability/reliability refers to factors influencing the day-to-day safety of operational systems. This category is heavily influenced by the safety features of the language as well as by the maturity of the implementation technology. Just as for the efficiency category, there may be substantial changes for Ada implementations and C++ preprocessor/compiler implementations. There may also be substantial deviations from the norms for compilers. This category has four subcategories.

4.3.1. Availability/reliability scores

1. Correctness -- "Extent to which the language, compilers, and run-time support systems are free from design and program defects and satisfy their specifications" [p. 24].

FAA Score:	Ada 5	C 3	[p. 85]
SEI Score:	Ada 4	C++ 3	

C is cited by the FAA report as having "nominal" correctness. The Ada validation process is cited as ensuring compliance with its language specifications. The validation process for compliance testing was not as effective as thought in 1985, and the correctness of early Ada implementations was not uniformly high. Even compilers that passed validation had numerous latent bugs. Section 2.3 of this report gives a more complete analysis of the standardization and validation situation, but correctness largely depends upon maturity and today the Ada compilation technology as a whole is more mature and robust is than the C++ compilation technology.

2. Computational accuracy -- "Extent to which the accuracy or precision of numeric computations is guaranteed or may be controlled with the language, compilers, or runtime support systems" [p. 24].

FAA Score:	Ada 4	C 3	[p. 85]
SEI Score:	Ada 4	C++ 3	

The FAA report cites C as having "nominal" computational accuracy. Ada's model numbers are cited as imposing more stringent requirements for the integrity/accuracy/precision of numeric data. Ada is unchanged from 1985 in this regard. In addition, it should be noted that neither Ada nor C++ provides inherent support for decimal numbers and decimal arithmetic, a major requirement for MIS. However, one of the requirements for Ada 9X is to provide such support.

3. Compile-time safety/consistency checking -- "Extent to which automatic language-defined safety/consistency checking of the program is performed during compilation" [p. 18].

FAA Score:	Ada 4	C 3	[p. 85]
SEI Score:	Ada 4	C++ 3	

Ada is cited in the FAA report for requiring extensive data type checks and interface checks at compile time, even between separately compiled programs. C (via its associated Lint tool) is cited for performing these checks. The situation with Ada is unchanged since 1985. C++ provides somewhat more checking, but C++ largely maintains the programming style of C. Because of its late binding philosophy, it is in many cases impossible for C++ to do compile-time checking because the types associated with objects are not known until runtime (i.e., polymorphism). Ada is still stronger in this category.

4. Runtime safety/consistency checking -- "Extent to which automatic language-defined safety/consistency checking of the program is performed during execution" [p. 24].

FAA Score:	Ada 5	C 1	[p. 86]
SEI Score:	Ada 5	C++ 2	

Runtime checks are required (but can be suppressed) for Ada. The exception mechanism permits default error handling or user-defined error handling. C is cited in the FAA study for performing practically no runtime checks. The situation for Ada is unchanged from 1985. C++ has not significantly changed the situation from what it was with C, but there are commercial tools that provide runtime checking code. An exception mechanism is planned for Release 3 of C++.

4.3.2. Availability/reliability summary

In 1985, the FAA report rated Ada significantly higher than C in availability/reliability. The situation today is largely unchanged. C++, like C, is still a permissive language that allows more freedom to make unconscious mistakes. For example, Ada forces a conscious decision to violate the typing rules through its unchecked conversion facility. This freedom in C++ adds a measure of flexibility and possibly efficiency in certain cases, but also has the potential of introducing errors that may not be discovered until late in the development process. Those interviewed for this study who have used both languages extensively cite Ada as a safer, more "bullet-proof" language.

4.4. Maintainability/Extensibility

Maintainability/extensibility refers to factors influencing long-term viability of operational systems. This evaluation category incorporates the software engineering "ilities." While no programming language can "enforce" or even "encourage" the use of good programming practices, there are language features that facilitate or hinder good programming practices. Because Ada has not changed since 1985, its rating in this category is unchanged. The C++ features added to the C subset largely address software engineering issues. This category has six subcategories.

4.4.1. Maintainability/extensibility scores

1. Modularity/encapsulation -- "Extent to which programs written in the language can be partitioned into comprehensible units, and provision for information hiding along structural lines" [p. 24].

FAA Score:	Ada 5	C 3	[p. 86]
SEI Score:	Ada 5	C++ 4	

This subcategory is a repetition of the two similarly named categories in the capability section. Ada's packaging and scoping rules are cited in the FAA report. C is cited for "nominal" modularity/encapsulation features. The class feature of C++ provides a different way of addressing this problem than does the Ada package. Which capability is better is still unclear.

2. Readability/understandability -- "Extent to which the lexical form and syntax of the language helps convey information about a program and its behavior and makes it easier to read and understand" [p. 24].

FAA Score:	Ada 4	C 2	[p. 86]
SEI Score:	Ada 4	C++ 2	

The FAA report rates Ada higher than C for clarity of expression, but downgrades Ada somewhat for its extensive scope of features. C is rated low because of the terseness of its lexical form and syntax and the lack of distinction between its expressions and statements. Ada is unchanged in this regard, and C++ has the same style of expression as C. As Stroustrup points out [23, p.20], "C++ (like C) is both loved and hated for enabling such extremely terse expression-oriented coding." The philosophy of making the language closer to the machine is characterized by the assignment "x[i+3] = x[i+3]*4" which can be rewritten as "x[i+3]*=4". The reason for allowing such obscure text is explained by Stroustrup as follows [23, p.17]: In the latter case the expression "x[i+3]" needs to be evaluated only once instead of twice so that this gives "a pleasing degree of runtime efficiency without the need to resort to optimizing compilers." In other words, the language is made less readable so that the compiler implementers do not have to work as hard. Some of the readability problems of C++ can be attributed to historical and cultural differences rather than to the language itself. Certainly it is possible to write readable programs in C++ if sensible guidelines are written and enforced. Conversely, Ada programs can be very unreadable without any guidance. Nevertheless, Ada is still considered superior to C++ in this category.

3. Usability -- "Degree of effort required to learn, operate, prepare input for, and interpret output from the language, the compiler, or other programming support environment tools" [p.24].

FAA Score:	Ada 2	C 4	[p. 87]
SEI Score:	Ada 3	C++ 3	

The FAA report rates Ada low on usability because of the "greater knowledge and education" required in the operation of programs and their compilation. Ada's rating is also downgraded because of the "temporary deficiencies in compiler maturity and tool availability." C is easier to use, but this is tempered by its "low-level orientation" and its low readability. The complexity of Ada remains the same as it was in 1985, but the maturity of compiler technology has improved significantly so that many usability problems have been alleviated. C++ has increased significantly in complexity and now approaches Ada in the knowledge and education required to use it effectively. The low-level orientation and low readability are retained from C. There is little to distinguish the languages in this category if readability is separate from usability.

4. Reusability -- "Degree of effort required to write and organize programs in a way that makes their later reuse easier and more likely" [p.25].

FAA Score:	Ada 4	C 3	[p. 87]
SEI Score:	Ada 4	C++ 4	

Ada is said in the FAA report to be clearly superior in this category. This is attributed to the way the package concept can be used to facilitate a software component approach to reuse. C is given credit for "common data typing capabilities." Although there is substantial talk about reuse for both Ada and C++, much of it is speculation. Many people in the reuse community now believe that component reuse is not the answer, but that reuse must take place at the architecture level. The questions about reuse have yet to be answered, so that neither language can be said to have a significant advantage.

5. Transportability -- "Degree of effort required to transfer a program from one target system configuration (or host system configuration, i.e., programming support environment) to another" [p. 25].

