(2)

ADA AND C++ BUSINESS CASE ANALYSIS

JULY, 1991

Ada and C++: A Business CASE Analysis
(CIM Collection)

Lloyde K. Mosemann II, Deputy Assistant Secretary of the Air Force (Communications, Computers and Logistics) (ed.)

SAF/AQK

None Assigned

Deputy Assistant of the Air Force (Communications Computers, and Logistics)

None Assigned

None

Distribution Unlimited

The FY91 DoD Appropriations Act prescribes, in part, that effective June 1 1991 "where cost effective, all Department of Defecse software shall be written in the programming language Ada, in the absence of a special exemption by an official designated by the Secretary of Defense." This report documents a business case conducted to determine under what circumstances a waiver to DoD's Ada requirement (DoDD 3405.1) might be warranted for the use of C++, particularly in the Corporate Information Management (CIM) program. There is no intention to question DoD's commitment to Ada, but only to identify when waivers for C++ might be warranted.

Several different approaches were undertaken to identify, from a business perspective, when the lifecycle cost effectiveness of C++ might be greater than that of Ada.
Each of the substudies reached the same conclusion: there are no compelling reasons to waive the Ada requirement to use C++.

Ada, C++, Corporate Information Management, programming language, Compiler , CIM (collection) 326

UNCLASSIFIED

7/1/92

ADA AND C++: A BUSINESS CASE ANALYSIS

Deputy Assistant Secretary
of the Air Force
(Communications, Computers, and Logistics)
Washington, D.C. 20330-1000

July 1991

FOREWORD

There are two fundamentally different ways to develop software. The first I will call "software as art." The emphasis here is on the skill of the practitioner, and the United States has achieved a lead in the world in software because we have many very, very good software practitioners. In developing software as an art, we put few restrictions on the practitioner: we want to give him/her powerful tools, flexible environments, and few restrictions in order to optimize his/her creativity. We have done so well with software as an art that one might ask why we would consider any other way.

The answer is that factors beyond our control are making software as art too costly. In the first place, requirements for software are increasing drastically, outstripping our supply of highly skilled practitioners. In the second place, and perhaps more seriously, software systems are getting much larger and more complex, exceeding the intellectual bounds of single practitioners. Today we frequently measure the size of software systems in millions of lines of code, whereas in the recent past a system with a few hundred thousand lines of code was more the norm. Because software costs increase with complexity, and complexity increases superlinearly with system size, large systems developed as art cannot be afforded.

There is a useful alternative method for software development that exhibits fewer disadvantages for many classes of software. This method is "software as engineering." Software engineering, rather than placing a premium on the skill of the practitioner, places a premium on the quality of the engineering process. Creativity is valued less than discipline. Engineering produced the Brooklyn Bridge; art produced Michelangelo's David. They are two fundamentally different kinds of products because they have two fundamentally different purposes.

Software for large, complex applications, such as in the Department of Defense, must be engineered. Such systems are too large to develop cost-effectively using software as art. Few programming languages, however, specifically facilitate software engineering, and thus few are appropriate for the military software task. Some years ago Ada was developed to fill this breach, to provide a programming language which facilitates and even encourages deliberate and careful engineered design of large, complex computer applications. Although, for this reason, Ada has been mandated as the DoD language of choice, the popular C language has given rise to C++, which claims many Ada attributes. Accordingly, to ensure DoD software is being engineered with the most appropriate language, this report examines under what circumstances, if any, use of C++ over Ada might be justified for DoD software developments.

Washington, DC
June, 1991

Lloyd K. Mosemann II
Deputy Assistant Secretary
of the Air Force for
Communications, Computers,
and Logistics

ADA AND C++: A BUSINESS CASE ANALYSIS

EXECUTIVE SUMMARY

The FY91 DoD Appropriations Act prescribes, in part, that effective June 1, 1991 "where cost effective, all Department of Defense software shall be written in the programming language Ada, in the absence of a special exemption by an official designated by the Secretary of Defense." This report documents a business case conducted to determine under what circumstances a waiver to DoD's Ada requirement (DoDD 3405.1) might be warranted for the use of C++, particularly in the Corporate Information Management (CIM) program. There is no intention to question DoD's commitment to Ada, but only to identify when waivers for C++ might be warranted.

Several different approaches were undertaken to identify, from a business perspective, when the lifecycle cost effectiveness of C++ might be greater than that of Ada. The first, conducted by the Institute for Defense Analyses (IDA), examined availability of tools and training for the two languages. The second, conducted by the Software Engineering Institute, applied to this problem a quantitative language selection methodology developed by IBM for the Federal Aviation Administration (FAA). The third, conducted by CTA, Inc., using data from Reifer Consultants' repositories, analyzed cost and productivity data from existing Ada and C++ projects (macro cost analysis). And the fourth, conducted by the TRW Corporation, applied a standard cost model in depth to both languages for a typical information systems/C^3 project (micro analysis). In addition, the Naval Postgraduate School (NPS) was asked to address the overall policy issue of Ada, particularly in the context of emerging fourth-generation language (4GL) software technology.

Each of the substudies reached the same conclusion: **there are no compelling reasons to waive the Ada requirement to use C++.**

The business case analysis documented herein was directed at information systems and C^3 systems. However, there is no reason to believe the results would differ for computer programs embedded in weapons systems.

ADA AND C++: A BUSINESS CASE ANALYSIS

CONTENTS

SECTION 1. BUSINESS CASE PLAN

DoD Directives 3405.1 and DoD Instruction 5000.2, as well as the FY91 DoD Appropriations Act, mandate use of Ada, where cost effective, for all applications. In recent months, "object-oriented programming (OOP)" has emerged as a promising software technology, so that the language C++, developed to exploit OOP, has been thought by some to be potentially better than Ada for some DoD applications.

This report describes a business case conducted to determine under what circumstances a waiver to the DoD Ada requirement might be warranted for use of C++, particularly in DoD's Corporate Information Management (CIM) program. There is no intention to question DoD's commitment to Ada, but only to identify when waivers for C++ might be warranted. This business case will support a proposal for DoD programming language policy for information systems and C^3 systems.

Programming language selection is not the major cost driver in software. Recent thinking focuses instead on controlling costs through software engineering throughout the lifecycle, including initial development, acquisition, and post deployment support. Typically, software design, coding, integration, and test consumes 30-40% of software lifecycle cost while software support accounts for the remainder (see Figure 1). Yet programming language choice is not unimportant. A language facilitating software engineering can produce code easier to learn and understand, easier to change, easier to reuse, and easier to interface with specification language/CASE technology.

Business considerations would indicate a waiver for C++ when the ratio of benefit to cost for C++ exceeds the same ratio for Ada. We thus identified the factors making up these ratio components and called upon experts to quantify them. Four different substudies were required. The first, conducted by IDA, examined availability of tools and training for the two languages (Appendix A). The second, conducted by the Software Engineering Institute, applied to this problem a quantitative language selection methodology developed by IBM for the Federal Aviation Administration (FAA) (Appendix B). The third, conducted by CTA Incorporated, analyzed cost and productivity data from existing Ada and C++ projects (macro cost analysis) (Appendix C). And the fourth, conducted by TRW, applied a standard cost model in depth to both languages for a typical information systems/C^3 project (micro analysis) (Appendix D).

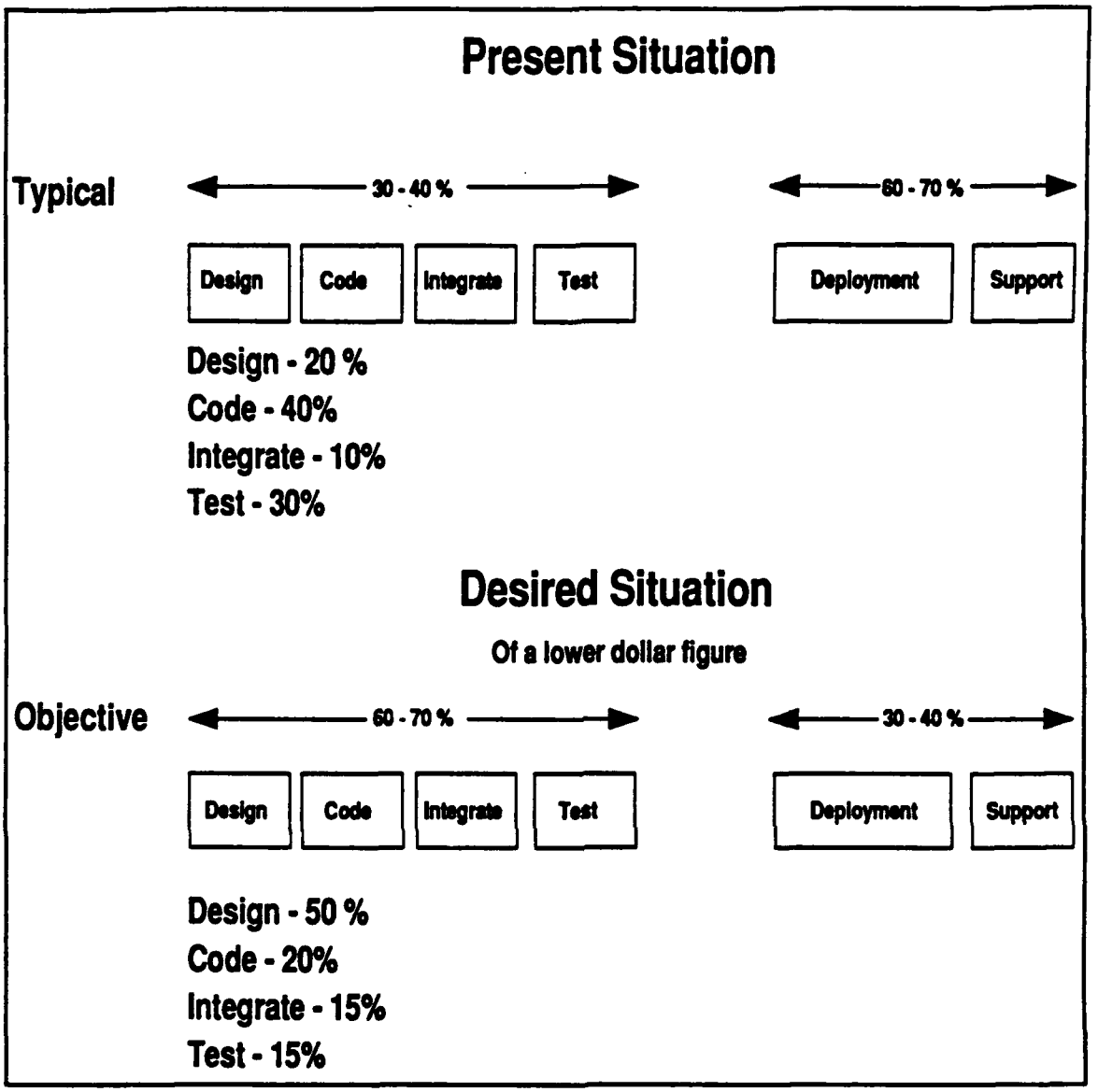

Figure 1 Software Development Cost Percentage in Lifecycle Phases. Traditionally, coding and testing accounted for the majority of software's cost during development. The desired objective is to place more emphasis upon design.

Results of these studies are summarized in Section 2. Also included in Section 2 is a summary of a parallel study (excerpted in Appendix E) on Ada usage policy issues conducted by the Naval Postgraduate School. Conclusions drawn are presented in Section 3.

SECTION 2. BUSINESS CASE RESULTS

A. Tools, Environments, and Training: IDA Substudy.

The Institute for Defense Analyses (IDA) collected and analyzed information (Appendix A) on the market availability of commercial off-the-shelf products available from U.S. sources for Ada and C++ compilers, tools, education, and training. Their primary findings follow.

There are 28 companies located in the U.S. that have Ada compilers with currently validated status; 18 vendors offer C++ compilers. The Ada compiler vendors are more likely to have been in business five years or more. Ada "validation" is more rigorous than that of other high order languages: only Ada is monitored and approved for conformity to a standard, without supersets or subsets, by a government-controlled process. By contrast, no validation or standard of any kind exists for C++, although a standard by 1994 is expected.

Both languages are supported on PCs and workstations. Ada is also supported on mainframes. Ada, but not C++, has cross compilation systems.

Ada is supported with program engineering tools. Compiler vendors provide a rich set. Code generators exist for Ada but none so far for C++. There is considerable variability among C++ products in language features supported and libraries provided.

Ada is taught in 43 states at 223 universities and 13 DoD installations. C++ is taught in four states at four universities and no DoD installations. There are more Ada than C++ courses available. The cost of training is about equal, but Ada course variety is wider.

B. Faceted IBM Language Selection Methodology: SEI Substudy.

The Federal Aviation Administration (FAA) contracted with IBM in the mid-1980s to evaluate high order languages for use on its Advanced Automation System (AAS) Program. In response, IBM developed a formal, quantitative faceted methodology comparing 48 language features (criteria) in six categories. The study concluded that 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."

Using this same methodology for each of the 48 criteria, the Software Engineering Institute (SEI) evaluated Ada and C++ for application in information systems/C^3 systems (Appendix B). The original FAA weighted scores for the six criteria categories were as follows:

Category	Max. Score	Ada	C	Pascal	JOVIAL	FORTRAN
Capability	16.7	16.1	9.6	10.4	7.6	3.9
Efficiency	16.4	8.0	11.8	10.8	11.0	11.1
Availability/Reliability	22.6	21.5	11.6	14.5	15.6	10.3
Maintainability/Extensibility	17.4	14.0	10.2	12.2	6.8	8.3
Lifecycle cost	11.3	8.2	7.4	7.8	4.9	5.2
Risk	15.6	8.8	8.9	7.6	9.6	8.2
Total	100.0	76.6	59.5	63.3	55.5	47.0

The 1991 weighted scores for the six criteria categories were:

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
Lifecycle cost	11.3	8.4	8.0
Risk	15.6	11.7	9.8
Total	100.0	78.8	63.9

In 1985 Ada was considered considerably more capable than C. Today, there is still a significant difference between Ada and C++, C's successor. Relative efficiency of Ada has improved markedly; Ada still scores significantly higher in availability/reliability; the Ada advantage in maintainability/extensibility persists; and from a position of parity in 1985, Ada has attained in 1991 a significant advantage over C++ in lowered risk.

An attachment (Appendix B) lists numerous major Ada information systems/C^3 systems. It is not widely appreciated that such extensive use is now being made of Ada: in fact, the U.S. Ada market, excluding training, services, and government research/development, now exceeds $1 billion (Ada 9X Project Office, June1991).

C. Macro Cost Analysis: CTA Substudy.

CTA compiled and compared available productivity and cost data of Ada and C++ (Appendix C). Results are summarized below. Much of the data comes from Reifer Consultants, Incorporated (RCI) extensive database.

Average productivity across the four domains for which data exists (environment/tools, telecommunications, test (with simulators) and other) for both Ada and C++ projects was:

	Productivity (SLOC/MM)	Number of Data Points
* Norm (all languages)	183	543
* Average (Ada)	210	153
* Average (C++)	187	23
* First project (Ada)	152	38
* First project (C++)	161	7

C++ project data reflected information on 23 projects taken from seven firms who had been using C++, Unix, and object-oriented techniques for over 2 years. All projects were new developments. Application size ranged from 25 to 500 KSLOC (thousand source lines of code). Average size was about 100 KSLOC.

Average costs across the four domains for both Ada and C++ projects were:

	Cost ($/SLOC)	Number of Data Points
* Cost (all languages)	70	543
* Average (Ada)	65	153
* Average (C++)	55	23

Typically, Ada developments were in accordance with military standards and incorporated formal reviews, additional documentation, and additional engineering support activities such as formal quality assurance (QA) and configuration management (CM). Most C++ projects are commercial and do not extensively incorporate such activities. Additionally, on such projects developers are typically intimately involved with users, resulting in considerably less requirements engineering effort. Consequently, applications on which C++ is used are inherently less costly, so that reported productivity rates are favorably skewed toward C++.

Average error rates across the four domains for both Ada and C++ projects were:

	Integration Error Rates (Errors/KSLOC)	FQT Error Rates Errors/KSLOC)	Number of Data Points
* Norm (all languages)	33	3	543
* Average (Ada)	24	1	153
* Average (C++)	31	3	23

Integration error rates include all errors caught in test from start of integration testing until completion of software Formal Qualification Test (FQT). The FQT error rate includes only those errors found during the FQT process.