FAA Score:	Ada 4	C 3	[p. 88]
SEI Score:	Ada 3	C++ 3	

In the FAA report Ada is cited for its objective of facilitating transportability by separation of logical and physical data representations and the ability of its syntax and semantics to "highlight" target hardware and implementation dependencies. C is rated "nominal" in transportability because of its simplicity and popularity. The portability of Ada was highly overrated in 1985. Portability remains difficult, particularly within the embedded environment for which Ada was designed. However, the portions of Ada that are not dependent upon the machine and implementation have been quite portable across many hardware and operating system platforms. Because of the tool support they require, C and C++ have been largely products of the UNIX environment; transportability across UNIX systems is relatively high.

6. Interoperability -- "Degree of effort required to couple one program/system with another" [p. 25].

FAA Score:	Ada 5	C 3	[p. 88]
SEI Score;	Ada 4	C++ 4	

The FAA study rated Ada high because of its objective of interoperability within embedded systems. C was rated "nominal" because of its relative simplicity. The meaning of this category is somewhat unclear, given its description and the reason for the rankings; the "coupling" described in the report will be interpreted as meaning the coupling of unlike programs or systems. Ada implementations have the pragma interface, which allows them to work with a variety of other languages including C and assembly language. Since this feature is implementation dependent, calls to C++ will eventually be available as well. Being relatively low-level, C and C++ can also be used to invoke programs in other languages. As Ada matures, many "bindings" are being defined to other programming systems, such as databases, windowing systems, and graphics systems. Similar bindings from C++ to these systems are available through C. At this time, neither language can be said to have a significant advantage in this category.

4.4.2. Maintainability/extensibility summary

In 1985, the FAA report found Ada to have a significant advantage over C in the category of maintainability/extensibility. Subsequent events would indicate that this evaluation was unduly generous to Ada at the time. Other than in the readability/understandability category, there are no significant differences, but that is a rather important and highly weighted category by most users of a language, particularly for large complex systems with long lifetimes. Ada has a small advantage in maintainability/extensibility.

4.5. Life-cycle Cost

Life-cycle cost refers to elements of budgetable cost associated with or affected by the implementation language. Because cost factors are very much influenced by maturity, the situation has changed for Ada since 1985 and the situation for C++ is quite different from what it was for C. Estimating cost factors is risky and can be highly unreliable when based on past history in extremely volatile technologies; consequently, these observations must be viewed with extreme caution. This category has ten subcategories.

4.5.1. Life-cycle cost scores

1. Compiler acquisition -- "Cost of acquiring compilers and run-time support systems" [p. 25].

FAA Score:	Ada 2	C 4	[p. 89]
SEI Score:	Ada 3	C++ 4	

The assessment in the FAA report is a reflection of (1) the number of commercially available compilers for Advanced Automation System (AAS) candidate architectures, (2) the number of committed, ongoing compiler development efforts, and (3) the overall number of groups and companies involved in compiler development for the languages. The number of companies sponsoring validated Ada compilers in early 1991 was 33 (down from a high of 52 in December of 1988), and the number of validated Ada compilers is 135 (down from 292 in December of 1989). An additional 43 Ada compilers have completed testing or are scheduled for testing. The declines may in some cases be due to companies going out of the Ada compiler business, but in other cases can be attributed to vendors who still support a compiler, but who have chosen not to renew its validation under a new validation suite. The Ada figures are inflated to some degree due to the use of Ada in embedded systems, where there are, in general, many cross compilers targeted to different microprocessors for a single base compiler. The situation with C++ is that it is supported on any UNIX system with the AT&T language system and the C compiler. A small number of companies produce native C++ compilers. In 1991 the availability of Ada and C++ compilers is high. Because of the involvement of AT&T in distributing the cfront preprocessor, and because of volume considerations, the per unit cost of C++ compilers can be expected to be lower than for Ada.

2. Other tool acquisition -- "Cost of acquiring other programming support environment tools" [p. 25].

FAA Score:	Ada 3	C 4	[p. 89]
SEI Score:	Ada 3	C++ 4	

In the 1985 FAA report, C was rated higher than Ada because of existing tools and tool development activities at the time. Ada was rated lower because a more integrated tool set was only a potential capability. Now, Rational produces a highly integrated tool set for Ada that has received high praise. Integrated tool sets are also available for C++ from companies such as Borland. C++ can be said to have a slight advantage in this category because of supply and demand factors.

3. Documentation -- "Cost of documentation for the language, compilers, run-time support systems, and other programming support environment tools" [p. 25].

FAA Score:	Ada 4	C 4	[p. 89]
SEI Score:	Ada 4	C++ 3	

C is cited for its maturity and support and Ada is cited for its "enormous thrust" and "standardization," which creates a common basis for documentation. It should be noted in 1991 that there has been only one definition of Ada and it will have remained a constant standard for at least 10 years by the time Ada 9X is introduced. The reference manual is the single definitive text that is used. C++ has had separate reference manuals for its separate versions, and will probably have more before becoming a standard. Both Ada and C++ must have volumes of clarifying documentation for detailed semantic interpretations. Ada may have a slight edge in this category because of its stability over time.

4. Training -- "Cost of language-specific, compiler-specific, etc. training for software developers/maintainers" [p. 25].

FAA Score:	Ada 1	C 4	[p. 89]
SEI Score:	Ada 3	C++ 4	

The FAA report gives C high marks because its simplicity relative to Ada and because of its established training material. Since 1985, many companies have been formed specifically to provide Ada training and many compiler vendors also provide training. C++ training is also readily available today. Both Ada and C++ require training that incorporates the principles of software engineering. C++ can be said to have a slight advantage in this category because there are more C++ training courses and, consequently, more competition.

5. Transition from design to code -- "Cost of the transition from design in the PDL/Ada design language to code in the candidate implementation language" [p. 25].

FAA Score:	Ada 5	C 3	[p. 90]
SEI Score:	Ada 4	C++ 4	

The FAA report cites the high-level nature of Ada in comparison to C for expressing design information. Although Ada has been used for software design, its effectiveness was perhaps overestimated in 1985. Whether C++ is being used extensively to express software design is unknown. There is nothing to support a significant rating difference between Ada and C++ in this category.

6. Compiler and other tool execution -- "Cost of 'running' the compilers and other programming support tool" [p. 25].

FAA Score:	Ada 1	C 3	[p. 90]
SEI Score;	Ada 3	C++ 4	

The ratings reflect the relative simplicity or complexity of the two languages. Ada is more difficult to compile than C because of its rich set of features. The expense of compiling Ada code stems from the number of different constructs that need to be handled by the compiler, thus increasing the compiler's size, the amount of information that needs to be kept by the compiler to check the legality of the source code, the potential complexity of some of the compile-time checks and actions (e.g., evaluating constant arithmetic expressions exactly at compile time using a rational arithmetic package), and the complexity of the semantics that must be supported at runtime (e.g., exception handling and tasking). Clearly, a language with fewer constructs, fewer required checks, and more primitive semantic constructs leads to a smaller compiler that is simpler to build. Ada will probably still be more difficult to translate to machine code than C++, but the difference is much less than that between Ada and C.

7. Compiler maintenance -- "Cost of maintaining the compilers and run-time support systems" [p. 25].

FAA Score:	Ada 5	C 3	[p. 91]
SEI Score:	Ada 4	C++ 2	

Due to the degree of committed backing for Ada and its compiler validation process (which should result in a lower compiler maintenance cost), the FAA ranked Ada superior to C. Again, this is largely a question of stability and maturity. The original FAA report was overly sanguine about the quality of the initial compiler product delivery, but the situation has improved significantly since 1985. Because C++ is a "living language," it will change at more frequent intervals than Ada. Ada will also change with Ada 9X, but at a more predictable and less frequent rate. Ada gets a substantially higher rating in this category.

8. Other tool maintenance -- "Cost of maintaining other programming support environment tools" [p. 25].

FAA Score:	Ada 4	C 3	[p. 91]
SEI Score:	Ada 4	C++ 4	

The FAA rationale for scoring in this category follows the same logic as for the previous category. Because the support tools for Ada are not controlled in the same manner as the language itself, it is hard to justify this argument. The support tools will be evolving at the same rate as the support tools for C++. Thus, no justification exists for different scores in this category.

9. Software development cost impact -- "Relative impact on software development costs (i.e., design, implementation, and test activities)" [p .25].

FAA Score:	Ada 4	C 3	[p. 91]
SEI Score:	Ada 4	C++ 4	

The FAA report cites C for "nominal" software development costs. Ada is cited for higher design costs, which are more than offset by lower testing and integration costs. While this was speculation in 1985, these observations have generally been borne out by data on real projects. Software development costs are generally slightly higher when Ada is used for the first time, but lower by the second or third project. This same progression could also be expected for C++. If we are starting from scratch in both languages (as will initially be the case in most MIS developments), there is little reason to believe that the costs would be significantly different.

10. Software maintenance cost impact -- "Relative impact on software maintenance cost (i.e., debugging, enhancement, and extension activities)" [p. 25].

FAA Score:	Ada 4	C 3	[p. 91]
SEI Score:	Ada 4	C++ 3	

The FAA study cites Ada for its "inherent" reuse opportunities and the potential of reuse for reducing general software maintenance effort. Whether or not it is due to the reuse opportunities, the experience to date with maintenance, limited though it may be, has been good. Primarily for the reasons mentioned in the categories of readability/understandability and availability/reliability, Ada is rated slightly higher than C++ in this category.

4.5.2. Life-cycle cost summary

In 1985, the FAA report found Ada to be significantly lower than C in life-cycle cost. Subsequent events and data would seem to support that original conclusion. With respect to C++, there is little data to support any conclusions. A concern is that C++ will be harder to maintain for large systems because it will continue to evolve, and every step in the evolution will cause some small unexpected inconsistencies or incompatibilities. On the other hand, if commonality can be exploited through the use of classes, C++ may require less code to maintain. Also, Ada will evolve into Ada 9X in a few years. On balance, Ada has a small advantage over C++ in life-cycle cost.

4.6. Risk

Risk refers to the areas of uncertainty or concern associated with or affected by the implementation language. Risk must be associated with any requirement that a candidate does not currently meet, but must be made to meet. Risk changes significantly over time. Factors that were risky in 1985 may no longer be risky. New risks may be identified over time. The risk factors for C++ may be very different from the risk factors of C. Again, the overall aspect of maturity plays a major role. This category has six subcategories. Note that high scores correspond to low risk.

4.6.1. Risk scores

1. Functional risk -- "Uncertainty/concern regarding the technical feasibility of meeting the system's functional requirements with the candidate language, its compilers, and its run-time support systems" [p .26].

FAA Score:	Ada 3	C 2	[p. 92]
SEI Score:	Ada 3	C++ 3	

The FAA report scores refer to the real-time embedded command and control systems—in the FAA application domain and also the domain for which the Ada requirements were written. For MIS, both Ada and C++ have inherent risks because neither was designed for large information systems: Ada was designed for embedded systems and C++ was designed for systems programming. Neither embedded systems nor systems programming is similar to information systems. MIS requires decimal arithmetic, transaction processing, and good I/O facilities. Therefore, with respect to MIS, neither Ada nor C++ can be ranked very high in this category.

2. Performance risk -- "Uncertainty/concern regarding the technical feasibility of meeting the system's performance requirements (e.g., availability limits, response time limits) with the candidate language, its compilers, and its run-time support systems" [p. 26].

FAA Score:	Ada 1	C 3	[p. 93]
SEI Score:	Ada 3	C++ 3	

C is cited as being an "average, but dependable" performer, and Ada is cited as still being weak in the performance area. Since 1985 this risk factor has been largely mitigated for Ada. There are several benchmarking studies demonstrating that Ada can produce code that is as fast or faster than FORTRAN or C. But there are also language rules in Ada (Chapter 11 of the reference manual) that preclude some forms of optimization. For C++ this category is an acceptably small risk factor. C++ is probably not as far along as Ada in optimization of its newer features, but the difference is not significant.

3. Development schedule/cost risk -- "Uncertainty/concern regarding the software engineering job of designing, implementing, and testing systems with the candidate language, its compilers, and its run-time support systems" [p. 26].

FAA Score:	Ada 1	C 2	[p. 93]
SEI Score:	Ada 4	C++ 2	

The 1985 FAA report downgrades Ada as being a new and untried language that would have a significant negative impact on the development costs and schedules. The uncertainties are cited as "enormous." The report cites the inexperience of IBM with C for the software development effort of the type and magnitude required by the FAA for the AAS. For Ada this risk has been greatly reduced since 1985. Many large systems have been successfully implemented in Ada. The Army's STANFINS-R development in Ada is a largely successful demonstration that million-line systems can be written in Ada for MIS applications. No evidence has been found that C++ has been demonstrated on large MIS applications. Those people interviewed for this study know of no use of C++ on million-line systems outside of AT&T. In 1991, use of C++ for MIS entails much greater risk than use of Ada for the same purpose.

4. Transition schedule/cost risk -- "Uncertainty/concern, in terms of systems support, regarding the transition from existing systems implemented in certain other languages to new systems implemented in the candidate language" [p. 26].

FAA Score:	Ada 4	C 3	[p. 93]
SEI Score:	Ada 3	C++ 2	

Again, the application domain and the base language for the FAA report differs from those considered in this report. The primary base language in the MIS case is COBOL. The transition from COBOL to Ada has been demonstrated by STANFINS-R, but there has been no transition from COBOL to C++ on any scale according to those consulted for this study. While the transition from COBOL to either Ada or C++ may be costly, the risk must be considered somewhat higher for C++ than for Ada.

5. Maintenance schedule/cost risk -- "Uncertainty/concern regarding the software engineering job of debugging, enhancing, and extending systems implemented with the candidate language, its compilers, and its run-time support systems" [p. 26].

FAA Score:	Ada 4	C 3	[p. 94]
SEI Score;	Ada 4	C++ 3	

The 1985 FAA report cites Ada's support for software engineering principles and the concept of software components. Since C++ was an evolution of C designed to support the same engineering principles, there is reason to believe that C++ will be much better than C in this regard. Because of Ada's maturity and stability, it must be considered lower in risk in this category.

6. Long-term viability risk -- "Uncertainty/concern regarding whether the candidate language will retain its technical vitality, whether the compilers and run-time support systems will continue to be supported, etc., throughout the expected life-cycle of the systems (i.e., through 2010)" [p. 26].

FAA Score:	Ada 5	C 4	[p. 94]
SEI Score:	Ada 5	C++ 5	

The 1985 FAA report cites the technical vitality of Ada as a new language. Today, C++ is the new language and could be so cited. Unless the government reverses its long-standing support of Ada, it will be viable for the long term, particularly if the changes envisioned in Ada 9X are viewed as an improvement. C++ will continue to be viable as long as UNIX is viable and will inherit the considerable software base that has been written in C. Both languages appear to be quite viable in the long term.