Parametric analysis reveals that Ada and C++ are comparable software technologies, each with a learning curve and transition period. Factors to which both Ada and C++ projects tend to be most sensitive are:

* Learning curve - it takes 3-5 projects' worth of experience before a team has the experience to effectively use the features of either language (about two years' time).

* Investment strategy - it takes substantial investments in tools, methods, equipment, and training to tap benefits of either language (about $10,000 per person).

* Process discipline - effective use of either language assumes that the firm has scored as a level 2 on the SEI process assessments rating scale (Humphrey, 1989).

Transition state analysis (method of moving averages) indicates that 26 of the 38 firms within the Ada database have successfully made the changeover to effective use of Ada, while none of the 7 firms in the C++ database have made the transition. Also, none of the 7 firms were fully using C++'s inheritance and other advanced features.

The standardization maturity of Ada is important. While Ada has a firm and well policed standard, allowing neither supersets nor subsets, it will be years before a stable C++ language specification is established. New features are being considered for the latest standard C++ release. Vendors are likely to offer their own enhanced versions of C++ compilers and CASE tools, complicating portability and reuse.

"Class libraries" for C++ are being offered commercially for reuse, similar to "component libraries" for Ada. For the most part, project unique reusable components appear to be the more valuable. Examples of highly leveraged reused project components exist for both languages. The C++ language provides easy access to extensive existing system software implemented in C (such systems include graphics support, communication protocols, operating systems, and database management). Effective Ada bindings to such systems are emerging, however, so no particular advantage is ascribed to either language from the standpoint of availability of reusable existing software.

Finally, the original arguments for establishing a single programming language for military applications remain. Common training, tools, understanding, and standards simplify acquisition, support, and maintenance. After maturing for a decade, Ada's benefits have been proven for all application classes.

In conclusion, Ada projects have reported 15% higher productivity with increased quality and double the average size. Normalizing these data to comparable size projects would result in an expected Ada productivity advantage of about 35%. Ada should be the near term language of choice. C++ still needs significant maturing before it is a low risk solution for a large DoD application.

D. Micro Cost Analysis: TRW Substudy.

TRW performed a tradeoff analysis (Appendix D) that generalized recent corporate cost analyses on a typical real-world information systems/C^3 systems project. Their substudy defined a set of maximally independent criteria, judged each language with respect to those criteria, and then translated those judgments into cost impacts to emphasize the importance of each criterion from a lifecycle cost perspective. Results were translated into perturbations of the Ada COCOMO cost model (Boehm, 1981).

Rankings of the two languages based on this analysis are provided below (0 = no support; 5 = excellent support), followed by a total score, a weighted sum of the rankings based on weights determined by an expert panel:

	Ada	C++
Reliable S/W Engineering	4.5	3.2
Maintainable S/W Engineering	4.4	3.2
Reusable S/W Engineering	4.1	3.8
Realtime S/W Engineering	4.1	2.8
Portable S/W Engineering	3.6	2.9
Runtime Performance	3.0	3.6
Compile-Time Performance	2.3	3.1
Multilingual Support	3.1	2.4
OOD/Abstraction Support	3.9	4.6
Program Support Environment	4.1	2.1
Readability	4.4	2.9
Writeability	3.4	3.5
Large Scale S/W Engineering	4.9	3.3
COTS S/W Integration	2.8	3.6
Precedent Experience	3.6	1.5
Popularity	2.8	4.0
Existing Skill Base	3.0	1.8
Acceptance	2.5	3.3
TOT SCORE FOR MIS (Ada score is 23% higher)	1631	1324
TOT SCORE FOR C^3 SYSTEMS (Ada score is 24% higher)	1738	1401

Both Ada and C++ represent improved vehicles for software engineering of higher quality products. Currently, C++ is approximately 3 years behind Ada in its maturity and tool support. The case study used in this report (the Command Center Processing and Display System--Replacement) demonstrated development cost advantages for Ada on the order of 35% and maintenance cost advantages for Ada on the order of 70% under today's technologies. In the far term (1994+), this Ada advantage may erode to approximately a 10% advantage in development costs and 30% in maintenance costs for a typical development intensive system.

The primary strengths of Ada are in its support for realtime domains and large scale program development. Its primary weaknesses are its compile-time and runtime efficiency. The primary strengths of C++ are its better support for object oriented design, support for COTS integration, and its compile-time and runtime efficiency. Its main weaknesses are its support for reliability and large scale program development. In general, Ada's weaknesses are solved by the

ever-increasing hardware performance and compiler technology advancement. C++ weaknesses must be solved by advances in its support environment.

The substudy report concludes with a set of DoD programming language policy recommendations.

E. Ada Policy Issues: NPS Study.

Concurrently with the preparation of this Ada and C++ Business Case Analysis, the Naval Postgraduate School (NPS) reported (Appendix E) on policy issues on the use of Ada for Management Information Systems. A summary of their report (an analysis of the need to see Ada in a total and evolving context) is included below as an important vision statement leading from Ada as the primary third-generation language (3GL) to its conception as the basis for evolving to higher levels of productivity in so-called $3^1/_2$ GL and 4GL environments. Also included below are our comments on $3^1/_2$ GL and 4GL issues not addressed in the NPS report but relevant to this business case.

1. Report Summary.

Rather than concentrating on programming language selection, the NPS report focuses on and argues for needed advances in software development technology. In particular, the Report contends, while traditional factors such as programming language selection, better training, and computer-assisted software engineering (CASE) tools can enhance productivity modestly, a fundamental change in the software development paradigm will be necessary to achieve an order of magnitude gain. Such a gain is possible through use of 4GLs, languages that will ultimately enable the developer to define the complete design of an application entirely in the 4GL's own high-level specification language. The specification is then translated automatically by the 4GL into an executable program. When accompanied by a productive development environment, an evolutionary implementation methodology, and well trained development teams, the report asserts, 4GLs can provide a tenfold gain in productivity (Emery and McCaffrey, 1991).

An intermediate step in the movement to 4GLs is $3^1/_2$ GL programming, a term referring to the extensive use of CASE tools coupled with a high level of code reuse. The $3^1/_2$ GL approach requires a strong commitment to codifying and accrediting code modules, to the point where it becomes easier and more desirable to reuse code than to rewrite it.

Although experience with 4GLs has not yet been extensive (with existing experience limited largely to specific functional domains such as financial management and transaction processing), 4GLs are attractive for several reasons. One is their robustness under change: changes made to the application, for whatever reason, are made at the specification level and then re-translated automatically into executable code. Another is the facility with which they can be integrated into tightly knit and full-featured development environments. For these reasons, the report strongly recommends that the DoD discourage use of traditional 3GL programming and take bold steps to incorporate the 4GL paradigm.

Finally, the report recommends that, given the importance of Ada to DoD software, greater effort and funding should be provided for the key Ada initiatives: the Ada Technology Improvement Program, Ada 9X (see description in Section III, following), and Ada education initiatives.

2. Comments on Related Issues.

Two issues on $3^1/_2$ GLs and 4GLs related to this business case were outside the scope of the NPS report. The first of these is that, for the foreseeable future, state-of-the-art limitations will probably keep 4GLs from generating more than half the total code required by many applications. In such cases, where a substantial amount of 3GL programming will be required to complete application development, use of a $3^1/_2$ GL approach, rather than a 4GL approach, is preferable.

Another issue outside the scope of the NPS report was the evaluation of the relative merits of Ada and C++ as target (output) languages for 4GL application generators. However, as section V.C of the NPS report points out, a "standard, stable target language portable to a variety of hardware platforms" with good software reuse and interface definition capabilities is appealing. Although more study of the characteristics desired in 4GL target languages is warranted, the SEI and TRW substudies suggest no particular advantage of C++ over Ada in these desirable attributes, so there appears to be no reason to waive DoD's Ada requirement in favor of C++ as a target language for 4GLs.

SECTION 3. CONCLUSIONS.

All four substudies which specifically compared Ada and C++ (IDA, SEI, CTA, TRW) report a significant near term Ada advantage over C++ for all categories of systems. This advantage could be eroded as C++ and its supporting environments mature over the next few years. On the other hand, as aggressive overseas Ada initiatives stimulate even wider domestic Ada interest, as Ada tools/environments further mature, and when the Ada update (Ada 9X) is complete, the balance could tip further in Ada's favor.

Adding to the case for Ada is the fact that the Ada scoring so well herein is Ada's 1983 version, MIL-STD-1815A. Just as C++ incorporates into C certain software engineering concepts already in Ada (e.g., modularity, strong typing, specification of interfaces), so an Ada update now underway will bring into Ada selected features now included in C++. This update, known as the Ada 9X Project, is targeted for completion in 1993 (Ada 9X, February 1991). The product of extensive community involvement (including the C^3 and MIS communities), Ada 9X will bring to Ada such improvements as decimal arithmetic, international character sets, improved input/output, support for calls between Ada and other languages, further representation specifications, and inheritance/polymorphism (popular features of C++).[1] At the same time, Ada 9X has been designed so that neither existing Ada benefits nor performance will be lost. For example, Ada 9X inheritance will be controlled so as not to reduce lifecycle supportability.[2]

In summary, Ada is the most cost effective programming language for DOD applications. Specifically, it is not possible to make a credible case for the existence of classes of "more cost effective" C++ systems compared to Ada. Business cost effectiveness data collected for this study are typified by the TRW conclusion (Appendix D) that Ada provides development cost advantages

---

[1] "One of the goals of Ada 9X is to provide all the flexibility of C++ with the safety, reliability, and understandability of Ada 83" (Ada 9X Project Office, June 1991).

[2] Controlling the employment of OOP features is of substantial importance to DoD software mission goals (e.g., safety, reliability, and dependability). Wild (Wild, 1990) makes the case that

> the concept of inheritance in C++ [is not] attractive. The leap in complexity introduced by it does not balance well with the benefits of using it. Big mistakes are being made with it because people have not been sufficiently careful in looking at its scope of applicability or its side-effects on the maintainability of systems. Nor do people generally understand how to create a well structured class hierarchy.

Bjarne Stroustrup himself, the originator of C++, has been quoted as follows: "C makes it easy for you to shoot yourself in the foot. C++ makes that harder, but when you do, it blows away your whole leg" (Polese and Goldstein, 1991).

on the order of 35% and maintenance cost advantages on the order of 70%.[3] In terms of full lifecycle costs, it will be some time before data exists which could justify a cost savings for C++. Today, there is limited lifecycle data available for Ada and almost none for C++.

For the foreseeable future, then, there are more than enough reasons for the DoD to stick firmly with Ada, both for all high order language (3GL and $3^1/_2$ GL) development and for exclusive use as a target language of 4GL application generators in the large class of applications for which 3GL code must supplement generated code.

3 Ada's advantages are not, however, widely appreciated. Dean vendeLinde, of the Whiting School of Engineering at Johns Hopkins University, asserts this is true in large part because C have become entrenched at U.S. colleges and universities, providing industry with a ready source of graduates familiar with and thus partial to these languages (vandeLinde, 1990). However, this entrenchment in academia appears to be principally a matter of tool availability (due to the rich UNIX/C heritage) rather than of language technical merit. Some universities are now switching to Ada to capitalize on Ada's support for software engineering: for example, the University of Washington has adopted Ada as the language of choice for all computer science classes (Ada 9X Project Office, June 1991). It has further recently been reported that "every university in England teaches Ada. It has become the *de facto* standard software language in the U.K." (Mossakowsky, 1991).

BIBLIOGRAPHY

Ada 9X Project Office, "Ada 9X Revisions Relating to Information Systems Applications." Fact Sheet, February 1991.

--------, private communication, June 1991.

Ada Information Clearinghouse (AdaIC) Newsletter, "The Congressional Ada Mandate." IIT Research Institute, Lanham, MD, March 1991, p. 18.

"Alsys Executive Briefing on Ada." Burlington, MA: Alsys, Inc., 1990.

Boehm, B.W., *Software Engineering Economics*. Englewood Cliffs, NJ: Prentice-Hall, 1981.

Brooks, F.P., Jr., "No Silver Bullet: Essence and Accidents of Software Engineering." *Computer 20*, 4 (April 1987), pp. 10-19.

Caldwell, B., ed., "A New Bible for MIS." *Information Week*, (October 15, 1990), pp. 36-40.

Department of Defense Directive 3405.1, "Computer Programming Language Policy," April 2, 1987.

Department of Defense Instruction 5000.2, "Major System Acquisition Process," February 23, 1991.

Emery, J. and L. Gremillion, "Adopting a New Software Development Paradigm: A Strategic Imperative." Working Paper, April 1991.

Emery, J.C. and M.J. McCaffrey, "Ada and Management Information Systems: Policy Issues Concerning Programming Language Options for the Department of Defense." Monterey, CA: Naval Postgraduate School, Department of Administrative Sciences, June 1991.

Gray, M. and G. Fisher, "Functional Benchmarks for Fourth Generation Languages." National Institute for Standards and Technology, Gaithersburg, MD, March 1991.

Halbert, J., "Embedded Real-Time Multitasking Kernel." *The C Users Journal*, March 1991, pp. 33-37.

Heller, S., "Gray Skies Frowning at Me." SHARE, May 1991, pp. 131-134.

Humphrey, W.S., *Managing the Software Process*. Reading, MA: Addison-Wesley, 1989.

IBM Federal Systems Division, "Federal Aviation Administration Advanced Automation System Design Competition Phase Language Selection Analysis Report." IBM Corporation, May 20, 1985.

IBM Measurements Group, "Line-of-Code to Function Point Conversion." Stamford, CT.

Kaplan, H.T., "The Ada COCOMO Cost Estimating Model and VASTT Development: Estimates vs. Actuals." *Vitro Technical Journal 9*, 1 (Winter 1991), pp. 48-60.

Knight, J., "On the Assessment of Ada Performance." *Ada Letters* Special Edition, Software Productivity Consortium.

Marsh, A., "Is It Time to Throw Ada Out?" *Government Executive*, April 1991, pp. 39-40.

Merriam, R., "Paradigm Versus Performance." *Embedded Systems Programming*, March 1991, pp. 18-30.

Mossakowsky, M., quoted in "Let My People Grow!", *Aerospace and Defense Science*, Spring 1991, p. 40.

Pearson, G., "Array Bounds Checking with Turbo C." *Dr. Dobb's Journal*, May 1991, pp. 72-83.

Polese, K. and T. Goldstein, "OOP Language and Methodologies", *SunWorld*, May 1991, pp. 45-59.

"Reserve Demobilization System Built Around Reused Ada Code," *Government Computer News*, May 13, 1991, p. 63.

Rowe, J., "Producing With Ada: Real-Time Solutions for Complex Problems." *Defense Electronics*, July 1990, pp. 41-51.

Sammet, J., "Why Ada is Not Just Another Programming Language." *Communications of the ACM 29*, 8 (August 1986), pp. 722-732.

Strassmann, P., *The Business Value of Computers*. New Canaan, CT: The Information Economics Press, 1990.

vandeLinde, D., address to TRI-Ada '90 Symposium, Baltimore, MD, December 4, 1990.

Wild, F., "A Comparison of Experiences with the Maintenance of Object-Oriented Systems: Ada vs. C++." Association for Computing Machinery, *TRI-Ada '90 Proceedings*, pp. 66-73.

APPENDIX A. Excerpts (IDA Substudy)

This appendix contains the following excerpts of the IDA substudy: sections 1 through 3, appendix E, F, and H. The full report contains an additional five appendices.

UNCLASSIFIED

IDA PAPER P-2601

AVAILABILITY OF Ada AND C++

COMPILERS, TOOLS, EDUCATION, AND TRAINING

Audrey A. Hook, Task Leader

William E. Akin
Lewis E. Dimler
Kathleen A. Jordan
R. Danford Lehman
Catherine W. McDonald
Christine Youngblut

12 June 1991

Prepared for

Director of Defense Information
Office of the Assistant Secretary of Defense (OASD)
Command, Control, Communications, and Intelligence (C3I)

INSTITUTE FOR DEFENSE ANALYSES
1801 N. Beauregard Street, Alexandria, Virginia 22311

IDA Log No. HQ 91-038328/1

UNCLASSIFIED

PREFACE

IDA Paper P-2601, "Availability of Ada and C++ Compilers, Tools, Education and Training" presents the results of a five-week study to determine the comparative availability of compilers, tools, education and training for the Ada and C++ languages.