4.6.2. Risk summary

In 1985, adopting Ada for a large project for which it was not mandated was indeed a risky proposition. Today, the risk of adopting Ada, even for MIS where it has had less exposure, is relatively low. This is because today Ada is stable and relatively mature. It has been adopted by numerous organizations outside government for numerous applications (see Appendix B). C++ is newer than Ada and carries some risks that Ada has already mitigated through time and usage. C++ is unproven in large systems and unproven in MIS applications. C++, therefore, is a higher risk language today than Ada.

4.7. Summary of Deviations

The original FAA weighted scores for the 6 categories in 1985 were as follows:

Category	Max. score	Ada	C
Capability	16.7	16.1	9.6
Efficiency	16.4	8.1	11.8
Availability/Reliability	22.6	21.5	11.6
Maintainability/Extensibility	17.4	14.0	10.2
Life-cycle cost	11.3	8.2	7.4
Risk	15.6	8.8	8.9
Total	100.0	76.6	59.5

The FAA weights for the 6 broad categories and the 48 individual criteria were reviewed for this report in light of the differences between the embedded real-time systems application domain and the MIS application domain. We found that most of the categories were at such a high level that modification of the weights was not necessary. Any minor changes were deemed to be insignificant to the overall results and would cloud direct comparison with the original rankings.

The SEI weighted scores for the 6 criteria categories in 1991 are as follows:

Category	Max. score	Ada	C
Capability	16.7	15.3	11.3
Efficiency	16.4	10.7	10.9
Availability/Reliability	22.6	19.1	12.6
Maintainability/Extensibility	17.4	13.6	11.4
Life-cycle cost	11.3	8.4	8.0
Risk	15.6	11.7	9.8
Total	100.0	78.8	63.9

In both sets of scores, the maximum column reflects the relative weightings received by the 6 criteria categories. In 1985 Ada was considered more capable than C. Today, there is a smaller but still significant difference between Ada and C++ in this category. In 1985, based on language definition, Ada implementations were expected to be less efficient than C implementations. Today, there is only a slight advantage for C++. In 1985, Ada was ranked significantly higher in the availability/reliability category. Today, there is still reason to believe that Ada ranks significantly higher than C++. In 1985, Ada was ranked somewhat higher than C in maintainability/extensibility. Today the relative difference is narrower, but still applies for Ada and C++. In 1985, Ada ranked somewhat better in life-cycle cost than did C. Again, the difference is slightly narrower, but the same situation still applies. Finally, in 1985 Ada was found to have roughly the same risks as C for the FAA application domain. Today, we find that the risks are somewhat higher for C++ than for Ada in the MIS domain.

5. Experience to Date with Ada and C/C++

Another means of evaluating whether Ada or C++ is appropriate for the MIS environment is to examine the experience that exists today. This section highlights some direct experience with both Ada and C++. Except for the last section, the descriptions are based on interviews. Where attributions are made, the contributors were given the opportunity to review the material for accuracy.

5.1. Experience at Xerox

The Digital Systems Department at Xerox is one of the few organizations that has made a direct comparison of C++ and Ada. The application domain considered in the study was the "large real-time embedded systems software" domain rather than the MIS domain. The department is responsible for most of the control software embedded in Xerox copier products.

The Xerox study was conducted in late 1989 by a 12-person task force. A requirements team was first formed to develop two sets of requirements: one for language issues and another for tool set/implementation issues. The first set of requirements dealt with broad language issues such as the support for abstraction, the stability of the virtual machine, and standards. The second dealt with implementation issues such as availability of compilers, vendor support, and distributed development environments. Based on these requirements, a list of potential candidate languages was narrowed down to four: Mesa (including the Xerox PARC extension called Cedar), Sequel (a Xerox real-time proprietary language), C++, and Ada. The two proprietary languages were included because of the large existing base of software written in those two languages.

Four teams then evaluated the four candidates and presented their findings in a common format, using weighted scoring and tables of results, to the group as a whole. Then a Delphi technique similar to the FAA methodology was used to refine the evaluations. The results of the study were that, from both a language and an implementation point of view, Ada was slightly superior to the other three for the application domain of the Digital Systems Department. From a financial point of view, it was projected that over a 12-15 year period Ada was far superior to any of the three other candidate languages. Ada was predicted to be more costly initially, with a break-even point occurring after about 4 years. C++ was predicted to have higher maintenance costs due to lack of standardization when compared to Ada. The assumptions used to derive these projections were documented and circulated in Xerox, and were not challenged.

After a year of prototype development with Ada, Xerox views its experiences with Ada positively. A strong grassroots movement toward Ada within Digital Systems is attributed to both the language and the Rational Ada development environment. However, use of Ada at Xerox is still a controversial topic at the corporate level. There are factions who believe that Ada is not commercially viable and may be a liability in terms of interfacing with other systems in Xerox and in the wider marketplace. Emil Gottwald, a department head in Digital

Systems, believes that the real solution is a combination of Ada with ANSI C for external interfaces since C++ is still too unstable to be a viable alternative to Ada. He has found no evidence that any major corporation other than AT&T has selected C++ as its language of choice for large embedded systems applications.

5.2. Experience at Rational

Rational is a company whose birth and development closely parallels the birth and development of Ada. Rational developed one of the first Ada compilers. Their business strategy has been to provide development environments for large, complex Ada systems. They have developed more than one and a half million lines of Ada code for their Ada tool set.

More recently, to extend their business outside the defense/aerospace community, Rational has expanded into software engineering management consulting. In this new business endeavor they have come into contact with C, C++, and Smalltalk, a very early and pure object-oriented language developed at Xerox Parc. They have found that C is firmly entrenched in certain markets such as the workstation and telecommunication markets, and that Smalltalk is being used for prototyping in MIS applications.

In those businesses where C is firmly entrenched, they find that the language of choice for new development is C++. Because C++ is a powerful superset of C, existing C code and newly developed C++ code can be easily linked together. This enables existing C systems to be redesigned in the object-oriented style and C++ in an iterative, evolutionary way with lower cost and risk than that of complete redesigns.

Among the points made in comparing C++ and Ada based on their experience at Rational, Brett Bachman (Vice President and General Manager of the Object-Oriented Products Group) mentions:

- The inheritance features of C++ are particularly valuable for certain applications such as graphics and user interfaces. In graphics applications, for example, you would like to use polymorphism so that you can use the same operation to move a figure independent of where it appears in the type hierarchy (e.g., whether it is a quadrilateral, a rectangle, or a square).
- The choice between C++ and Ada will typically not be made on purely technical grounds. Rather it will depend on structural or infrastructure requirements. The choice should be influenced by the current software base of a project and the current development tools available to the team members. For example, C code can be more easily merged with newly developed C++ code. If the current software base of a project is C or C++, C++ is probably a better choice for new development. Or if there is COTS software available for reuse, the language this software is written in may influence the choice of language for system development.
- The C++ language is about 5 years behind Ada in terms of the maturity of its tool set, but is expected to mature faster than Ada because the compilation technology required is less complex than Ada. For example, C++ compilers

and runtime systems do not need to support constraint checking, generics, tasking, and so forth. Furthermore, there is significant commercial investment being made in C++ tool development because of the market demand for these products.

- For projects starting from scratch with no structural or infrastructure requirements, Ada would be the language of choice because it is better engineered and the tools available are more mature and robust. C++ is a hybrid language based on C; it was not engineered from the beginning with the goal of supporting modern software engineering approaches, as Ada was. The Ada compilers and environments are available today supporting a wide range of computer hardware architectures. Although C++ compilers are available today, no C++ environments are yet available.
- Rational is not aware of C++ applications larger than a million lines, while several customers are writing Ada applications of well over a million lines.
- One area in which C++ may have a distinct advantage is in personal computer applications. Cooperative processing involving access to mainframes and minicomputers using Macintoshes and IBM PCs may require bindings to products such as Windows 3.0. Such bindings will be slow to develop in Ada.