The delivery of this paper fulfills IDA Task Order T-J5-954 "to identify, analyze, and report on (1) compiler and automated engineering tools that can support and supplement current software development, integration, test, and support functions of Ada and C++ programming languages and (2) associated training and education available for each language." This report will be one of several information sources used by DoD in the development of a business case to determine whether any waivers to the Ada requirement may be warranted for business sytems.

This document was reviewed by the following members of IDA: Dr. Richard Morton, Dr. Richard Wexelblat, and Dr. Richard Ivanetich.

TABLE OF CONTENTS*

*This Table of Contents represent the contents of the full IDA substudy. Excerpts of the substudy are contained herein.

TABLE OF CONTENTS (Cont'd)*

1. INTRODUCTION

The use of compilers and tools that support modern software engineering practices has the potential to greatly increase programmer productivity. Many U. S. and European companies are offering off-the-shelf products that support some aspect of the software engineering process with choices of design and development paradigms, and implementation language. The Department of Defense (DoD) is interested in the status of market offerings for software engineering environments to support the software life cycle.

The Institute for Defense Analyses (IDA) was tasked by the Director of Defense Information, Office of Assistant Secretary of Defense (C3I) to identify, analyze, and report on (1) compiler and automated engineering tools that can support and supplement current software development, integration, test, and support functions of Ada and C++ programming languages, and (2) associated training and education available for each language. This report will be one of several information sources used by the DoD in the development of a business case to determine whether any waivers to the Ada requirement for business systems may be warranted.

1.1 BACKGROUND

The Ada programming language, standardized in 1983, is Congressionally mandated for software development within the DoD. The 1983 standard, informally known as Ada83, is currently under revision in the normal American National Standards Institute (ANSI) process. Two important changes planned are an extension of Ada's data abstraction capabilities, adding object-oriented programming features and improved control over concurrency for real-time applications. The DoD has also established a rigorous compiler testing and validation process used in the U. S. and Europe as a mechanism for determining conformity to the standard.

C++ is an incremental addition to the C language that includes type checking and provides object-oriented programming features. The C language was standardized in 1989 but there is no standard for C++ and no formal compiler testing and validation process for C or C++. Thus, there could be considerable variation among the C++ products reported in this study: time constraints preclude conducting an in-depth analysis of this variability.

1.2 SCOPE

This report documents a five-week effort to collect and analyze information on the market availability of Ada and C++ compilers, tools, education, and training. We have eliminated from discussion such application domains as Artificial Intelligence (AI), Computer-aided Design (CAD), and embedded systems because the primary focus of this study is on business systems. We also excluded Fourth Generation Languages (4GLs) as a category of Computer-aided Software Engineering (CASE) tools because 4GLs are for the most part proprietary, non-procedural languages that have limited utility during the maintenance phase of a large, complex business application. Where they were reported, we made note of extended compiler libraries that provide interfaces or bindings to other Federal Information Processing Standard (FIPS) languages and protocols and to International Standards Organization (ISO) libraries. For the purpose of this report, we considered operating system services and utilities generally provided with computer systems as basic extensions to the capabilities of a software engineering environment. Finally, only commercial off-the-shelf (COTS) products available from U. S. vendors were considered in this study.

1.3 DEFINITION OF TERMS

There are many tool vendors who offer products for specific jobs during software development. Some tools are designed for use with a particular programming language, with a particular program development method, or during a specific part of the software life cycle. In this report, we have investigated the availability of tools that are coupled with compilers and those that extend software engineering support of certain phases of the software life cycle. For the purpose of this report, the following definitions of terms apply:

- Tool: A tool is a software product or package which serves a quite specific and narrow purpose for programming such as, for example, a source code editor or a static debugger.

- CASE: CASE tools are collections of tools that support specific task activities performed during the software life cycle, such as requirements analysis, preliminary design, program testing, or verification.

- Environment: An environment is used here to mean computer and communications hardware and software, including operating systems and a tool set for supporting

tasks during the software life cycle. Some degree of interoperability among tools may exist but the general translation of data structures and their semantics among tools and environments without loss of information requires further research and development (R&D).

1.4 APPROACH

Commercial suppliers of Ada and C++ compilers, CASE tools, and training in the use of Ada and C++ were contacted by telephone to solicit the information used in this study. The source of information concerning commercial suppliers was lists published by the Association for Computer Machinery (ACM) Special Interest Group on Ada (SIGAda), Ada Joint Program Office (AJPO), journals and data collected by IDA in connection with several other tasks such as Ada Technology Insertion and the Strategic Defense Initiative Office (SDIO) Software Technology Plan. Data collected during the survey was analyzed to determine current status and indications of trends of significance to information business systems. The information collected during this study is documented in Appendices A-H.

2. FINDINGS AND DISCUSSION

2.1 Ada COMPILERS AND TOOLS

There are 28 companies located in the U. S. that have Ada compilers with current validated status. The official list of validated Ada compilers published by the AJPO and National Institute of Science and Technology (NIST) pairs Ada compiler names with the computer systems that make up a validated Ada implementation.

For this survey, the following information was solicited from compiler vendors:

- products (how the compiler is marketed and any other tools)
- prices
- maturity (earliest validation date)
- education/training (includes courses and consulting)
- other languages (specifically C++)
- customer base

Table 1 provides the names of companies contacted during this survey along with data on platform type, prices, and primary business of the company.

Appendix A documents the information provided by the compiler vendors.

Table 1. Ada Compiler Vendors

	Price Range low	high	Platform PC/WS/MF*	OS	Training
1. AETECH Compilers	$795	$2495	PC	DOS UNIX	Yes
2. Aitech Systems Ltd. Systems	n/a**		n/a		
3. Alliant Computer Systems Systems	$15.000	$75,000	MF	Alliant	Yes
4. Alsys Compilers	$940 $38,000	$3,000 $7,500 $126,000	PC WS MF	MacIntosh DOS UNIX VMS MVFS	Yes
5. Apollo Computer Systems	n/a		n/a		
6. Concurrent Computer Corp. Systems	n/a		n/a		
7. CONVEX Computer Corp. Systems	n/a		n/a		
8. DDC International Compilers			WS MF	UNIX VMS	
9. Digital Equipment Corp. Systems	$15,200	 $330,000	WS MF	VMS ULTRIX	Yes
10. E-Systems, Inc. Systems	n/a		n/a		
11. Encore Computer Systems	n/a		n/a		
12. Harris Systems	$18,500	$30,000	MF	Harris	Yes
13. Hewlett-Packard Systems	n/a		n/a		
14. IBM, IBM Canada Ltd. System	$25,000 $10,000	$400,000 $38,000	MF WS	IBM UNIX	

* PC = Portable Computer; WS = Work Station; MF = Main Frame
** n/a = not available

Table 1. Ada Compiler Vendors (Cont'd)

	Price Range low	Price Range high	Platform PC/WS/MF*	Platform OS	Training
15. Intermetrics Compiler	 $50,000	$30,000	WS MF	VMS MVS	
16. Irvine Compiler Compiler	$5,000 $25,000	$18,000 $90,000	(self-host) (cross compiler)	VMS UNIX	Yes
17. Meridian Software Systems Compilers	$249	 $6,500	PC WS	MacIntosh UNIX VMS	Yes
18. MIPS Computer Systems Systems	n/a		n/a		
19. R.R. Software Compilers	n/a		PC	DOS UNIX	Yes
20. Rational Systems	$25,000	$48,000	WS/MF	Prop.	Yes
21. Rockwell International Systems	n/a		n/a		
22. SD_SCICON Systems	n/a		WS MF	VMS	Yes
23. Silicon Graphics Systems	n/a		n/a		
24. Tartan Laboratories, Inc. Compilers	$20,000 $30,000	$48,000 $140,000	WS MF	VMS UNIX	
25. TeleSoft Compilers	$4,500 $20,000	$7,500 $90,000	WS MF	Sun UNIX	Yes
26. Texas Instruments Systems	n/a		WS	VMS	
27. Verdix Compilers	n/a		WS MF	SUN OS UNIX VMS	
28. Wang Laboratories Systems		n/a		n/a	

* PC = Portable Computer; WS = Work Station; MF = Main Frame
** n/a = not available

2.1.1 Ada programming tools are available with the compiler or as extra options.

All of the vendors provide a minimal set of tools for Ada code development which includes the compiler, editor, debugger, library manager, and runtime environment. Beyond this minimal set, vendors also offer an optimizer, profiler, language-sensitive editor, cross referencer, math library, and simulator (if a cross-compilation system). The major variability of these offerings is whether the tools are bundled in the compiler price or are sold separately. Special tools, such as the language-sensitive editor or profiler, are often part of a package of software engineering tools that can be purchased separately. Bindings to software products such as IBM's database (IMS), graphical data display, and interactive program development facility are provided by several vendors who supply the IBM mainframe Ada environment for business applications. Compiler vendors are beginning to provide bindings to standards such as X-Windows, Structured Query Language (SQL), Programmer's Hierarchical Interactive Graphics (PHIGS), and MOTIF to facilitate development of user interfaces to applications and data.

2.1.2 Ada compilers and tools are hosted on a variety of computer manufacturer equipment and widely available operating systems.

Compilers and environments are offered for portable computers (PCs), workstations, and mainframes that are available on General Services Administration (GSA) schedules, DoD requirements contracts, or are part of the government's installed inventory of general purpose computers. Industry promotion of Motorola and Intel processors has resulted in the availability of compilers that are compatible with PCs and workstations sold under many brand names. The enduring popularity of MS/DOS and UNIX for PCs and workstations is also reflected in the availability of Ada compilers from more than one vendor. For example, four Ada compiler vendors provide compilers for PCs operating under MS/DOS 3.0 or higher while eight vendors provide compilers for UNIX-based operating systems for PCs, workstations (including Reduced Instruction Set Computer (RISC) machines), and mainframe computers. The installed customer base of Digital Equipment Coporation (DEC) in the U. S. is reflected in the number of Ada compiler vendors (six) who provide compilers and tools for DEC's VMS operating system. Three vendors provide compilers and tools for IBM's mainframe operating systems. Two vendors also provide Ada compilers and tools for the Macintosh.

Ada compiler vendors are sensitive to commercial demand for a particular computer and/or operating system. Watching what a compiler vendor drops from his validation schedule is a perceived weakness in commercial demand for a computer system. The cost of obsolescence is unknown; however, it is true that the government must pay higher than typical maintenance fees

for equipment, operating systems, and Ada environments that have been made obsolescent by technology advances. One compiler vendor stated that the maintenance fee is $50,000 per year for a compiler version that is not a current product. It has been estimated by several compiler vendors that they spend approximately $100,000 for each compiler version that successfully completes the Ada validation process every two years. Naturally, vendors intend to maximize their return on investment by targeting growing industry markets. However, government users may not be able to find an Ada compiler for vintage Automated Data Processing Equipment (ADPE) and operating systems without paying a compiler vendor to customize a compiler for them.

2.1.3 There are two major vendor categories: compiler developers and system vendors.

The Ada compiler developers (12 of 28) are those that build Ada compilers as their primary business activity. They build compilers (and tools) for a variety of hosts and target computers with cross-compilation support suitable for real-time and embedded applications. The second category of system vendors (16 of 28) are those that build systems and provide an Ada compiler for their hardware systems. In some cases, the system vendors have obtained a compiler from an Ada compiler developer.

During the survey, one vendor indicated that he believed that almost all the system vendors had their compilers originally developed by one of the "Ada compiler developers." It appears that these developers and at least one of the system vendors (DEC) were the commercial source of the Ada compiler technology. For example, Telesoft does about $1 million in business a year with Cray to maintain the Ada compiler on that machine, though the compiler is marketed through Cray only. Thus, many of the system vendors are actually customers of the compiler developers, and the same compiler can in some cases be obtained from either the system vendor or the developer.

2.1.4 Compiler purchase prices range from $249 for a PC to $400,000 for a multi-user mainframe.

The average price for an environment is $7500 for a network file server. For a PC, there are compilers ranging from $249 to $3000, depending on the number of tools provided and the power of the PC. Discounts of 20-30% are negotiable and at least two vendors provide discounts to academic users. The price of software for mainframes is the highest and also provides a richer environment than is possible for a PC or workstation. Some vendors provide monthly lease options and separate maintenance contracts. A maintenance contract with the compiler vendor

includes software problems/errors fixes and product improvements in successively validated versions of the compiler.

2.1.5 Three Ada vendors support IBM business system environments.

Historically, business systems maintain corporate data bases and financial systems on IBM equipment or Instruction Set Architecture (ISA) compatible computers. The following is a profile of the tools and interface packages available for mainframes and IBM operating systems (i.e., VM/SP, VM/XA, VM/ESA, MVS/SP):

- on-line publication system
- source-code formatter
- library manager
- source-level debugger
- profiler (run-time performance measurements)
- dependency lister
- cross-reference utility
- interface to graphical data display (IBM environment)
- interface to interactive program development facility (IBM environment)
- interface to Information Management System (IMS) (IBM environment)
- standard math functions, including ISO Numerics Working Group (NUMWG)

Information provided by IBM indicates that Ada is a major product strategy and that implementing bindings and protocols to access products implementing other standards is being pushed (e.g., SQL, PHIGS, Portable Operating System Interface for Computing Environments (POSIX)). In addition to IBM, Ada compilers for IBM system environments are provided by Intermetrics and Alsys.

2.1.6 Stability and maturity characterize Ada vendors.

Most of the vendors (20 of 28) have provided validated compilers for more than 5 years. That is a relatively mature group of vendors, given that the Ada language standard dates from 1983. In the past three years, vendors have enlarged the basic compiler tool set to include design, documentation and testing tools and are now offering some bindings to FIPS and industry standards (e.g., X-Windows, MOTIF).

Information concerning the customer base was either not available or companies were unwilling to disclose these numbers. From the information obtained, there appears to be a wide

variance in the size of the customer base. If the vendor (such as Alliant) makes supercomputers, then its customer base may only be a handful. Conversely, a vendor of DOS-based systems (such as Meridian) may claim a customer base of several thousand.

2.1.7 Ada compilers provide interfaces to other languages.

The pre-defined pragma interface is a feature of the Ada language that has caused concern about the uniformity of "openness" among Ada compilers. A review of recent validation documents for the 150 compilers formally tested under Ada Compiler Validation Capability (ACVC) 1.11 shows that almost all compilers support pragma interface to assembler languages of various sorts, C, and Fortran languages. Several provide an interface to Pascal and one to Cobol. The ability to import and export names and objects permits programmers to reuse non-Ada programs and operating systems or run-time services. (See Appendix B for interface names.)

2.1.8 New developments

For a handful of vendors (DEC, IBM, Verdix), there is a movement towards providing an "integrated development environment" that encompasses most phases of the software development life cycle. For the implementation phase, there are tool sets offered with the compiler. For the phases of requirements definition and design, this environment supports various off-the-shelf CASE tools. The objective is to eliminate some of the redundant work in going from requirements to design and from design to implementation. Both DEC and Verdix have either a database or "object repository" that maintains those objects.

2.2 C++ COMPILERS AND TOOLS

Eighteen out of the 22 vendors surveyed market C++ products on the commercial market as well. One of the 22 vendors sells only to other software vendors and 3 companies claimed to not have the C++ products. Table 2 is a summary of the data collected and documented in Appendix C.

Table 2. C++ Product Vendors

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Operating System:																			
DOS	☐ Yes	Yes	Yes	☐ Yes	☐	☐	☐ Yes	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	Yes
Microsoft Windows	☐ Yes	No	☐	☐ Yes	☐	☐	☐ Yes	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	Yes
Unix	☐	☐	☐ Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
VMS	☐	☐	☐	☐	☐ Yes	☐	☐	☐	☐	☐ Yes	Yes	☐	☐	☐	☐ Yes	☐			
Other	☐	☐	☐ Yes	Yes	Yes	☐	☐ Yes	☐	☐	☐	Yes	☐	☐	☐	☐	☐ Yes			
Hardware Platforms:																			
PC/Compatibles	☐ Yes	Yes	Yes	☐ Yes	☐	☐	☐ Yes	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐ Yes		
386/486	☐ Yes	Yes	Yes	Yes	Yes	Yes	☐	☐ Yes	☐	Yes	Yes	Yes	☐	☐	☐	☐ Yes	Yes		
Mac	☐	☐	☐	☐	☐	☐	☐	☐ Yes	☐	☐	☐	☐	☐	☐	☐	☐	☐ Yes		
Workstations (Which)	☐	☐	☐	☐ Yes	☐	☐ Yes	☐	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	☐		
Product features:																			
cfront (AT&T)	Yes	No	No	Yes	No	Yes	Yes	☐ Yes	Yes	☐ Yes	No	☐ Yes	Yes	Yes	No	No			
Class library	☐ Yes	Yes	Yes	Yes	Yes	Yes	☐	☐ No	☐ Yes	No	☐ No	☐ Yes	No	Yes	No	Yes			
IDE	☐ Yes	Yes	Yes	☐ Yes	Yes	☐	☐ No	☐ No	No	☐ Yes	☐ Yes	☐ Yes							
Multiple inheritance	☐ Yes	Yes	Yes	☐ Yes	Yes	☐	☐ Yes	☐ Yes	Yes	☐	☐	☐ No	☐ Yes						
Version control	☐ No	No	No	Yes	Yes	Yes	☐	☐ No	☐ No	Opt	☐	☐	☐	☐	☐ No				
Translator	☐ No	No	Yes	Yes	Yes	Yes	☐	Yes	Yes	☐Also	☐	☐ Yes	☐ Yes	No	☐				
Compiler	☐ Yes	Yes	☐	☐	☐	No	☐	☐ No	☐ Yes	Yes	☐	☐ Yes	☐ Yes	Yes					
Cross compiler	☐ No	Yes	☐	☐	☐	☐	☐	☐	☐ Yes	Yes	☐	☐ No	☐ No	No					
ANSI-C Compatible	☐ Yes	Yes	Yes	Yes	☐ Yes	☐	☐	☐	☐ Yes	Yes	☐	☐	☐	☐ Yes	Yes				
Assembler	☐ Yes	Yes	☐	Yes	☐ No	☐	☐	☐	☐ Yes	Yes	☐	☐	☐	☐ No	No				
Debugger	☐ Yes	Yes	☐	Yes	Yes	Yes	☐ Yes	No	Yes	No	Yes	☐ Yes	Yes	Yes	No	Yes			
Profiler	☐ Yes	Yes	☐	☐	☐ No	☐	☐	☐	☐ No	No	☐	☐	☐	☐	☐ No				

Product information:

Item	Age	Sites	Price
1			
2	Feb 1991	350000	$495
3	May 1990		$150
4	1 Year	Unknown	$250 - $5500
5	3 Years		$200
6	1985	20,000	$499 - $9000
7	2 Years	450	$1195
8	Oct 1990	2000	$2500
9	1 Year	3	$1500 - $19,000
10	3 Years	60	$495
11	1 Year	30,000	With OS
12	18 Mo	2000	$1000 - $20,000
13	June 1988	2000	$1700
14	2 Years	25	$1000
15	Dec 1990	1000	$2696 - $3696
16	6 Mo	Few	$1195
17	Apr 1991	Unknown	$2000
18	---------- Commercial Developer for Resellers ----------		
19	May 1988	200,000	$200 - $1000

2.2.1 C++ vendors provide programming environments composed of products that are differentiated by features and implementation strategy.

The two kinds of development products that accept C++ programs are compilers and translators. For the purpose of this survey, a compiler is a process which accepts a C++ source file as input and produces a file containing an executable or linkable program for some computer. Whereas, a translator is a process which accepts a C++ source file and produces a C language source file that can be input to a C compiler. Vendors provide compilers or translators with or without class libraries and various development tools.

Differences among C++ development products include operating systems and hardware platforms on which they function and the availability of other compatible product features. These features include operating environments and tools as well as language elements. Descriptions of some C++ product features follows.

AT&T provides a product called "cfront" which is a front end or preprocessor for C++ source code. This product has been adopted by some as a standard for the C++ language semantics. While there continues to be no formal C++ standard, several vendors offer products which began as licensed versions of "cfront" or are fully compatible with its semantics. In the survey of C++ vendors, nearly half claim such compatibility.

A feature of the C++ language is its facility for inheritance by an object from a parent object or object class. To augment this facility, vendors may supply libraries of object classes with their products: more than half the vendors surveyed do so. An ANSI committee, seeking to define C++ standards, plans to describe the minimum list of required classes for a class library.

An implementation of a C++ development product generally provides either command line execution or an integrated development environment (IDE) or both. An IDE is a facility to interactively connect a source editor, a compiler or translator, and a runtime environment. Usually the IDE is centered around a user interface such as a windowing capability. From the IDE a developer can maintain the connection among the edit, compile or translate, and execute processes. In other words, a user who is editing the source of a program can tell the environment to compile and execute the program. The IDE will then provide the necessary connections among the source file, the compiler, the runtime environment and any other tools or libraries needed. Most of the C++ vendors claim to have an IDE.

Inheritance of attributes by an object from another object is a feature of object-oriented programming (OOP) and the C++ language. For an object to inherit from a single parent is called single inheritance: to be able to inherit some features from one parent and some from another is called multiple inheritance. Multiple inheritance is more powerful but is considered more difficult for programmers. Users of C++ do not agree on whether multiple inheritance should be included in the language; however, most of the vendors surveyed claim to provide multiple inheritance.

Vendors provide several features which, for purposes of this survey, are called version control. Version control includes the ability to keep track of previous versions of various levels of program elements such as source code, relocatable objects, and executable modules. In the software development area version control includes archiving previous versions, providing release descriptions, controlling which modules need to be compiled before linking (called the "make" feature), etc. Nearly half the vendors surveyed claim to provide some kind of version control.

Cross compilation is a process which executes on one platform producing an executable program that runs on a different target platform. As an example, a Fortran or Pascal compiler running on a DEC VAX computer may produce output which will execute on an IBM PC. Some of the vendors surveyed claimed to provide cross compilers.

C++ compilers accept source program input which adheres to some description of C++ syntax and semantics. A subset of C++ is some version of the C language, but not necessarily ANSI C. A feature of a C++ compiler is its ability to accept and correctly compile any source file which complies with the ANSI C standard. Most vendors surveyed claim to be ANSI C compatible.

A C++ development product may provide the capability to use other languages in several ways. The product may allow instructions in another language, usually assembler, to be included within the source file along with the C++ statements. In C++ this capability is called in-line code. Another way other languages can be used is by providing a way to link the output of another compiler or assembler with the output of the C++ compiler. In the DOS product world it is not uncommon for a vendor to provide such compatibility for some of its own products and some limited number of other products. In addition, many vendors include assemblers with their C++ products to provide programmers the ability to develop their own additional functions. This last case seemed to be most common among the vendors surveyed since about half claim to provide an assembler with their products.

Many vendors provide debugging tools. A profiler is a more advanced debugger which provides a link between an executing program running under debug mode and the source statements from which each instruction came. Most vendors provide some form of debugger; a few claim to have profilers.

2.2.2 The majority of C++ products are for PCs and workstations.

The largest number of product offerings are for IBM PC-compatible systems running DOS and workstations running UNIX. For several other platforms there are individual offerings by platform vendors and by third party suppliers, such as, products that run on VAX/VMS from Digital Equipment Corporation and Bull/GCOS from Honeywell.

The large mainframe manufacturers are not yet offering C++ for their systems. Thus, C++ compilers and translators are only available on small multi-user systems (e.g., AT&T B2).

2.2.3 Most vendors are software distributors who have recently entered the market.

C++ development products, like Ada products, are available from both computer vendors and third party software vendors. The clear majority of currently available products are from software vendors. However, several computer companies have development efforts underway. Some may develop their own products. Others are prone to license existing products from compiler development companies. Most vendors claim to have delivered their C++ development product within the past two years. About half of those have been on the market for a year or less.

2.2.4 Purchase prices range from $150 to $20,000 for PC's and small multi-user systems.

With most software products like compilers, prices vary with the category of platform. In general, products divide along the lines of PC compatibles, workstations, and shared systems such as minicomputers and mainframes. This appears to hold for C++ development products. Products which run under DOS on PC compatibles are typically priced under $500. Workstation products tend to be under $2000. Some products for small, multi-user systems are priced up to $20,000. These prices tend to be in line with prices of other language compilers for the same platforms.

2.2.5 Efforts are being organized to develop a C++ standard for the language and the class library.

The companies are currently working on establishing ANSI and ISO standards for C++ are listed in Appendix D. These standards will be in two areas, the language and the class library. Although the participants represent many companies and the committees are currently active, adoption of a standard is not expected in the immediate future. At present the committee appears to have the beginnings of a working document for the language but may not have begun to construct one for the library.

2.2.6 Vendors of low cost C++ development products have a relatively large customer base.

Claims of installed base vary from very few to a high of around 350,000. These figures were not available from most vendors. The ones that were seem to be estimates and may not be accurate. There is, however, a trend which tends to indicate substantial sales of at least two products for DOS systems, Zortec C++ with 200,000 copies and Borland C++ with 350,000 copies, as well as some considerable activity in the workstation market. The estimated installed base figures show both interest by the development community and enough sales to indicate acceptance of the products. The apparent flurry of computer companies to provide C++ products for their systems indicates some acceptance of C++ as a programming language.

2.2.7 C++ products provide interfaces to other software implemented in C or assembler.

External interfaces to other software products are available from some vendors. In particular, vendors tend to provide access to an assembler and in some cases other language interfaces. Other accesses are available to data base management systems and user interfaces like X-windows. It appears that almost any product available to a vendor's C language product is also available to its C++ product.

2.2.8 New developments target mainframe hardware systems.

Although C++ development products are now on the market for PCs, workstations and shared systems, many more are on the way. As with most previous languages, computer vendors are anxious to provide C++ products which will take advantage of their own platform configura' ns. C++ projects are now underway at IBM, Honeywell, Hewlett-Packard, and many

other companies. Expectations are that the language will be available for most major platforms in the United States.

2.3 AVAILABILITY OF Ada AND C++ TRAINING AND EDUCATION

In preparing this analysis, the following sources were used:

- Ada Software Engineering Education and Training (ASEET) Data Base
- The Journal for Object-Oriented Programming
- Contacts within the academic and DoD areas

Appendix E includes the updated ASEET database and sources for C++ training. The database includes the types of courses taught, and when available the cost and a point of contact.

2.3.1 There are more sources of training and/or education for Ada usage than for C++.

Since 1983, when Ada was adopted as an ANSI standard, the AJPO has emphasized the need for Ada education and training within the DoD, industry, and academia. One of the first initiatives was to encourage the creation of numerous Ada courses by both government and commercial organizations. Today, Ada training is available throughout the country, at least one university in every state teaching Ada. All three military academies offer Ada in their computer courses. We were not able to find any DoD facilities that taught C++; however, we have been told that the Naval Postgraduate School does use C++. In fact, most said they used Ada when teaching object-oriented design. The results of the survey on C++ in the universities is incomplete since most of the time was spent gathering information from C++ training vendors. Ada compiler vendors provide training for system designers and programmers in a classroom setting or as self-study books and software.

Recent programmer interest in C++ parallels some of the developments of object-oriented system design methods and object-oriented data base products. Object-oriented programming (OOP) is an engineering technique used to solve problems that can be expressed in terms of objects, classes of objects, inherited properties, and state data. The superiority of OOP for all types of systems is yet to be demonstrated but it is a convenient solution when the environment is based upon UNIX and C. On the other hand, Ada is being used by computer scientists and programmers to implement systems that require solutions to a range of problems (i.e., temporal, function, and structure). See Appendix F for discussion of design paradigm needs.

Table 3. Sites Teaching Ada and C++ Listed by State

State	Ada Univ	Ada-DoD	Ada Commercial	C++ Univ	C++ Commercial
Alabama	7	-	-	-	-
Alaska	2	-	-	-	-
Arizona	3	-	-	-	-
Arkansas		-	-	-	-
California	19	-	2	1	6
Colorado	5	1	-	-	1
Connecticut	6	-	-	-	-
Delaware	-	-	-	-	-
Florida	11	-	1	1	-
Georgia	7	1	-	-	-
Hawaii	2	-	-	-	-
Idaho	-	-	-	-	-
Illinois	7	1	-	-	-
Indiana	6	-	1	-	-
Iowa	4	-	-	-	-
Kansas	4	-	-	-	-
Kentucky	4	-	-	-	-
Louisiana	3	-	-	-	-
Maine	1	-	-	-	-
Maryland	6	1	4	1	-
Massachusetts	5	-	2	-	5
Michigan	7	-	-	-	2
Minnesota	2	-	-	-	-
Mississippi	4	1	-	-	-
Missouri	6	-	-	-	-
Montana	-	-	-	-	-
Nebraska	-	1	-	-	-
Nevada	-	-	-	-	-
New Hampshire	-	-	-	-	-
New Jersey	5	-	1	-	3
New Mexico	4	-	-	-	-
New York	11	1	-	-	2
North Carolina	3	-	-	-	-
North Dakota	3	-	-	-	-
Ohio	10	2	-	-	1
Oklahoma	6	-	-	-	-
Oregon	-	-	-	-	1
Pennsylvania	12	-	-	-	1
Rhode Island	1	-	-	-	-

- indicates unknown; note results on C++ in Universities is incomplete due to time constraints.

Table 3. Sites Teaching Ada and C++ Listed by State (Cont'd)

State	Ada Univ	Ada-DoD	Ada Commercial	C++ Univ	C++ Commercial
South Carolina	1	-	-	-	-
Tennessee	7	-	-	-	-
Texas	14	1	2	-	2
Utah	4	-	-	-	-
Vermont	1	-	-	-	-
Virginia	6	2	2	-	-
Washington	3	-	-	-	-
West Virginia	7	-	-	-	-
Wisconsin	2	-	-	-	-
Wyoming	-	-	-	-	-
Washington, D.C.	5	1	1	1	-

- indicates unknown; note results on C++ in Universities is incomplete due to time constraints.

2.3.2 There is some disparity between ada and C++ training providers.

In addition to university and compiler vendor courses, there are several Ada education and training vendors who specialize in teaching software engineering with Ada. The courses vary from two-to-four hour introduction courses for managers to a one-or-two week long intensive Ada programming course. Some vendors charge a flat fee ($10,000) and limit the course to 12-20 people, while others charge per student ($1100/each). These courses may be taught either at the customer's site or at a public seminar or a the vendor's site. Most of the hands-on workshops do limit the number of participants, while a course such as the executive overview is left open.

Most of the listings for C++ were independent training vendors. Many are small consulting firms that offer training only on the customer's site. The average course is five days long and includes some type of hands-on lab. Most claim to provide hands-on for any type of platform for which C++ products are sold, although one firm stated that they only teach C++ on the Macintosh (Arbor Intelligent Systems, Inc.). The cost of these courses varies and does not include the travel and living expenses of the instructor. The student cost ranges from $695/each for a two-day course to $1,200/each for a five-day course to a set price of $9,900 for a four-day course with a maximum number of 20 students. Vendors always indicated that they could develop or customize a C++ for their customer if needed. Most of the companies are small (i.e., two-five people) and some of the vendors listed in the November-December 1990 issue of Journal of Object-Oriented Programming appear to have already gone out of business.

2.4 STATUS OF CASE TOOLS.

From a list of 200 commercial vendors of products, informally known as CASE tools, data was collected on 155 with 44 being classified by our definition as CASE tools. Tools that support particular design or analysis methodologies are not usually influenced by the choice of implementation language, but the majority of these CASE tools is not completely language independent because most generate code. Appendix G provides ten tables that consolidate descriptive information about CASE tools. Appendix H documents, in more detail, the information collected on the 44 CASE tools. The following findings indicate the status of CASE tools.

2.4.1 Structured Analysis (SA) and Structured Design (SD) are the most widely supported software development methods, although increasing support for object-oriented approaches is evident.

Methods for software design and then analysis fall into two groups: process-oriented methods to support the development of information systems, and behavior- or state-oriented methods for process-control systems. This distinction has blurred as the most popular, process-oriented methods, SA and SD, have been augmented with techniques for expressing behavior. In the last few years, an object orientation to software development has evolved.

Over 65% of CASE tools provide support for SA and SD and three quarters of these include the augmentations for expressing behavior. Over one quarter support OOD, and a quarter of these also support OOA. Nearly a fifth support both SA/SD and object-oriented approaches. More details on the method support offered by particular CASE tools are presented in Appendix G, Table 2. Information on operating environments, breadth of use, report generation, adaptability, etc., can be found in Appendix G, Table 3.

2.4.2 CASE tools for the development of information systems differ from those that support the development of other types of software.