Rational is following market forces in pursuing business in the object-oriented design and programming arena; they build products and deliver services only when their customers are willing to place orders for them. But Rational recognizes that today Ada is still a better choice for building large complex systems for many applications.

5.3. Experience at Cadre

Cadre Technologies, Inc., provides computer-aided software engineering (CASE) tools to software developers. They have considerable experience in writing and maintaining code in C, C++, and Ada. They have developed over 1 million source lines of code (SLOC), most of which has been delivered in products. According to Project Manager Fred Wild, Cadre has developed "several hundred thousand" lines in C++ and "several hundred thousand" lines in Ada, with the remainder being written in C. Of their engineering staff, approximately one-half program in C, one-quarter program in C++, and one-quarter program in Ada. There is a small crossover group who program in both C++ and Ada. The version of C++ being used at present is Release 1.2 with conversion to Release 2.1 planned soon.

Cadre plans to continue to use both C++ and Ada. They are primarily being driven by the expressed demands of their customers rather than by technological forces. They think that both C++ and Ada will be popular state-of-the-practice languages in the 1990s. They are also driven by a need to support their products on a wide variety of different hardware/software platforms. They consider themselves pioneers in the C++ world, but expect a widespread migration from C to C++.

An early adopter for C++, Cadre is one of the few organizations that has thus far reported on experience with developing large systems in C++. In the proceedings of TRI-Ada '90, Wild [26] presents some cautions about maintaining code written in C++. In particular, he cautions that inheritance can cause serious problems in maintenance. Poor class structuring guidelines can easily yield class systems that are difficult both to use and maintain. He recommends that there be strict guidelines for class design, tools for understanding the structure of classes, and guidelines for code inspections, none of which exists in mature forms today.

The design of class systems is a new discipline that needs further maturation. It is a rare system whose solution space consists of a deeply layered hierarchy of classes. Most classes should be thought of as one-of-a-kind rather than nested in a highly abstract tree of classes. For example, it makes little sense to incorporate a list and a symbol table into a class called a data structure.

Among the points Wild makes in comparing C++ and Ada based on Cadre experience are:

- Ada is more "bullet proof" than C++ and does more checking at both compile time and runtime. There is more opportunity to get into trouble in C++, especially for neophyte programmers. As examples, he mentions the superiority of Ada's "with" statement over C++'s lexical inclusion, Ada's automatic library support over C++'s manual configuration management, Ada's type-safe generics over C++ macros, as well as Ada's runtime constraint checking and formal tasking model.
- Both Ada and C++ do a reasonably good job of providing support for software engineering principles.
- C++ code tends to be denser, with more function points per line of code. Use of the inheritance features increases this code density.
- C++ has more robust and less expensive compilers and tools. [This opinion differs from the one expressed by Rational and reflects experience with a different compilation system.]
- Some applications are more suited to C++ (those requiring inheritance mechanisms and for which a class system is obvious), while others are less suited to C++ (those requiring specific structures that cannot be easily generalized to other structures).
- Because of implicit activities associated with pointers and classes, there is a tendency in C++ to lose memory in long-running applications because programmers neglect to free space for objects in a variety of situations. The impact of this is latent runtime bugs that may not manifest themselves until a long-running application has exhausted its virtual memory.
- From both a language theory point of view and a software engineering point of view, Ada is probably better than C++, but Eiffel is probably better than either. Specifically, Eiffel includes mechanisms to ensure the integrity of a class by introducing preconditions, postconditions, and invariants. This protects the class from semantic errors that can be introduced by inappropriate overriding of operations by subclasses.

In summary, both Ada and C++ are deemed to be significant improvements over C. Cadre will continue to be driven by market forces and tool support rather than by the subtle differences in language design. Cadre finds that it is the adherence to software engineering principles, not a particular language, that makes a project succeed or fail.

5.4. Experience at NASA

Very recently (early 1991), NASA studied the question of what language to use on two major programs requiring the development of several million lines of software. The systems are the Space Station Control Center (SSCC) and the Space Station Training Facility (SSTF). In addition to their internal language studies, NASA contracted for two external studies, one by Mitre [21] and one by the Southwest Research Institute (SwRI) [2]. The alternatives in this study were Ada and C, but C++ was also considered in the studies as the natural successor to C.

The contractors were asked to consider ten specific questions, which included concerns of reuse of existing C code, productivity, availability of resources (people and tools), risks, COTS, and other more application-oriented questions. The studies were based on the experience with both languages at Mitre and SwRI, on data collected from other contractors with experience in both languages, and on other sources such as Don Riefer's cost database. Both of the contractors, as well as NASA, have concluded that for the large NASA systems, the language of choice is Ada rather than C.

One of the reasons that this question was raised at NASA is that a considerable amount of code exists in C for the Space Shuttle that might be of use in the Space Station. The potential for reusing this code at lower cost was at issue.

Among the points raised by both of the independent studies are the following:

- Reuse of existing C code was not a sufficiently compelling argument to recommend development of new code in C. However, the mixing of C and Ada code was deemed to be an acceptable alternative where reuse was possible or where C was particularly appropriate.
- Productivity gains cited for C over Ada were largely discounted. Early projects may cost as much as 10-20% more for Ada, but by the third and fourth projects the advantage switches to Ada. The larger the project, the greater are the life-cycle productivity gains attributable to Ada.
- The availability of resources (people and tools) over the long term (ten to twenty years) was not considered significant. Both studies foresee widespread support of both languages and the supply of programmers will meet the demand for software. They cite instances of contractors bidding Ada even though it is not a requirement.
- With regard to risk, C carries a higher and less manageable risk factor for development of large systems. Both studies gave Ada higher marks for software engineering factors such as portability, maintainability, and reliability.
- Both studies expected more COTS software to be developed in C than in Ada. Both point out, however, that the language in which COTS software is written is irrelevant if there is a standardized interface. They point to interfaces to POSIX, SQL, X11R4, MOTIF, and XView. They believe more COTS interfaces will be available for C, but that an Ada pragma interface is usually available as a substitute when specific bindings are not available.

Both studies recommended the use of Ada rather than C for new program development of the large, complex, long life-cycle NASA applications. They both allow for the possibility of using C in subsystems under certain limited circumstances. These include C code that can be reused with very minor revisions, relatively small subsystems requiring specialized interfaces or expertise, or relatively small programs that must be interfaced to software without Ada bindings.

Among the observations made about C++ by one or both of the studies are the following:

- Many of the features being added to C++ are features that were the basis for the design of Ada. These include object-oriented data encapsulation, abstract data types, function inlining, and compile-time consistency checking.
- The rapid movement toward object-oriented programming (OOP) has spurred the popularity of C++ and the studies expect the 1990's to be dominated by OOP and languages such as Ada and C++.
- C++ has shown "phenomenal" growth over the last two years and may surpass C in the next five years for development of COTS products. For large, complex, long life-cycle applications, C++ will be a better option, in all likelihood, than C.
- While better than C for supporting software engineering, C++ tools and standards are not as mature as those for Ada and there are few C++ programmers available.
- C++ programs that make heavy use of inheritance may cause additional problems with maintenance.

The studies done for NASA are the most recent and most comprehensive regarding C and Ada. While they are anecdotal rather than scientific studies, they do provide a great deal of corroborative evidence of the limitations of C.