Roughly half as many CASE tools are intended for the development of information systems as for other types of software systems (for example, real-time and process control systems). The distinction between these two groups of CASE tools is evidenced in several ways. For example, only those CASE tools intended for the development of information systems typically support data base design and, in the few cases where prototyping is provided, it supports user interface (forms and screen) design. Again, only information system-oriented CASE tools typically support business analysis and planning activities. On the other hand, CASE tools in the second group are

more likely to support simulation and requirements tracing activities and to provide the users with a selection of development methods.

2.4.3 Support for CASE tool customization is limited.

Over 65% of CASE tools provide free-form or customizable graphics. Tailoring of the underlying development methods is much less frequent and generally requires the user to develop new code. Three vendors market tools that support rule-based customization of their CASE tool, two offer tools specifically intended to the user screens or menus, and one markets a meta-CASE tool that can be used to develop CASE software. See Table 4, Appendix G.

2.4.4 The majority of CASE tools support source code generation.

Virtually all CASE tools generate some type of code, though those that support the development of information systems may only generate data handling or user interface code. The language(s) generated varies, depending on the type of CASE tool considered:

- CASE tools supporting the development of information systems either include tool components that generate code or link with independent application generators for this function. In the first case, code generators typically produce Cobol and C, and the introduction of Ada and C++ has had little impact. In the second case, application generators (see Table 5, Appendix G) are traditionally devoted to the production of Cobol; although no application generators that support Ada have been identified, some support for C++ is evident.

- Code generation for other types of software systems (e.g., process control, embedded, real time) favors (in descending order) C, Ada, Pascal, Fortran, C++, PL/I, and Jovial. The entire source code is not necessarily generated and some tools provide user-customizable templates that govern this partial generation. Support for C++ is one of the most frequently cited planned tool enhancements and C++ is expected to follow Ada in popularity within the next 18 months.

2.4.5 C is being used by CASE tool developers.

In terms of tool implementation language, the majority of CASE tools are implemented in C. However, over 20% of the vendors already have, or plan to, reimplement their products in C++. Fewer tools have been developed or reimplemented in Ada. Reasons for using C or C++ for tools may be based on economics. For example, C compilers are relatively inexpensive (no validation costs, smaller language, etc.) and existing C interfaces to windows and UNIX facilities reduces effort.

2.4.6 Workstations are the favored hardware platform.

The majority of CASE tools operate on workstations and are capable of supporting multiple concurrent users over a network. Roughly two thirds are also supported on PCs, and roughly one third are also supported on mainframes. PCs and mainframes are rarely the only operating platform. The dependence of these tools on the underlying programming support environment is restricted to a language compiler and related language-sensitive tools.

2.4.7 CASE vendors say they support open systems and interoperability.

Roughly half of the CASE tool vendors state that their tools exist in an open environment. Many vendors further support interoperability by conforming with the de facto industry standard X-Windows. Support for the CASE Data Interchange Format (CDIF) (Electronics Industry Association) standard is less prevalent but increasingly apparent.

2.4.8 CASE vendors offer relatively mature products.

While six tools are major extensions or reworks of products developed in the late 1970s and early 1980s, roughly half of the currently marketed CASE tools were introduced between 1984 and 1987. Tools continue to be introduced. The initial focus on support for development of information systems has gradually changed and the majority of recent offerings support the development of real-time software systems.

Some vendors report the number of licenses they have sold, whereas others measure usage in terms of the number of installations. Until recently, information system-oriented CASE tools have been the most widely used, with installations and licenses numbering in the thousands. Over the last few years, increased awareness of software engineering and, perhaps, better marketing of

more likely to support simulation and requirements tracing activities and to provide the users with a selection of development methods.

2.4.3 Support for CASE tool customization is limited.

Over 65% of CASE tools provide free-form or customizable graphics. Tailoring of the underlying development methods is much less frequent and generally requires the user to develop new code. Three vendors market tools that support rule-based customization of their CASE tool, two offer tools specifically intended to the user screens or menus, and one markets a meta-CASE tool that can be used to develop CASE software. See Table 4, Appendix G.

2.4.4 The majority of CASE tools support source code generation.

Virtually all CASE tools generate some type of code, though those that support the development of information systems may only generate data handling or user interface code. The language(s) generated varies, depending on the type of CASE tool considered:

- CASE tools supporting the development of information systems either include tool components that generate code or link with independent application generators for this function. In the first case, code generators typically produce Cobol and C, and the introduction of Ada and C++ has had little impact. In the second case, application generators (see Table 5, Appendix G) are traditionally devoted to the production of Cobol; although no application generators that support Ada have been identified, some support for C++ is evident.

- Code generation for other types of software systems (e.g., process control, embedded, real time) favors (in descending order) C, Ada, Pascal, Fortran, C++, PL/I, and Jovial. The entire source code is not necessarily generated and some tools provide user-customizable templates that govern this partial generation. Support for C++ is one of the most frequently cited planned tool enhancements and C++ is expected to follow Ada in popularity within the next 18 months.

2.4.5 C is being used by CASE tool developers.

In terms of tool implementation language, the majority of CASE tools are implemented in C. However, over 20% of the vendors already have, or plan to, reimplement their products in C++. Fewer tools have been developed or reimplemented in Ada. Reasons for using C or C++ for tools may be based on economics. For example, C compilers are relatively inexpensive (no validation costs, smaller language, etc.) and existing C interfaces to windows and UNIX facilities reduces effort.

2.4.6 Workstations are the favored hardware platform.

The majority of CASE tools operate on workstations and are capable of supporting multiple concurrent users over a network. Roughly two thirds are also supported on PCs, and roughly one third are also supported on mainframes. PCs and mainframes are rarely the only operating platform. The dependence of these tools on the underlying programming support environment is restricted to a language compiler and related language-sensitive tools.

2.4.7 CASE vendors say they support open systems and interoperability.

Roughly half of the CASE tool vendors state that their tools exist in an open environment. Many vendors further support interoperability by conforming with the de facto industry standard X-Windows. Support for the CASE Data Interchange Format (CDIF) (Electronics Industry Association) standard is less prevalent but increasingly apparent.

2.4.8 CASE vendors offer relatively mature products.

While six tools are major extensions or reworks of products developed in the late 1970s and early 1980s, roughly half of the currently marketed CASE tools were introduced between 1984 and 1987. Tools continue to be introduced. The initial focus on support for development of information systems has gradually changed and the majority of recent offerings support the development of real-time software systems.

Some vendors report the number of licenses they have sold, whereas others measure usage in terms of the number of installations. Until recently, information system-oriented CASE tools have been the most widely used, with installations and licenses numbering in the thousands. Over the last few years, increased awareness of software engineering and, perhaps, better marketing of

CASE products have led to wide usage of CASE tools supporting the development of other types of software systems. Table 1 (Appendix G) lists product introduction date and estimated customer base.

2.4.9 Future Trends

Bridges between CASE tools are increasingly used to extend the scope of software development activities supported by particular tools. Roughly one third of the CASE tools have vendor-supported bridges that exploit the capabilities of other CASE tools. While the majority of current bridges only support a one-way transition between tools, some bi-directional bridges are beginning to appear. In addition to allowing the use of specialized tools as required, these bridges can facilitate the reuse of software products developed using different tools. Table 6 (Appendix G) identifies the available bridges.

2.4.10 CASE tools continually increase their coverage of software development activities.

Early CASE tools focused on software analysis and design activities. Initial extensions focused on earlier development activities and led to the provision of requirements traceability capabilities. Roughly half the CASE tools provide this capability, the majority of which do so as an integral part of the tool. Another area of early extension was the provision of system specification and simulation capabilities. Roughly one third of the tools support system simulation, usually via a separately purchasable option.

In the last few years, vendors have been introducing support for reverse engineering to facilitate software maintenance and, to some extent, reuse. Roughly half the CASE tools have this capability, and several more expect it within the next 18 months. Although usually provided as an integral part of the CASE tool, reverse engineering tools are also available as separately purchasable options and as stand-alone tools. Roughly equivalent numbers of tools are available for reverse engineering of Ada and C++. See Table 7 (Appendix G).

A few CASE vendors are starting to support software testing. This capability is generally provided through separately purchasable options, primarily for Ada and C code. The stand-alone testing tools identified. See Table 8 (Appendix G), predominantly support Ada, although one vendor does offer support for C++.

2.4.11 CASE vendors talk about migration to repositories.

Early CASE tools used a data dictionary to store definitions of the various data flows, processes, data stores, etc., specified as part of software analysis and design activities. A repository, in simple terms, is a central database that contains all information pertaining to a development effort. It provides better support for information sharing among team members, tool integration, and new development paradigms such as Boehm's risk-driven approach. An object-oriented repository, in particular, provides the flexibility to facilitate CASE customization and extension. All CASE tools introduced in the last couple of years employ repositories. A significant number of early tools have recently switched to a repository.

2.4.12 Integration frameworks are increasingly preferred as a mechanism for integrating project management and similar tools with CASE tools.

Repositories have led to the development of integration frameworks that provide a consolidation of the underlying information architecture to offer a disciplined approach to tool integration. They allow CASE tools to be integrated into a base set of capabilities supporting, for example, resource management, change management, and access to multiple databases. Identified repositories are listed in Table 9 (Appendix G).

IBM's announced integration framework, AD/Cycle, is expected to have a a significant impact on CASE tool evolution, and the majority of vendors plan to ensure compatibility with AD/Cycle as it becomes available.

2.5 CONCLUSIONS

Conclusions based on the limited scope of the survey and analysis of findings are:

- Ada compilers are available for PCs, workstations, and mainframes, including the mainframe computers most often used for large business applications. C++ products are available for PCs and some multi-user engineering workstations but not in general for mainframes.

- There is stability and maturity among Ada compiler vendors with the majority of Ada companies providing validated compilers for five or more years. The majority of C++ vendors have entered the market during the last two years although many have provided C compilers for many years.

- There is considerable variability among C++ products in the language features they support, the libraries provided, and strategy for language support. The standardization effort for C++ and libraries is just beginning. The Ada 9X standard with its object-oriented programming support is expected to be adopted by ANSI and ISO by the time the C++ standardization effort results in an adopted standard.

- The wide availability of Ada training and education reflects DoD efforts to promote Ada as a way to teach software engineering methods. Currently, Ada is being taught and used in university computer science departments. Most Ada compiler vendors are a source of training materials and instruction while C++ training and education is in limited supply.

- CASE tools exist to support both Ada and C++. Structured analysis and structured design are the most widely supported development methods but object-oriented design and analysis are just entering the picture. CASE tools marketed for business applications do not contain features such as requirements tracing and simulation and choices among design paradigms. Future plans among CASE and compiler vendors call for an integration framework so that tools can be distributed as commercial-off-the-shelf products for a variety of platforms.

3. ACRONYMNS

4GL	Fourth Generation Language
ACM	Association for Computer Machinery
ACVC	Ada Compiler Validation Capability
ADPE	Automated Data Processing Equipment
AI	Artificial Intelligence
AJPO	Ada Joint Program Office
ANSI	American National Standards Institute
ASEET	Ada Software Engineering Education and Training
C3I	Command, Control, Communications, and Intelligence
CAD	Computer-aided Design
CASE	Computer-aided Software Engineering
CDIF	CASE Data Interchange Format (Electronics Industry Association)
COTS	Commercial Off-the-Shelf
DEC	Digital Equipment Corporation
DoD	Department of Defense
FIPS	Federal Information Processing Standard
GSA	General Services Administration
IDA	Institute for Defense Analyses
IDE	Integrated Development Environment
IMS	Information Management System
ISA	Instruction Set Architecture
ISO	International Standards Organization
MF	Main Frame
n/a	not available
NIST	National Institute of Standards and Technology
NUMWIG	Numerics Working Group, International Standards Organization
OOA	Object-oriented Analysis
OOD	Object-oriented Design
OOP	Object-oriented Programming
PC	Personal Computer
PHIGS	Programmer's Hierarchical Interactive Graphics (ANSI, FIPS, ISO)
POSIX	Portable Operating System Interface for Computing Environments
R&D	Research and Development
RISC	Reduced Instruction Set Computer
SA	Structured Analysis
SD	Structured Design
SDIO	Strategic Defense Initiative Office
SIGAda	Special Interest Group, Ada
SQL	Structured Query Language
WS	Work Station

Appendix E - Status of Training and Education

C++ Education and Training

C++ Training

	Provider	Courses Provided	Address	City	State	Zip Code	Point of Contact	Phone Number
1	Mark V Systems, Ltd	In-house and hands-on training in C++	16400 Ventura Blvd. Ste 303	Encino	California	91436	Mo Bjornestad	818-995-7671
2	ParcPlace Systems, Inc.	Intro to Object-Oriented Concepts and C++ - 5 day course taught in-house ($1400/person) or at the customer site ($15000 for up to 10 students, additional $1200 each with max of 20)	1550 Plymouth Street	Mountain View	California	94043	Debbie Hudson	415-691-6755
3	Santa Clara University	C++		Santa Clara	California			
4	Rational Consulting	1-Design workshop - 6 session (5 2-day sessions & wrap-up session) for 12 students ($30,000) 2- Intro course - 1 week ($12,500) 3- Adv course - 1 week ($12500)	3320 Scott Boulevard	Santa Clara	California	95054	Brock Peterson	408-496-3684
5	Hewlett-Packard	Public seminars, in-house training and hands-on training in C++	1266 Kifer Road	Sunnyvale	California	94086	Jagi Shahani	408-746-5780
6	Versant Object Technology	Public seminars, in-house training and hands-on training in C++	4500 Bohannon Drive	Menlo Park	California		Sandra Philpott	415-325-2380
7	Fowler Software Design	In-house training and hands-on training in C++	P.O. Box 365	Eldorado Springs	Colorado	80025	Jan Fowler	303-494-5755
8	Florida Institute of Technology	C++ courses available in 1992	Department of CS 150 West University Blvd	Melbourne	Florida	32901	Charles Engle	407-768-8000

C++ Training

	Provider	Courses Provided	Address	City	State	Zip Code	Point of Contact	Phone Number
9	Capital College	1-Object Oriented Programming 2-Advanced C course		Laurel	Maryland		Jack Bieler Harry Harrison	703-941-6888 301-953-0060
10	Object Design Support Services (training facil. in Mass and CA)	1-Intro to OOP and C++ (2 days, $695/person with max of 10 people) 2-Mastering C++ (2 days, $695/person with max of 10 people) All course available on-site or at training centers	One New England Exec Park	Burlington	Massachu-setts	1803	Cheryl Flust	617-270-9797 ext. 132
11	Technology Exchange Co./ Addison-Wesley	Public seminars, in-house training and hands-on training in C++	Rte 128	Reading	Massachu-setts	1867		800-333-0088 617-944-3700
12	MacGregor Group	In-house C++ training	34 Summit Road	Wellesley	Massachus-setts	2181	Steven Levy	Phone number changed-now unlisted
13	Empathy (All training at customer site)	1-OOP using C++ (4 days, max 20 people, $9,900) 2-Advanced C++ and Design Techniques (4 days, max 20 people, $9,900) 3- OOD (3 days, max 20 people, $7,900)	P.O. Box 632	Cambridge	Massachuse tts	2142	Rich Mitchell	617-787-3089
14	Semaphore Training	1- Introduction to C++ and OOD (5 days at $11,495 for 15 people - incl lab) 2-Efficient impl of OOD in C++ 3-Advanced C++ & OOD (4 days at $10,995 for 15 people - incl. lab)	800 Turnpike Street, Suite 200	North Andover	Massachuse tts	1845	Ted Cannie	508-794-3366
15	Object Resources	In process of developing C++ courses and will customize courses for the customer as needed	39500 14 Mile Road, Suite 206	Walled Lake	MI	48088	John Kills	313-661-5343
16	Arbor Intelligent Systems, Inc.	C++ Training on the MAC - Introduction and Advance courses that run 5 days for approximately $1400/person. Will do on site-training and develop C++ course for platform other than MAC		Ann Arbor	Michigan		Ron Suarez	313-996-4238

C++ Training

	Provider	Courses Provided	Address	City	State	Zip Code	Point of Contact	Phone Number
17	Invention Software	Public seminars, In-house training and hands-on training in C++	P.O. Box 3168	Ann Arbor	Michigan	48106	Mike Davidson	Phone disconnected
18	EDP Consultants, Inc.	In-house and hands-on training in C++	77 Meredith Road	Colonia	New Jersey	7067	Richard Estock	
19	Institute for Zero Defect Software	Both courses taught at customer site: 1-C++ Prog for C Programmers (5 day hands-on workshop, $8500)(16 people) 2-OODesign for C++ (5 day hands-on workshop, $8500) (max 16 people)	85 Poplar Drive	Sterling	New Jersey	7980	Hwe-Chu Tu	201-604-8701
20	DeerWorks	1-Intro to C++ and OOD - 4 days hands-on course ($2,000/day at customer site, $350/person public seminar) 2-OOD mapped into C++ - 3 day course with case studies (same price)	411 Valentine Street	Highland Park	New Jesey	8904	Tsvi Bar-David	201-985-7427
21	Center for Object-Oriented Training	1-Stepping Up from C to C++ 2-Advanced programming in C++	588 Broadway, #604	New York	New York	10012	Melanie Younossi	212-274-0640
22	ImageSoft	5 day lab-intensive C++ course taught at the New York Office or on-site. $1750/student with max of 15 people. Fee includes notes, and 2 textbooks	2 Haven Ave	Port Washington	New York	11050	Ramana Murthy	516-767-2233 800-245-8840
23	Saks & Associates	In-house training and hands-on training in C++	287 W. McCreight Avenue	Springfield	Ohio	45504	Dan Saks	513-324-3601
24	Quality Software Engineering	1-C++: Programming, Paradigms and Techniques (4 days, hands-on lab) 2-Structured Approach to OOD (4 days, hands-on lab) All courses $2,000/day for up to 20 students	P.O. Box 303	Beaverton	Oregon	97075	Paul Blattner	503-538-8256

	Provider	Courses Provided	Address	City	State	Zip Code	Point of Contact	Phone Number
25	Instantiations	On-site courses that consist of 2 days of OOD and 3 days of programming using C++. If hands-on course, is limited to 14 people, if not, course limited to 20. $10,000 + instructor's expenses		Portland	Oregon		Leslie Menashe	503-242-0725
26	Revolution 2	In-house training and hands-on training in C++	P.O. Box 760	Kenneth Square	Pennsyl-vania	19348	Bruck Eckel	Phone disconnected
27	Object International, Inc.	Public seminars, in-house training and hands-on training in C++	9430 Research Blvd. IV-400	Austin	Texas	78759	Sylvia Owens	512-343-4549
28	Genesis Development Corp.	Public seminars and in-house training in C++	1303 Columbia Dr., Ste 209	Richardson	Texas	75081	Susan Estes	214-644-8559
29	George Washington University	1-Software Engineering (in Fall 91) 2-Softare Engineering - graduate level	Computer Science Department		Washington D.C.		Shmuel Rotenstreich	202-994-5252

Commerically Available Ada Courses

1	Telesoft	Introducton to Ada - comprehensive series of new Ada training targeting large-scale and embedded real-time programming issues.	5959 Cornerstone Court West	San Diego	CA	92121	Jeff Kelley	619-457-2700
2	Systems Engineering Research Corporation	Advanced Ada Topics Series - includes several Ada topics and language issues	415 Clyde Avenue Suite D	Mountain View	California	94043		415-962-8092
3	Ada Technology Group	Ada Software Engineering for Defense Systems - 10 day hands on program on Ada	1900 L. Street, Suite 500	Washington	D.C.	20036	Walter Rollins	202-296-1321
4	Integrated Software	Ada For Real-Time Systems - a two day seminar that addresses the practical considerations of real-time pro-gramming in Ada. ($4000 - may include up to 25 attendees)	P.O. Box 060295	Palm Bay	Florida	32906	Marilyn Pelo	407-984-1986
5	Advanced Software Technology Specialists	Ada Design and Coding: 1- 5 days ($11000) [All courses are taught at the customer's site]	4 Lutz Road	Ossian	Indiana	46777	Donald G. Firesmith	219-639-6305
6	Advanced Software Technology Specialists	Ada Design and Coding: 2 - 5 days ($1100)	4 Lutz Road	Ossian	Indiana	46777	Donald G. Firesmith	219-639-6305
7	Advanced Software Technology Specialists	Ada Design and Coding: 3 - 5 days ($1100)	4 Lutz Road	Ossian	Indiana	46777	Donald G. Firesmith	219-639-6305
8	Advanced Software Technology Specialists	Ada Project Management - 4 days ($12500)	4 Lutz Road	Ossian	Indiana	46777	Donald G. Firesmith	219-639-6305
9	Advanced Software Technology Specialists	Ada Technology Transition - An Executive Overview - 1 day ($3000)	4 Lutz Road	Ossian	Indiana	46777	Donald G. Firesmith	219-639-6305

Commerically Available Ada Courses

10	Advanced Software Technology Specialists	Ada Testing, Quality Assurance and IV&V - 3 days ($7000)	4 Lutz Road	Ossian	Indiana	46777	Donald G. Firesmith	219-639-6305
11	Advanced Software Technology Specialists	Ada Tools and Environments - 2 days ($5500)	4 Lutz Road	Ossian	Indiana	46777	Donald G. Firesmith	219-639-6305
12	Advanced Software Technology Specialists	DoD-STD-2167A and Tailoring for Ada Projects - 2 days ($5500)	4 Lutz Road	Ossian	Indiana	46777	Donald G. Firesmith	219-639-6305
13	Advanced Software Technology Specialists	Object-Oriented Development in Ada - 5 days ($12500)	4 Lutz Road	Ossian	Indiana	46777	Donald G. Firesmith	219-639-6305
14	Advanced Software Technology Specialists	Object-Oriented Requirements Analysis - 1 day ($3000)	4 Lutz Road	Ossian	Indiana	46777	Donald G. Firesmith	219-639-6305
15	Advanced Software Technology Specialists	Software Economics in Ada - 2 days ($5500)	4 Lutz Road	Ossian	Indiana	46777	Donald G. Firesmith	219-639-6305
16	Advanced Software Technology Specialists	Software Engineering and Methods in Ada - 2 days ($5500)	4 Lutz Road	Ossian	Indiana	46777	Donald G. Firesmith	219-639-6305
17	Fastrak Training, Inc.	Ada - Management Perspective - a 3 day seminar for managers and senior technical staff - available upon request	9175 Guilford Road, Suite 300	Columbia	Maryland	21046	Abby Eden	301-498-5601
18	Fastrak Training, Inc.	Ada Cost Modeling - one day seminar designed for technical managers and staff responsible for estiamting size, effort and schedule on Ada software development projects	9175 Guilford Road, Suite 300	Columbia	Maryland	21046	Abby Eden	301-498-5601

Commerically Available Ada Courses

19	Fastrak Training, Inc.	Advanced Ada Programming - 5 day hands on workshop created for software engineers with previous Ada experience and/or training - available upon request	9175 Guilford Road, Suite 300	Columbia	Maryland	21046	Abby Eden	301-498-5601
20	Fastrak Training, Inc.	Designing Ada Software - 4 day workshop for programmers and software designers to introduce a methodical design process for OOD in a 4-step approach	9175 Guilford Road, Suite 300	Columbia	Maryland	21046	Abby Eden	301-498-5601
21	Fastrak Training, Inc.	Evaluating Ada Code - 5 day seminar designed for government personnel and IV&V contractors who read and evaluate compiled Ada PDL or code - available upon request	9175 Guilford Road, Suite 300	Columbia	Maryland	21046	Abby Eden	301-498-5601
22	Fastrak Training, Inc.	Introduction to Ada Programming - 5 day hands-on workshop for software engineers with no prior experience programming in Ada - available upon request	9175 Guilford Road, Suite 300	Columbia	Maryland	21046	Abby Eden	301-498-5601
23	Fastrak Training, Inc.	Software Engineering in the Ada Environment - 4 day seminar for technical staff participating in Ada software development and maintenance for large systems	9175 Guilford Road, Suite 300	Columbia	Maryland	21046	Abby Eden	301-498-5601
24	EVB Software Engineering, Inc.	Advanced Ada Programming Workshop - 5 day seminar designed for programmers, analysts, and managers ($10,000)	5320 Spectrum Drive	Frederick	Maryland	21701	Jennifer Lott Ann Hawkins	301-695-6960
25	EVB Software Engineering, Inc.	Creating Reusable Ada Software - 5 day seminar for technical software professionals with a reading knowledge of Ada ($10,000)	5320 Spectrum Drive	Frederick	Maryland	21701	Jennifer Lott Ann Hawkins	301-695-6960
26	EVB Software Engineering, Inc.	Fundamental Object Oriented Concepts - 5 day seminar for those interested in an OOD approacd to Ada software developing ($10,000)	5320 Spectrum Drive	Frederick	Maryland	21701	Jennifer Lott Ann Hawkins	301-695-6960
27	EVB Software Engineering, Inc.	Object Oriented Development for Ada Software - 5 day seminar intended for software engineers and technical managers using OOD as a methodology for Ada development ($10,000)	5320 Spectrum Drive	Frederick	Maryland	21701	Jennifer Lott Ann Hawkins	301-695-6960

28	EVB Software Engineering, Inc.	Testing Ada Software - 3 day seminar for programmers, analysts and managers who are interested in various software testing techniques and strategies ($6000)	5320 Spectrum Drive	Frederick	Maryland	21701	Jennifer Lott Ann Hawkins	301-695-6960
29	IIT Research Institute	Ada For Managers - 4 hour course - explore philosophy of Ada and maximizing benefits ($800 - up to 10 students)	4600 Forbes Blvd.	Lanham	Maryland	20706	Ms. Mary Armstrong	301-731-8894
30	IIT Research Institute	Ada For Software Engineers - 20 hour discussion and 20 hour hands-on. Provides a summary of syntax and how best to utilize the Ada features. ($8000 - up to 10 students)	4600 Forbes Blvd.	Lanham	Maryland	20706	Ms. Mary Armstrong	301-731-8894
31	IIT Research Institute	Executive Overview of Ada - 2 hour discussion of the types of contracts an Ada software lab can expect to acquire and the up-front investments that must be made ($400)	4600 Forbes Blvd.	Lanham	Maryland	20706	Ms. Mary Armstrong	301-731-8894
32	IIT Research Institute	Object-Oriented Development in Ada - 10 hours discussion and 30 hours hands-on. Introduce the software engineer to state-of-the-art software development theory ($8000 - up to 10 students)	4600 Forbes Blvd.	Lanham	Maryland	20706	Ms. Mary Armstrong	301-731-8894
33	IMR Systems Corp	Ada Training Laboratory - training focuses on software development and compliance with DoD standards - lab has a validated Ada compiler and Ada development environment	11400 Rockville Pike, Suite 501	Rockville	Maryland	20852	Mr. Will Spencer	301-468-1160
34	Alsys Inc. - courses taught at customer site for up to 20 students	Ada Software Engineering Design Methodologies - 5 day seminar for those who need to understand how Ada cna best be used and hwo to establish a coherent Ada-based methodology	14 Main Street	Waltham	Massachusetts	2154	Dr. Benjamin M. Brosgol	617-890-0030
35	Alsys Inc. - courses taught at customer site for up to 20 students	Ada Software for Managers - 3 day seminar on the management issues of Ada use for large systems development	14 Main Street	Waltham	Massachusetts	2154	Dr. Benjamin M. Brosgol	617-890-0030
36	Alsys Inc. - courses taught at customer site for up to 20 student	Ada Technology Issues - 2 1/2 day seminar for the computer executive who needs to know what the advantages, risks, etc. are in choosing Ada for software development	14 Main Street	Waltham	Massachusetts	2154	Dr. Benjamin M. Brosgol	617-890-0030

37	Alsys Inc. - courses taught at customer site for up to 20 students	Advanced Ada Topics and Real-Time Systems in Ada - 7 day seminar for those who need to udnerstand possible peculiarities of Ada real-time systems.	14 Main Street	Waltham	Massachusetts	2154	Dr. Benjamin M. Brosgol	617-890-0030
38	Alsys Inc. - courses taught at customer site for up to 20 students	Intermediate Ada - 5 day seminar for those who need to know the strengths and weaknesses of the language in order to design and develop Ada programs.	14 Main Street	Waltham	Massachusetts	2154	Dr. Benjamin M. Brosgol	617-890-0030
39	Alsys Inc. - courses taught at customer site for up to 20 students	Introduction to the Ada Language - one day seminar for software project managers or others who wish a broad view of Ada and its implications.	14 Main Street	Waltham	Massachusetts	2154	Dr. Benjamin M. Brosgol	617-890-0030
40	Alsys Inc. - courses taught at customer site for up to 20 students	Introductory Ada - 5 day seminar for software engineers, etc. who need to become familiar with Ada and its features in order ot wirte Ada programs	14 Main Street	Waltham	Massachusetts	2154	Dr. Benjamin M. Brosgol	617-890-0030
41	SofTech	Ada for Software Mangers - 3 day presentation of Ada's in its entirety from the viewpoint of a technical manager	460 Totten Pond Road	Waltham	Massachusetts	2254	Ada Training Department	617-890-6900
42	SofTech	Ada Management Overview for COBOL Background - 4 day course that presents an overview of software engineering in Ada to managers in business applications	460 Totten Pond Road	Waltham	Massachusetts	2254	Ada Training Department	617-890-6900
43	SofTech	Ada Orientation for Mangers - 1 day overview of Ada's development and features	460 Totten Pond Road	Waltham	Massachusetts	2254	Ada Training Department	617-890-6900
44	SofTech	Ada Program Design Langauge -3/4/5 day course that teaches how to use Ada program desing language (PDL) as a design tool	460 Totten Pond Road	Waltham	Massachusetts	2254	Ada Training Department	617-890-6900
45	SofTech	Ada Programming Support Environment Overview - 1 day course that provides an understanding of the complete software development environment	460 Totten Pond Road	Waltham	Massachusetts	2254	Ada Training Department	617-890-6900

46	SofTech	Ada Technical Overview - 1 day overview for software engineers, programmers, ssytem analysts and software engineering mangaers	460 Totten Pond Road	Waltham	Massachusetts	2254	Ada Training Department	617-890-6900
47	SofTech	Ada Technical Overview for COBOL Background - 4 day course that presents a technical overview of software engineering for business applications	460 Totten Pond Road	Waltham	Massachusetts	2254	Ada Training Department	617-890-6900
48	SofTech	Advanced Ada Topics- 5/10 day course that teaches modern abstraction concepts and the related facilities of Ada	460 Totten Pond Road	Waltham	Massachusetts	2254	Ada Training Department	617-890-6900
49	SofTech	Advanced Ada/Concurrent Processing Topics - 10 day course that introduces advanced techniques in the proper Ada context and Ada desing conceptsin the context of examples	460 Totten Pond Road	Waltham	Massachusetts	2254	Ada Training Department	617-890-6900
50	SofTech	Basic Ada Programming 5/10 day course teaching how to write basic Ada programs	460 Totten Pond Road	Waltham	Massachusetts	2254	Ada Training Department	617-890-6900
51	SofTech	Instructor's Course Module - 1 to 5 day course that trains students to become effective instructors.	460 Totten Pond Road	Waltham	Massachusetts	2254	Ada Training Department	617-890-6900
52	SofTech	Introduction to Ada - 1 day overview of Ada	460 Totten Pond Road	Waltham	Massachusetts	2254	Ada Training Department	617-890-6900
53	SofTech	Introduction to Software Enginering - teaches the fundamental concepts of software engineering to programmers and software designers	460 Totten Pond Road	Waltham	Massachusetts	2254	Ada Training Department	617-890-6900
54	SofTech	Programming Methodology - 1 1/2 day ourse that teaches a practical approach to writing reliable, readable, and maintainable Ada software	460 Totten Pond Road	Waltham	Massachusetts	2254	Ada Training Department	617-890-6900