Although NASA does intend to use C++ in a few limited places as prototypes for focused and small domains, it is concerned about the lack of standardization of C++ and the complex inheritance networks that can evolve in large systems.

5.5. Experience at ObjectWare

ObjectWare, Inc., has recently completed a conversion to C++ of a library of Grady Booch's software components [3] written in Ada. The Ada version of the components consisted of over 100,000 lines of Ada in 500 packages with approximately 12 members per package. The resulting C++ code contained about 20,000 lines of C++ with 380 classes with 8-9 members per class. This reduction in size was enabled by the highly regular structure of the Ada library and the mechanisms of C++ that permitted the exploitation of the commonality.

In particular, three capabilities permitted the reduction of the size of the code:

1. Inheritance in C++ permitted the construction of new components by deriving them from other components. For example, the concurrent versions of the data abstractions could be derived from the sequential versions of the analogous data abstractions.

2. The use of a feature called friends in C++ for the iterators in Ada avoided the need to duplicate each data abstraction in iterator and non-iterator supporting forms.
3. Constructors and destructors were used to avoid the manual error checking that was required in the Ada version due to exceptions.

One should be cautioned, however, that an 80% reduction in the number of lines of code does not necessarily translate to an 80% reduction in development effort. Much of the reduction described above is due to the textual repetition of code in Ada that is not required in C++. The potential for reducing the size of the original Ada code was not within the scope of the effort.

Among the points made by Mike Vilot, president of ObjectWare, about the comparison of Ada and C++, are the following:

- Language choice is heavily dominated by non-technical constraints, including the inventory of existing code, the tool sets available, the enhancements required over time, and political and economic factors.
- Inheritance is the "go to" of the 1990's. In other words, inheritance introduces a level of indirection, as do "go to's." Reasoning about a program involves understanding more than just the local source text. Ada has similar issues with generics and overloading.
- To help with the indirection problem described above, good programming environments with tools to navigate through the inheritance structure can be useful.
- In using several C++ language implementations, not as much divergence was found with the translators as was found with the provided iostreams library.

In general, Vilot believes that Ada may be a better choice for replacing COBOL than C is. Ada may be a reasonable replacement for C, but for the most part he sees no interest in doing so—C users are primarily moving to C++. He believes that C++ adds to C the same sort of better support for software engineering that Ada adds to Pascal. He feels that a COBOL programmer will face more of a learning curve when changing to a C-based language than to Ada. Also, the only successful large COBOL project that he has heard about is the Army STANFINS-R effort, which is a financial system. He is unaware of large projects that have converted from COBOL to C or C++. He also points out that there are efforts under way to enhance COBOL for an object-oriented style of programming.

5.6. Experience at the FAA

In May 1991, the FAA held a conference on their experiences with Ada on their Advanced Automation System [13]. Full-scale code production began at the start of their acquisition phase in November 1989. To date, they have developed over 600K SLOC in a planned development of 2 million SLOC. The target environments are the RISC System/6000 with the AIX operating system for common console processors and the S/370 with the MVS/XA operating system for their central processors.

Experience to date has been positive and confirms many of the findings of their earlier trade study. Nevertheless, they are quick to point out that the language is not the major issue. Rather, it is the software process and software engineering discipline that have accompanied the introduction of Ada. Among the lessons presented at the conference were that more effort and resources are spent on design than on coding, error rates are lower for Ada code than for non-Ada code, and that code must be developed specifically for reuse and is necessarily more costly as a result. Current assessments are that development productivity gains over the long term will be in the 160 to 180% range.

5.7. Experience Using Ada for MIS Applications

Ada is being used more and more for MIS applications both within and outside of government. Mike Feldman, a professor at George Washington University, keeps a list of non-government Ada applications. (This list is reproduced in Appendix B.) It includes a number of MIS applications, notably the Reuters transaction processing system for financial information, the Genesis Software bill paying system, and Japan's National Telephone and Telegraph's (NTT) database management system. These are all very large systems, from several hundred thousand lines to two million lines in the case of NTT.

The Genesis Software, Inc., (GSI) case is of particular interest, because they overcame a number of interoperability problems that are usually raised when Ada is proposed for MIS. The Prompt PayMaster (PPM) application is a quarter-million line system that was developed in eight months by a team of eight programmers [8]. The system was developed for a Wang VS computer using a Wang/Alsys Ada compiler. Wang has an Ada environment with a series of application programming interfaces (APIs) that permit access to functions in the platform. Two APIs that were used were Image, which accesses Wang's imaging technology (Wang Integrated Image System or WIIS), and XDMS, which accesses a keyed access method similar to Virtual Sequential Access Method (VSAM) on an IBM platform. Genesis is working on several other APIs for access to Wang's voice product, communications interfaces, relational database products, office automation products, and local area networks.

Two government Ada MIS systems of note are STANFINS-R, an Army financial accounting system redesigned from COBOL, and a redesign and enhancement of an Army personnel system from FORTRAN. STANFINS-R includes general ledger, accounts receivable, and cost accounting facilities. It consists of 500 programs and 2,000,000 lines of Ada developed by the Computer Sciences Corporation (CSC). Of 100 programmers, half were COBOL programmers and the remainder had either C or Ada experience. STANFINS-R was started in 1987 and completed in 1990. According to Rational, which provided consulting support, the system labor cost was half of what was expected due to productivity improvements. Productivity was roughly double what was expected. A key objective of this system was to increase productivity through automatic generation of application code from specifications. In fact, approximately half the code was generated in this way. [1] Ken Fussichen, Application Manager for SFANFINS-R at CSC, has found that while the technical problems of decimal arithmetic and interfacing to databases have been solved for MIS in an acceptable, but inelegant, manner, the social and cultural problems have not been adequately addressed. [15]

The Army personnel management system [9] is a subsystem of the Keystone program called MRP for MOS (Military Occupational Specialty) Readiness Priority. It is a quarter million line system with 2,000 compilation units and is much larger than the original FORTRAN system because of enhanced functions. It was developed by System Automation Corporation. Some interfaces were required to COTS software, such as the VSAM, and they were written in assembler.

The difficulty of interfacing to windows systems, particularly X-Windows, is often presented as an impediment to using Ada for MIS and other applications, and often provides an argument in favor of using C/C++. Two solutions to this problem were discussed at a Commercial Ada Users Working Group (CAUWG) in August 1990 [7]. Two issues of interfacing with the C code are callbacks and passing of aggregate data to C functions. Rational has taken the approach of avoiding the C interface problems by re-implementing the Xlib library that is written in C. SAIC, on the other hand, has produced an Ada binding to the latest release of Xlib (Version 11, Release 4). Ada 9X may provide some help by sponsoring a 9X binding to Xlib and Xt Intrinsics, the "runtime executive" for X.

Another company that has solved various interoperability problems is Wells Fargo Investment Advisors (WFNIA). In March 1991 they delivered a 100,000 line Ada/C/SQL investment management program for the DEC/VAX architecture [20]. This application required interfaces to DECWindows (the DEC version of X-Windows), an SQL database, a decision support system, and a spreadsheet. They found that Ada's strong typing mixed poorly with Generalized User Interfaces (GIUs) and databases and that Ada lacked support for business-oriented mathematics.

6. Conclusions

Based on the comparative analysis of Ada and C++ using the FAA evaluation criteria, Ada received a slightly higher rating in 1991 than it did in 1985. C++ received a slightly higher rating than C did in 1985, and its rating would have been higher still if not mitigated by concerns about stability and maturity. Ada will face similar maturity and stability concerns with the introduction of Ada 9X. Using these criteria, Ada was rated higher than C++. Based on these criteria alone, Ada would still be the language of choice for MIS even in the absence of a mandate to use it.