Commerically Available Ada Courses

55	SofTech	Real-time Concepts - 1 day course teaching approaches to real-time programming	460 Totten Pond Road	Waltham	Massachusetts	2254	Ada Training Department	617-890-6900
56	SofTech	Real-Time Systems in Ada - 5/10 day course in conepts of concurrent programming.	460 Totten Pond Road	Waltham	Massachusetts	2254	Ada Training Department	617-890-6900
57	SofTech	Software Engineering for Mangers - 1 day course that teches managers modern software engineering concpts	460 Totten Pond Road	Waltham	Massachusetts	2254	Ada Training Department	617-890-6900
58	SofTech	Software Engineering Methodologies - 5 day course that provides a thorough understanding of software methodologies and how they can be used with Ada	460 Totten Pond Road	Waltham	Massachusetts	2254	Ada Training Department	617-890-6900
59	SofTech	Systems Engineering Methodology - 3 day course learning to understand systems requirements through the use of structured analysis techniques	460 Totten Pond Road	Waltham	Massachusetts	2254	Ada Training Department	617-890-6900
60	SofTech	Using the Ada Language Reference Manual - 2 day course to learn how to use the reference manual	460 Totten Pond Road	Waltham	Massachusetts	2254	Ada Training Department	617-890-6900
61	Texel and Company	Mgt Track - Ada Impact Issues (1/2 day), Ada for Technical Management (2 1/2 days), Ada: Bids and Proposal (1 day) and Ada: Software Development Plan (1 day)	Victorial Plaza, 615 Hope Road	Eatontown	New Jersey	7724	Harry Copperthwayte	201-922-6323
62	ADAPLUS, INC.	5 day workshop with lab and lecture taught at customer site ($20,000/week) Sept 91 have 5 day advanced training course OOD+Ada+Xwindows	P.O. Box 77113	Houston	Texas	77215	Stephen J. Hyland	713-488-1480
63	GHG Corporation	Advanced Ada Language Features - continuation of the introductory course and intended for those who require the utmost in Ada literacy.	1300 Hercules, Suite 111	Houston	Texas	77058	Gary O'Neal	713-488-8806

64	GHG Corporation	Concurrent Programming in Ada - specialized class that focuses on the anture of concurrent programming and the use of the Ada language in applications.	1300 Hercules, Suite 111	Houston	Texas	77058	Gary O'Neal	713-488-8806
65	GHG Corporation	Embedded/Realtime Programming in Ada - approaches the Ada language from the point of view of embedded real-time systems.	1300 Hercules, Suite 111	Houston	Texas	77058	Gary O'Neal	713-488-8806
66	GHG Corporation	Introduction to High Order Langauge - provides necessary background material for those who have experience in languates which differ significantly from the Ada language	1300 Hercules, Suite 111	Houston	Texas	77058	Gary O'Neal	713-488-8806
67	GHG Corporation	Introduction to the Ada Language - provides bases for using Ada in a broad class of applications	1300 Hercules, Suite 111	Houston	Texas	77058	Gary O'Neal	713-488-8806
68	GHG Corporation	Programming with X Window System - examines X Window System and focuses on teveloping Ada software that will run any X Window system environment	1300 Hercules, Suite 111	Houston	Texas	77058	Gary O'Neal	713-488-8806
69	Computer Sciences Corporation	Ada for Project Mangers - 2 day seminar for software development managers ($4600) Seminar may be presented at customer site	3160 Fairview Park Drive	Falls Church	Virginia	22042	Jeff Seigle	703-876-1438
70	Computer Sciences Corporation	Ada Orientation for Managers - 1/2 day for senior and mid-level managers - non-technical ($2300) Seminar may be presented at customer site	3160 Fairview Park Drive	Falls Church	Virginia	22042	Jeff Seigle	703-876-1438
71	Computer Sciences Corporation	Ada Technical Overivew - 2 day seminar for people with experiencr in a high-level language and have had some exposure to Ada ($4600) Se.minar may be presented at customer site	3160 Fairview Park Drive	Falls Church	Virginia	22042	Jeff Seigle	703-876-1438
72	Computer Sciences Corporation	Advanced Ada Programming - 20 hours of lecture and 20 hours of hands on exercises ($9500) Seminar may be presented at customer site	3160 Fairview Park Drive	Falls Church	Virginia	22042	Jeff Seigle	703-876-1438

Commerically Available Ada Courses

73	Computer Sciences Corporation	Introductory Ada Programming - 2 week course that introduces the attendee to language features in the context of modern software engineering practives ($18500) May be a customer's site	3160 Fairview Park Drive	Falls Church	Virginia	22042	Jeff Seigle	703-876-1438
74	Computer Sciences Corporation	Object-Oriented Design with Ada - 4 day workshop illustrating how object-oriented techniques can be used to constrcut high quality, maintainable Ada software systems($7900)	3160 Fairview Park Drive	Falls Church	Virginia	22042	Jeff Seigle	703-876-1438
75	Computer Sciences Corporation	Object-Oriented Requirements Analysis - 4 day workshop ($7900) (still under development) Workshop may be taught at customer's site	3160 Fairview Park Drive	Falls Church	Virginia	22042	Jeff Seigle	703-876-1438
76	Computer Sciences Corporation	QA and CM for Ada Projects - 3 day workshop under development ($5900) Workshop may be taught at customer's site	3160 Fairview Park Drive	Falls Church	Virginia	22042	Jeff Seigle	703-876-1438
77	Honeywell Federal Systems, Inc.	Ada Application Programming - 10 day course desinged for programmers, software analysts and software engineers. Defines goals and principles of Ada and software engineering	1861 Wiehle Avenue	Reston	Virginia	22090	Willie Griffin	703-478-2032
78	Honeywell Federal Systems, Inc.	Ada for Mangers - 1 day course to introduce non-technical mangers ot the ocncepts and issues involved in the administration of Ada projects.	1861 Wiehle Avenue	Reston	Virginia	22090	Willie Griffin	703-478-2032
79	Honeywell Federal Systems, Inc.	Ada for Project Mangers - 5 day course to develop skills for managing an Ada project.	1861 Wiehle Avenue	Reston	Virginia	22090	Willie Griffin	703-478-2032
80	Honeywell Federal Systems, Inc.	Ada Programming - 10 day course designed to teach programmers experienced in a high-level language	1861 Wiehle Avenue	Reston	Virginia	22090	Willie Griffin	703-478-2032
81	Honeywell Federal Systems, Inc.	Ada Programming Concepts - 5 day course that introduces goals and principles of software engineering. Introduces Ada syntax, data typing and the Ada reference manual	1861 Wiehle Avenue	Reston	Virginia	22090	Willie Griffin	703-478-2032

82	Honeywell Federal Systems, Inc.	Ada Programming Tools - 5 day course designed for systems managers and engineers involved in the development of an Ada system.	1861 Wiehle Avenue	Reston	Virginia	22090	Willie Griffin	703-478-2032
83	Honeywell Federal Systems, Inc.	Ada Software Applied Design - 5 day course discussing aspects of object-oriented design as it relates to software life cycles.	1861 Wiehle Avenue	Reston	Virginia	22090	Willie Griffin	703-478-2032
84	Honeywell Federal Systems, Inc.	Advanced Ada Programming - 10 day course designed to teach programmers experienced in Ada how to code I/O statements and develop code with exception handlers, tasks and generics	1861 Wiehle Avenue	Reston	Virginia	22090	Willie Griffin	703-478-2032
85	Honeywell Federal Systems, Inc.	Advanced Ada Programming - 15 day course where attend designs, encodes and tests complete programs.	1861 Wiehle Avenue	Reston	Virginia	22090	Willie Griffin	703-478-2032

Ada Education and Training

Universities Teaching Ada

	University	Courses Provided	Address	City	State	Zip Code	Point of Contact	Phone Number
1	Alabama A&M	Structured Programming with Advanced Languages: Ada	Dept of Computer & Info Sciences P.O. Box 88	Normal	Alabama	35762	Dr. Hrishikesh Saha	205-859-7339
2	Auburn University	Advanced Programming in Ada	Comp Sci and Eng Dept 111 Dunstan Hall	Auburn	Alabama	36849	Dr. Thomas Phillips	205-826-4330
3	Birmingham-Southern College	1-Alternative Languages 2-The Ada Programming Lang.	Div. of Science and Math 800 8th Ave West	Birmingham	Alabama	35254	Richard Turner	205-226-4870
4	University of Ala/Huntsville	Software Development and Design Using Ada	CSC Department	Huntsville	Alabama	35899	Warren Mosely	205-895-6088
5	University of Alabama	Ada and Concurrent Programming	Dept. of Comp Sci	Univeristy	Alabama	35487	Dr. Wen-Kai Chung	205-348-6363
6	University of Southern Alabama	Programming Lang: Ada	Div of Computer & Information Science	Mobile	Alabama	36688	Marino Niccolai	205-460-6390
7	University of Alaka	Computer Programming II (Includes Ada as a second language)	Dept of Math & Computer Science Chapman Building	Fairbanks	Alaska	99775	Barbara Lando	907-474-7332
8	University of Alaska Southeast	Ada for Programmers	School of Bus & PA 1108 F. Street	Juenau	Alaska	99801	Timothy J. Fullam	907-789-4402
9	Arizona State University	Introduction to Ada	Computer Science Department	Tempe	Arizona	85287	Dr. Terry Mellon	501-965-2774
10	Azusa Pacific University	Structured Programming 2 - Ada	Computer Science Department P.O. Box APU	Azusa	California	91702	Wendell Scarborough	818-969-3434
11	CA State Polytechnic Univ/Pamona	1-Ada 2- Software Engineering	Department of Computer Science	Pamona	California	91788	Dr. Kenneth McDonald	714-869-3440

Universities Teaching Ada

	University	Courses Provided	Address	City	State	Zip Code	Point of Contact	Phone Number
12	Carnegie Mellon	Software Engineering	Department of Computer Science	Pittsburgh	Pennsylvania	15213	Dr. Nico Habermann	412-268-2592
13	Stephen F. Austin State University	Software Development Applications	Schol of Bus Admin Dept of Comp Sci	Nacogdoches	Texas	75962	Dr. Jarrell Grout	409-568-1876
14	SW Texas State	One undergraduate course and one advanced course			Texas			
15	CA State Univ/ Long Beach	Software Engineering with Ada	Comp Sci & Eng Dep 1250 Bellflower Blvd.	Long Beach	California	90840	Joel Carissimo	213-498-4285
16	CA State Univ/ Northridge	Software Engineering with Ada	Dept. of Computer Scelnce, School of Engineering	Northridge	California	91320	Shawn Barkataki	818-885-3398
17	CA State University	*Advanced Software Practices*	Department of Computer Science	Chico	California	9592041	Paul Luker	916-895-6442
18	California Lutheran University	CS 212A: Ada Programming	60 West Olsen Road	Thousand Oaks	California	91360	Roy James Guild	805-493-3362
19	California State Univ/LA	Ada Programming	Dept of Math & Comp Science 5151 State Univ Dr	Los Angeles	California	90032	Mr. Fraser	213-224-3287
20	CSU/Dominguez Hills	High Level Languages: Ada	1000 E. Victoria St Building NSM A132	Carson	California	90747	Dr. R. Huddlestone	213-224-3287
21	Harvey Mudd School	1-Programming Languages 2-Introduction to Programming	Dept. of Computer Science	Claremont	California	91711	Dr. Michael Erlinger	714-621-8225
22	Merritt College	Software Engineering with Ada		Oakland	California		Dr. Richard D. Riehle	415-858-1551

Universities Teaching Ada

	University	Courses Provided	Address	City	State	Zip Code	Point of Contact	Phone Number
23	RPI	Graduate Level courses			New York			
24	San Jose State University	Software Engineering with Ada	Dept. of Math & Computer Science	San Jose	California	95192	Evelyn E. Obaid	408-924-5139
25	Univ. of California/ Santa Barbara	Programming Languages	Computer Science Department	Santa Barbara	California	93106	Laura Dillon	805-961-3411
26	University of CA at Irvine	Ada	Information & Comp Science Dept	Irvine	California	92717	Dr. Dennis Volper	714-856-7403
27	Pikes Peak Community College	Introduction to Ada Programming	3675 S. Academy Blvd.	Colorado Springs	Colorado	80906	Vivian M. Challen	303-576-7711
28	University of Colorado at Denver	Parallel Computing with Ada	Math Department University Box 170 1200 Larimer St.	Denver	Colorado	80204	Dr. Zenas Hartvigson	303-556-8442
29	University of S. Colorado	1-Ada and Software Eng II 2-Software Engineering with Ada	Computer Science Department 2200 N. Bonforte	Pueblo	Colorado	81001	Robert Cook	303-549-2752
30	Central Connecticut State U	Advanced Topics in Computer Science: Ada	Department of Math & Comp Sci 1615 Stanley St	New Britain	Connecticut	6050	A. Zoe Leibowitz	203-827-7568
31	Central State University	Advanced Topics in Computer Science: Ada	1615 Stanley Street	New Britain	Connecticut	6050	A. Zoe Leibowitz	203-827-7568
32	Southern Connecticut State Univ	1-Ada Programming 2-Organization of Programming Languages	Computer Science 501 Crescent Street	New Haven	Connecticut	6515	Dr. JoAnn Parikh	203-397-4514
33	University of New Haven	Programming in Ada	Depart of Industrial Eng & Computer Science	West Haven	Connecticut	6516	Gary Walters	203-932-7067

Universities Teaching Ada

	University	Courses Provided	Address	City	State	Zip Code	Point of Contact	Phone Number
34	Brevard Community College	1-Intro to Ada Programming 2-Advanced Ada	Business Division 1519 Clear Lake Rd	Coco	Florida	32926	Dennis Koile	305-632-1111
35	Embry-Riddle Aeronautical University	Intro to Software Engineering	Computer Science ERAU (PD-AAC)	Daytona Beach	Florida	32014	Dr. Jagdish Agrawal	904-239-5590
36	Florida Institute of Technology	Ada and its Programming Environment	150 University Ave. Dept of Comp Sci	Melbourne	Florida	32901	Luwana Clever	305-768-8091
37	Florida State University	1-Real-Time Programming 2-Software Engineering with Ada	Compu Sci Dept Room 206 Love Building	Tallahassee	Florida	32306	Greg Riccardi/Ted Baker	904-644-2296
38	St. Peter's Jr College	Ada Programming	2465 Drew St.	Clearwater	Florida	34615	Jim Hill	813-791-2530
39	Univ. of Central Florida	1-Software Engineering I 2-Software Engineering II	Dept of Computer Eng CEBA 207	Orlando	Florida	32816	Dr. Darrell Linto	305-275-2236
40	University of Southern Florida	Introduced in Sophmore and Junior level courses			Florida			
41	University of Miami	Software Development with Ada	Elec & Comp Eng P.O. Box 248294	Miami	Florida		Susan D. Urban	305-284-3452
42	University of Florida	Upper level courses			Florida			
43	Armstrong State Univ	Comparative Languages	Dept of Comp Sci Dept of Math & Comp Sci	Savannah	Georgia	31419	Dr. Sigmund Hudson	912-927-5317
44	Atlanta Univ	1-Design and Programming Languages 2-Software Engineering	Dep Math & Com Sci James Brawley Dr., SW	Atlanta	Georgia	30314	Steven Ornburn	404-681-0251

Universities Teaching Ada

	University	Courses Provided	Address	City	State	Zip Code	Point of Contact	Phone Number
45	Georgia Institute of Technology	Abstraction and Specification in Program Development	School of Inofrmaiton/Comp Sci	Atlanta	Georgia	30332	Richard LeBlanc	404-894-2592
46	Georgia State University	Software Engineering	Dept. of Math & Compu Sci	Atlanta	Georgia	30303	Dr. Scott Owen	404-651-2253
47	La Grange College	Introduction to Object Oriented Design	Computer Science Department 601 Broad Street	LaGrange	Georgia	30240	Tony Valle	404-882-2911
48	Morehouse College	Introduction to Ada	Dept of Comp Sci P.O. Box 137	Atlanta	Georgia	30314	William McGuiver	404-525-1501
49	Univ of Georgia	Software Engineering	Dept of Com Sci 415 Boyd Graduate Studies Center	Athens	Georgia	30602	Dr. Orville Weyrich	404-542-2911
50	Chaminade University	Special Topics: Ada	Computer Science Department 3140 Waial	Honolulu	Hawaii	96816	Ward Hayward	808-735-4805
51	Leeward Community College	The Programming Language	96 045 Ala Ike	Pearl City	Hawaii	96782	LeRoy C. Johnson	808-455-0273
52	DePaul University	1-Programming in Ada 2-Software Engineering	Comp Sci & Info Sy 243 S. Wabash	Chicago	Illinois	60604	George Knafl	312-341-8381
53	Ohio State	Advanced courses in Ada			Ohio			
54	Elmhurst College	Software Engineering	Dept of Math & Comp Sci 190 Prospect Ave	Elmhurst	Illinois	60126	John Jeffrey	312-279-4100
55	Illinois Institute of Technology	1-Software Eng. With Ada 2-Concurrent Programming	Dept of Comp Sci SP Building, IIT Central	Chicago	Illinois	60616	Fred Maymir	312-567-5142