Based on the interviews with those who are familiar with both languages, there was a clear preference for using Ada for large complex systems with long lifetimes. These people cited Ada's early error detection, maintainability, and programming safety. On behalf of C++, they cited its growing popularity, its ability to support reuse through class libraries, and its ease of interface to a large body of software written in C.

Despite these study results, there is no clear answer to the question of whether the Ada programming language is better than the C++ programming language. Both are languages of the 1980's and improvements will be made to each in the 1990's. Both require robust compilers and supporting tool sets for effective usage. Both provide the capability for writing maintainable programs that are readable, efficient, portable, and reusable. To be used effectively to build large application systems, both languages require training and experience in the principles of software engineering.

There is, however, a clear answer to the question of whether sufficient justification can be provided for a waiver to use C++ instead of Ada for new development of MIS. The answer is that it is not possible at this time to make a credible case for "more cost effective" systems with C++ than with Ada. In terms of life-cycle cost, it will be some time before data exists to justify a cost savings for C++. Today, there is limited data available for Ada, and almost none for C++.

Because it is difficult or impossible to show greater life-cycle cost effectiveness for C++, waiver requests will rely on other reasons to try to justify lower costs. Among the predominant reasons will be: (1) greater popularity and hence more trained programmers, (2) greater capability, particularly in the areas of object-oriented programming and reuse, and (3) more interoperability with existing COTS software.

With respect to the growing popularity of C++ and the availability of programmers, it must be pointed out that the majority of MIS systems are written in COBOL, not in C or other higher level languages. Culturally, COBOL is likely to be closer to Ada than it is to C++ because of style of expression (statement orientation rather than expression orientation and verbose rather than terse text). Thus, if the majority of programmers come from within the COBOL community rather than from outside the COBOL community, Ada may be the better choice despite greater popularity of C++ in the wider community.

With respect to greater capability, the proponents of C++ will emphasize its object-oriented features, inheritance, and polymorphism in particular, and the prospects for reuse of "class" hierarchies. There is no accepted definition of what constitutes an object-oriented programming language. Cardelli and Wegner [6] distinguish languages that do not support inheritance as object-based rather than object-oriented. In his recent book on object-oriented design Booch [4] places both Ada and C++ in a class of programming languages that "best support the object-oriented decomposition of software."

With respect to COTS software, there may indeed be more COTS written in C++ than Ada in the future. But current policy does not preclude either the use of this COTS software or the use of Ada to interface with this software. Standards are either currently available or will soon be available to provide bindings (interfaces) from Ada to major COTS software products. Production quality implementations of these interfaces may be lagging for Ada, but workarounds exist in most cases today. If the need arises, there is no reason that Ada cannot be interfaced to C or C++ as an interim solution.

The most compelling reason not to grant waivers to use C++ for new developments is the lack of a stable standard. There is currently no C++ standard and there will be no standard for at least several years under the best of circumstances. The AT&T de facto standard will most likely evolve quite substantially over time. Ada, on the other hand, will be strictly controlled and will have a single conformance test suite which gives a greater degree of confidence in its uniformity and continuity over time. The transition to Ada 9X may negatively affect the language's maturity and stability, but there are efforts underway to minimize that impact.

Acknowledgements

The author wishes to acknowledge the following members of the SEI technical staff for reviewing portions of drafts of this paper or for providing leads to relevant technical information: Judy Bamberger, Mario Barbacci, Dan Berry, Dan Burton, Patrick Donohoe, Larry Druffel, Peter Feiler, John Foreman, John Goodenough, Marc Graham, Harvey Hallman, Bob Kirkpatrick, Mike Rissman, and Dan Roy. Dan Klein and Reed Little were particularly generous with their time in explaining the subtle and not so subtle aspects of C and C++ and their associated tool sets. Col. Richard R. Gross provided encouragement and support throughout the effort by reading and commenting on the drafts and asking the difficult questions. Frost McLaughlin, of the SEI Information Management staff, made significant contributions to the overall organization and readability of the final product. The author reserves for himself responsibility for both the opinions expressed and the errors that undoubtedly remain.

References

1. Ada Information Clearinghouse. "STANFINS-R—COBOL and C Programmers Moving Successfully to Ada". *Ada Information Clearinghouse Newsletter 8*, 2 (June 1990), 10,16.

2. Barton, Timothy J. and Dellenback, Steven W. An Investigation and Comparison of the C and Ada Programming Languages for Use In the Space Station Control Center and Space Station Training Facility. Tech. Rept. NASA Grant No. NAG 9-435, Southwest Research Institute, San Antonio, TX, March 1991.

3. Booch, Grady. *Software Components with Ada -- Structures, Tools, and Subsystems.* Benjamin/Cummings, Menlo Park, CA, 1987.

4. Booch, Grady. *Object-Oriented Design with Applications.* Benjamin/Cummings, Redwood City, CA, 1991.

5. Caldwell, Bruce. "Declaring War on IS Inefficiency". *Information Week* , (March 11, 1991), 26-32.

6. Cardelli, Luca and Wegner, Peter. "On Understanding Types, Data Abstraction, and Polymorphism". *ACM Computer Surveys 17*, 4 (December 1985), 471-522.

7. Commercial Ada Users Working Group. "Birds-of-a-Feather Session on X-Windows and Ada". *CAUWG Report 6*, 1 (April 1991), 3-4.

8. Crafts, Ralph . "Prompt Payment MIS System Developed in Ada". *Ada Strategies 4*, 3 (March 1990), 1-5.

9. Crafts, Ralph . "Ada Used for Personnel Management System". *Ada Strategies 4*, 4 (April 1990), 10-13.

10. Dewhurst, Stephen C. and Stark, Kathy T. *Programming in C++.* Prentice-Hall, Englewood Cliffs, 1989.

11. Eckel, Bruce. *Using C++.* Osborne McGraw-Hill, Berkeley, CA, 1990.

12. Ellis, Margaret and Stroustrup, Bjarne. *The Annotated C++ Reference Manual.* Addison Wesley, Reading, MA, 1990.

13. Federal Aviation Administration. *Advanced Automation Systems Experiences with Ada*, McLean, VA, May 15, 1991.

14. Feuer, Alan R. and Gehani, Narain (Eds.). *Comparing and Assessing Programming Languages: Ada, C, and Pascal.* Prentice-Hall, Englewood Cliffs, 1984.

15. Fussichen, Kenneth. Why COBOL Programmers Refuse Ada. Proceeding of TRI-Ada '90, ACM, December 1990, pp. 494-500.

16. Habermann, A.N. A Brief Comparison Between Ada and Unix + C. Tech. Rept. SEI-85-M-4, Software Engineering Institute, Pittsburgh, PA, June 1985.

17. Hill, David, et al. A Plan for Corporate Information Management for the Department of Defense. Tech. Rept. of the Executive Level Group for Defense Corporate Information Management, Department of Defense, The Pentagon, September 11, 1990.

18. IBM Corporation. Lanquage Selection Analysis Report. Tech. Rept. FAA-85-S-0874, IBM Federal Systems Division (prepared for the Federal Aviation Administration), Gaithersburg, MD, May 1985.

19. IEEE Computer Society and USENIX Association. *The Fifth Workshop on Real-Time Software and Operating Systems*, Washington, D.C., May 12-13, 1988.

20. Riehle, Richard . "Wells Fargo: Investing In (and With) Ada". *Ada Strategies 5*, 4 (April 1991), 1-7.

21. Robinson, Mary L. Comparison of Ada and C for SSCC and SSTF. Tech. Rept. , Mitre Corporation, Houston, TX, March 1991.

22. Shaw, Mary; Almes, Guy T.; Newcomer, Joseph M.; Reid, Brian K. and Wulf, Wm. A. "A Comparison of Programming Languages for Software Engineering". *Software: Practice and Experience 11*, 1 (January 1981), 1-52.

23. Stroustrup, Bjarne. *The C++ Programming Language.* Addison Wesley, Reading, MA, 1986.

24. United States Department of Defense. *Reference Manual for the Ada Programming Language.* American National Standards Institute, New York, 1983.

25. Weicker, Reinhold P. "Dhrystone: A Synthetic Systems Programming Benchmark". *Communications of the ACM 27*, 10 (October 1984), 1013-1030.

26. Wild, Frederic H. Comparison of Experiences with the Maintenance of Object-Oriented Systems: Ada vs. C++. Proceeding of TRI-Ada '90, ACM, December 1990, pp. 66-73.

Appendix A: List of Interviews

James Alstadt
Senior Systems Engineer
Hughes Radar System Group
Los Angeles, CA

Brett Bachman
Vice President and General Manager, Object-Oriented Products Group
Rational
Mountain View, CA

Linda Brown
Information Technology Group
Center for Information Management
The Pentagon

Emil Gottwald
Digital Systems Division
Xerox Corporation
Rochester, NY

Sam Harbison
Software Consultant
Pine Creek Software
Pittsburgh, PA

Paul Hilfinger
Ada Distinguished Reviewer
New York University
New York, NY

Gary McKee
Software Consultant
McKee Consulting
Littleton, CO

Katheryn Miles
Computer Specialist, Software Standards Validation Group
National Institute for Standards and Technology
Washington, DC

Mike Vilot
Independent Consultant
ObjectWare Inc.
Nashua, NH

Bill Wessale
Software Project Engineer for Space Station Training Facility
CAE-Link
Houston, TX

Thomas Wilcox
Senior Software Developer
Rational
Santa Clara, CA

Fred Wild
Project Manager
Cadre Technologies, Inc.
Providence, RI

Appendix B: Some Non-Government Ada Applications

Some Non-Governmental Ada Applications - December 1990

Sources: published reports, vendor newsletters, etc.
Summarized by Michael B. Feldman, George Washington University, December 1990
Additions by Nelson H. Weiderman, June 1991.

Chile	Empresa Nacional de Aeronautica (ENAER), real-time avionics system, Data General/TeleSoft
Finland	Nokia Information Systems, online banking system, >100,000 lines, uses Ada as its standard programming language
France	Thompson-CSF/SDC, Air traffic control systems and simulators in Denmark, Kenya, Pakistan, Switzerland, Ireland, Belgium, The Netherlands, and New Zealand, 300,000 source lines, DG
Germany	MAN Truck and Bus Company, payroll system (1982)
Japan	Nippon Telephone and Telegraph, videotex communication system, commercially available
Japan	Nippon Telephone and Telegraph, mobile communication system, commercially available
Japan	Nippon Telephone and Telegraph, satellite communication system, commercially available
Japan	Nippon Telephone and Telegraph, database management system, commercially available
	The 4 systems above represent a total of >2,000,000 LOC.
Sweden	ESAB, robotic welding stations for use in flexible manufacturing systems, TeleSoft, VAX, and 680x0
Sweden	Volvo, materials handling system (robotic parts carts), TeleSoft
Sweden	Color display element of hospital building control and monitoring system - 1600 I/O channels, 200 dynamic color displays, lots of tasking, Meridian
Sweden	Swedish Telecom, telephone switch controller
UK	DACMAN, simulation and data monitoring system for auto engines IBM PC, 80,000 lines of code, uses tasking, Meridian
UK	Process Plant and Chemicals, chemical process control systems, >20,000 lines, PC compatibles, Alsys

USA	Dowell-Schlumberger, oil exploration simulation software, 20,000 lines, DEC
USA	Reuters, news/financial services, transaction processing for financial information, 15,000 statement prototype
USA	General Electric, hot steel rolling mill, 200,000 lines, multiple MicroVaxen, DEC Ada
USA	Boeing, 747-400 subsystem components of cockpit displays, on-board maintenance systems, secondary flight controls
USA	Arthur Andersen, accounting/auditing/consulting, several internal projects done in Ada
USA	PC-based programmer for embedded medical products
USA	Boneck Printing, job costing system, 120,000 lines, Janus/Ada
USA	LDS Hospital, medical decision support system, 40,000 lines, DOS, Alsys (for NASA, but appears to be commercial-type application)
USA	Genesis Software, Inc., complete bill-paying system, >250,000 lines, Wang VS, Alsys
USA	Motorola - cellular phone switch testing system
USA	Xerox - Digital Systems Department decision to use Ada for all embedded copier S/W
USA	HP - hardware CAD system for internal use in chip development
USA	Wells-Fargo Investment Advisors (WFNIA), real-time investment database system, 50,000 lines, DEC/VAX

UNLIMITED, UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS None
2a. SECURITY CLASSIFICATION AUTHORITY N/A 2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for Public Release Distribution Unlimited
4. PERFORMING ORGANIZATION REPORT NUMBER(S CMU/SEI-SR-4	5. MONITORING ORGANIZATION REPORT NUMBER(S) Special Report
6a. NAME OF PERFORMING ORGANIZATION Software Engineering Institute 6b. OFFICE SYMBOL (if applicable) SEI	7a. NAME OF MONITORING ORGANIZATION SEI Joint Program Office
6c. ADDRESS (City, State and ZIP Code) Carnegie Mellon University Pittsburgh PA 15213	7b. ADDRESS (City, State and ZIP Code) ESD/AVS Hanscom Air Force Base, MA 01731
8a. NAME OFFUNDING/SPONSORING ORGANIZATION SEI Joint Program Office 8b. OFFICE SYMBOL (if applicable) ESD/AVS	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F1962890C0003
8c. ADDRESS (City, State and ZIP Code) Carnegie Mellon University Pittsburgh PA 15213	10. SOURCE OF FUNDING NOS. PROGRAM ELEMENT NO: 63752F PROJECT NO.: N/A TASK NO: N/A WORK UNIT NO.: N/A

11. TITLE (Include Security Classification)
A Comparison of Ada 83 and C++

12. PERSONAL AUTHOR(S)

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Yr., Mo., Day)	15. PAGE COUNT
Final	FROM TO	June 1991	60 pp.

16. SUPPLEMENTARY NOTATION

17. COSATI CODES

FIELD	GROUP	SUB GR.

18. SUBJECT TERMS (Continue on reverse of necessary and identify by block number)
Ada MIS
C++
comparison of Ada and C++ languages

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The purpose of this report is to provide technical input to the Deputy Assistant Secretary of the Air Force for Communications, Computers, and Logistics to assist that office in preparing a business case for using Ada or C++ to develop Corporate Information Management (CIM) systems. This technical input has been gathered by using the comparison methodology of a 1985 Federal Aviation Administration (FAA) report as a model, as well as by conducting interviews with experts in Ada and C++. The conclusion of this report is that technically neither language is clearly better than the other; for government use, however, there is clear justification and rationale for using ada rather than C++ for large complex systems with long lifetimes.

(please turn over)

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED ■ SAME AS RPT DTIC USERS ☐ ■	21. ABSTRACT SECURITY CLASSIFICATION Unclassified, Unlimited Distribution

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE NUMBER (Include Area Code)	22c. OFFICE SYMBOL
John S. Herman, Capt, USAF	(412) 268-7630	ESD/AVS (SEI