Universities Teaching Ada

	University	Courses Provided	Address	City	State	Zip Code	Point of Contact	Phone Number
56	McKendree College	1-Ada Programming I 2-Ada Programming II 3- Ada Programming III 4-Ada Programming IV	Computer Science Department 701 College Road	Lebanon	Illinois	62254	Capt. Roy Rogge	618-537-4481
57	Parks College of St. Louis Univ.	Software Engineering with Ada	Computer Science Department	Cahokia	Illinois	62206	Dr. C.C. Kirkpatrick	618-337-7500
58	Southern Illinois Univ at Edwardsville	1-Programming Lang Concepts 2-Topics in S/W Eng Using Ada	Department of Computer Science	Edwardsville	Illinois	62026	Dr. Hattemer	618-692-2386
59	Western Illinois University	The Language Ada	Department of Compter Science	Macomb	Illinois	61455	Dr. David Ballew	309-298-1452
60	Indiana State Univ at Terre Haute	Ada for Systems Programming	Math & Comp Sci Holmstedt Hall	Terre Haute	Indiana	47809	Dr. Guy Hale	812-237-2130
61	Indiana Univ-Purdue Univ/Fort Wayne	1-Data and File Structures 2-Object-Oriented System Development	Comp Tech Dept. 2101 Coliseum Blvd. E.	Fort Wayne	Indiana		Karl Rehmer Mark Temte	219-481-6176 219-481-6803
62	Rose-Hulman Institute of Technology	Introduction to Ada	5500 Wabash Ave	Terre Haute	Indiana	47803	Cary Laxer	812-877-1511
63	University of Evansville	Ada Programming	Dept of Compu Sci 1800 Lincoln Ave	Evansville	Indiana	47722	Mr. Bruce Mavis	812-479-2652
64	Cornell University	Programming Language Concepts	Dept. of Computer Science	Mt. Vernon	Iowa	52314	Tony DeLaubenfels	319-895-8811
65	Iowa State University	Software Engineering	Dept of Computer Science	Ames	Iowa	50011	Albert L. Baker	515-294-4377
66	Simpson College	Introduction to Programming Ada	Dept. of Comp Sci 701 N. C Street	Indianola	Iowa	50125	Richard A. Bee Be	515-961-1586

Universities Teaching Ada

	University	Courses Provided	Address	City	State	Zip Code	Point of Contact	Phone Number
67	University of Iowa	Prgramming Language Concepts	Dept. of Computer Science	Iowa City	Iowa	52242	Raymond Ford	319-335-0707
68	Hutchinson Community College	Ada Language Programming	1300 North Plum Street	Hutchinson	Kansas	67501	John Morrell	316-665-3500
69	St. Mary College	1-General Programming I 2-General Programming II	Comp Sci Dept 4100 South 4th Street	Leavenworth	Kansas	66048	Victor Meyer	913-682-5151 x319
70	Wichita State University	1-Ada 2-Ada & Software Engineering 3-Intro to Software Eng, 4-S/W Testing & Reliability	Comp Sci Dept Box 83	Wichita	Kansas	67208	Mark Rutter James E. Tomayko	316-689-3156 316-689-3155
71	Vanderbilt University	Software Engineering	School of Eng. Dept of Comp Sci	Nashville	Tennessee	37235	Dr. Stephen R. Schach	615-322-2924
72	Eastern Kentucky Univ	Advanced Programming Techniques with Ada	Dept of Statistics/Comp Science Wallace 402	Richmond	Kentucky	40475	Don Greenwell	606-622-5942
73	University of Kentucky	Programming Languages	Dept of Comp Sci 915 Patterson Office Tower	Lexington	Kentucky	40506	Prof. Harris	606-257-3961
74	Western Kentucky Univ	Ada Programming	Dept of Computer Science	Bowling Green	Kentucky	42101	Dr. Crenshaw	502-745-4642
75	Northern Kentucky Univ	Programming Lanugages	Dept of Math and Computer Science	Highland Heights	Kentucky	41076	Don Galli	606-572-5320
76	University of South Western Louisiana	1-Ada Programming II 2-Programming in Ada/Intro to Software Engineering	119 Stevens Memorial Hall	Lafayette	Louisiana	70503	Jagadeesh Namdigan	318-231-5647
77	University of Maine/Orono	Software Engineering	Comp Sci Dept 222 Neville Hall	Orono	Maine	4469	Dr. Larry Latour	207-581-3941

Universities Teaching Ada

	University	Courses Provided	Address	City	State	Zip Code	Point of Contact	Phone Number
78	Johns Hopkins University	Software Engineering with Ada	Cont. Prof. Programs-GWC Whiting School of Eng.-Merryman Hall	Baltimore	Maryland	21218	Mr. Gralla	301-338-8728
79	Universty of Maryland	1-Introduction to Ada 2-Applying Adv Features in Ada 3-Concepts in Ada	University College	College Park	Maryland	20742	Duane Jarc Helmut Theiss	301-985-7000 301-985-70
80	Univ. of Maryland at College Park	1-Programming in Ada 2-Software Design & Dev 3-Software Development in Ada	Dept of Comp Sci	College Park	Maryland	20742	Dr. Rombach	301-454-2002
81	Utah State University	Software Development/Implementation	College of Science Dept. of Comp Sci	Logan	Utah	84322-4205	Prof. Greg Jones	801-750-3267
82	Towson State University	Software Engineering Using Ada	Dept. of Computer and Information Sciences	Towson	Maryland	21204	Mr. Helmut Theiss	301-321-2633
83	Univ. of Maryland/ Eastern Shore	Topics in Programming Languages: Ada	Dept of Math and Computer Science	Princess Anne	Maryland	21853	Edward Chapin	301-651-2200
84	Boston University	1- System Design 2- Embedded Computer Software Design 3-Introduction to Ada	College of Eng. 110 Cummington St	Boston	Massachusetts	02215	Dr. Richard Vidale	617-353-2808
85	Univ of Mass/Amherst	Software Engineering	Dept of Computer and Information Sciences	Amherst	Massachusetts	01003	Eliot Moss	413-545-2744
86	Central Michigan University	Alternative Programming Langauges	Dept of Comp Sci Pearce Hall	Mt. Pleasant	Michigan	48859	Cindy Burt	517-774-3774
87	Michigan State	Ada: An Introduction	2244 Lansing Avenue	Detroit	Michigan	44657	Malcolm Davis	800-778-9009
88	Western New England College	1-Data Structures 2-Organization of Programming Langauges	Dept of Math and Computer Science	Springfield	Massachusetts	01119	Prof. L.S. Tang Prof. Lloyd Emerson	413-782-3111

Universities Teaching Ada

	University	Courses Provided	Address	City	State	Zip Code	Point of Contact	Phone Number
89	Eastern Michigan University	Softwrae Engineering	Comp Sci Dept 620 Pray-Harrold Bldg	Ypsilante	Michigan	48197	Dr. William McMillan	313-487-1063
90	North Adams State College	1-Advanced Programming Langs 2-Systems Software Design 3-Comparative Prog Languages	Dept of Computer Science	North Adams	Massachusetts	01247	Ernie Giangrande Beverly Smith	413-664-4511
91	Southeast Mass University	1-Software System Design with Ada 2-Process Based Design	Computer Science Department	N. Dartmouth	Massachusetts	02747	Jan Bergandy	617-999-8293
92	Western Michigan University	Programming Languages	Dpet. of Computer Science	Kalamazoo	Michigan	49008	Dr. Kenneth Williams	616-383-6151
93	Oakland University	Short Course in Ada Programming	Dept of Comp Sci & Eng. Dodge Hall of Eng.	Rochester	Michigan	48063	Dr. Frank Cioch	313-370-2200
94	University of Michigan	Ada Based Software Engineering	Computer Science 3314 EACS Building	Ann Arbor	Michigan	48109-2122	Dr. Richard Volz	313-763-0035
95	Saginaw University	Software Design and Development	Science 357 2250 Pierce Road	University Center	Michigan	48710	Katherine Kerr	
96	Winona State University	Introduction to Ada	Department of Computer Science	Winona	Minnesota	55987	Mr. Daryl Henderson	507-457-5385
97	Northwest Missouri State University	Specialized Languages: Ada	Compu Sci Dept	Maryville	Missouri	64468	Richard Detmer	816-562-1187
98	Jersey City State College	1-Introductory Ada 2-Software Engineering	Ada Tech Center 2039 Kennedy Blvd	Jersey City	New Jersey	07305	Phillip Caverly	201-547-3291
99	Hofstra University	Ada for PL/I, Pascal or Fortran Users Advanced Programming Techniques for Business App	Dept of Comp Sci Business Computer Info Systems	Hempstead	New York	11550	Dr. Phillip J. Pnzeca Dr. Vasiliscu	516-560-5555 516-560-5716

Universities Teaching Ada

	University	Courses Provided	Address	City	State	Zip Code	Point of Contact	Phone Number
100	San Diego State			San Diego	California			
101	University of Minnesota	1-Software Engineering I 2-Software Engineering II 3-Software Engineering III 4-Software Eng with Ada	Dept of Computer Science	Minneapolis	Minnesota	55455	Dr. Wei-Tak-Sai	612-625-4002
102	University of Virginia	Software Engineering	Department of Computer Science	Charlottesville	Virginia	22903	Prof. Robert Cook	804-924-7605
103	Texas A&M	400 Level Courses	Computer Science Department	College Station	Texas			
104	Univ. of Mississippi at Oxford	1-Software Engineering Using Ada 2-Programming in Ada	Comp & Info Sci Farley Hall, Room 331	University	Mississippi	38677	Pam Lawhead	601-232-7396
105	Univ. of Southern Mississippi	1-Operating Systems & Computer Architecture II 2-Software Engineering II	Dept of Comp Sci Box 5106 Southern Station	Hattiesburg	Mississippi	39406	Cliff Burgess Ralph Bisland, Jr.	601-266-4958 601-266-4949
106	University of Missouri at Columbia	Programming Languages	Dept of Comp Sci Mathematical Science Building	Columbia	Missouri	65211	William Slough	314-882-3842
107	St. Louis Community College Meramec	Ada Programming	11333 Big Bend	St. Louis	Missouri	63122	Robert L. Monsees	314-966-7526
108	Southeast Missouri State University	Ada Programming	Compu Sci Dept	Cape Girardeau	Missouri	63701	Michael Britt	314-651-2525
109	Montclair State College	1-Programming Languages 2-Programming Langauges Design	Dept of Math/Computer Science	Upper Montclair	New Jersey	07043	Carl Bredlau	201-893-4263
110	Fairleigh Dickinson University	1-Advanced Programming Language Constructs Using Ada 2-Concepts of Prog Languages 3-Prog Language Concepts	Dept of Comp Sci 1000 River Road	Teaneck	New Jersey	07666	Gertrude Neuman Levine	201-692-2020/ 2261

Universities Teaching Ada

	University	Courses Provided	Address	City	State	Zip Code	Point of Contact	Phone Number
111	Univ. of New Mexico/Albuquerque	Software Engineering With Ada	Computer Science Department	Albuquerque	New Mexico	87131	Chalres Crowley	505-277-3112
112	New Mexico University/Las Cruces	Ada Programming	Dept of Comp Sci Box 3CU	Las Cruces	New Mexico	88003	Don Dearholt	505-646-3723
113	Rochester Insitute of Technology	Algorithms and Data Structures	Graduate Computer Science Dept. One Lomb Memorial Drive	Rochester	New York	14623-0887	Dr. Peter Anderson	716-475-2529
114	Long Island UnivCW Post Campus	1-Embedded and Scientific Systems Using Ada 2-Software Engineering with Ada	Computer Science Department	Brookville	New York	11548	Ms. Susan Dorchak	516-299-2293
115	State University of New York/ Potsdam	Selected Lnaguage of Ada	Department of Computer Science	Potsdam	New York	13676	David Rokh	315-267-2073
116	Niagra University	1-Programming Languages 2-Topics in Computer Science	Department of Computer and Information Sciences	Niagra University	New York	14109	Dr. Hubbard	716-285-1212
117	East Carolina University	Organization of Programming Language		Greenville	North Carolina	27834	Dr. Masao Kishore	
118	University of North Dakota	1-Ada 2-Software Eng. with Ada	Department of Computer Science	Grand Forks	North Dakota	58202	Randy Molmen Dr. Lonny Winrich	701-777-4107
119	North Dakota State University	New Developments in Programming Languages	Box 5075	Fargo	North Dakota	58105	Ken Magel	701-237-8189
120	North Dakota State University	New Developments in Programming Languages	Box 5075	Fargo	North Dakota	58105	Ken Magel	701-237-8189
121	West Virginia College of Graduate Studies	1-Software Engineering with Ada 2-Introduction to Ada Programming	Eng. & Science Div Information Systems	Institute	West Virginia	25112	Robert N. Hutton	304-768-9711

Universities Teaching Ada

	University	Courses Provided	Address	City	State	Zip Code	Point of Contact	Phone Number
22	New Jersey Institute of Technology				New Jersey			
23	Canisius College	Programming Languages	Dept. of Comp Sci 2001 Main Street	Buffalo	New York	14208	Dr. Patricia Van Verth	716-883-7000
24	State University of New York/ Fredonia	1-Introduction to Ada 2-Ada: A Seminar for Faculty	Dept of Math and Computer Science	Fredonia	New York	14063	Dr. Joseph Straight	716-673-3459
25	Le Moyne College	1-Software Eng Project 2-Intro-Program Methodology 3-Data Structures & Program Development		Syracuse	New York	13214	James F. Smith	315-445-4544
26	University of Toledo	1-Survey of High Level Programming Languages 2-Concurrent Programming	Dept of Computer Science and Engineering	Toledo	Ohio	43606	Dr. Hilda Standley	419-537-2303
27	University of Dayton	1-Algorithms & Programming II 2-Data Structures	300 College Park CMSC Department	Dayton	Ohio	45469	Joseph Lang	513-229-3831 513-229-2192
28	Penn State University	Software Design Methods	220 Whitmore Lab	University Park	Pennsylvania	16802	Fred L. Bierly	814-863-1241
29	University of Wisconsin				Wisconsin			
30	Kent State University	1-Ada Programming 2-Advanced Ada	Dept of Mathematical Sciences	Kent	Ohio	44242-0001	Keith Yerian	216-672-2209
31	Univ. of Cincinnati	Sepcial Topics: Programming in Ada	Department of Computer Science	Cincinnati	Ohio	45221-0008	Dieter Schmidt	513-475-6964
32	Gwynedd-Mercy College	1-Ada 2-Software Engineering Using Ada	Comp Sci Dept Sumneytown Pike	Gwynedd Valley	Pennsylvania	19437	Michael G. Gonzales	215-641-5547

Universities Teaching Ada

	University	Courses Provided	Address	City	State	Zip Code	Point of Contact	Phone Number
133	Lebanon Valley College	Programming in Ada	Dept of Math Science	Annville	Pennsylvania	17003	Mike Fry	717-867-6188
134	Widener University	Programming Languages	Dept of Comp Sci Science Division	Chester	Pennsylvania	19013	Dr. Norman Adams	215-499-4002
135	Tennessee State University				Tennessee			
136	Cleveland State University	Development of Large Programming Systems	Comp Sci Dept Euclid At 24th St.	Cleveland	Ohio	44115	Paul Jalick	216-687-4760
137	Ohio Northern University	Software Engineering	Department of Math and Computer Science	Ada	Ohio	45810	David A. Retterer	419-772-2346
138	Franklin University	Organization of Programming Languages	Department of Computer Science 201 S. Grand Ave.	Columbus	Ohio	43215	Bob Vermilyer	614-224-6237
139	Marietta College	Data Structures in Algorithm Analysis	Computer Science Department	Marietta	Ohio	45750	E. Robert Anderson	614-374-4600
140	Oklahoma State University	Ada Programming Language	Department of Computing & Information Sciences, MS-219	Stillwater	Oklahoma	74078	Dr. K.M. George	405-624-5668
141	Cameron University	Intermediate Programming with Ada	Dept of Math Sciences West Gore	Lawton	Oklahoma	73505	Feridoon Molnian	405-581-2481
142	Villanova University	1-The Linguistics of Programming Languages 2-Organization of Programming Languages	Computer Science Program	Villanova	Pennsylvania	19085	Robert Beck Lillian Cassell	215-645-7307
143	University of Pittsburgh	Programming Languages	Dept of Comp Sci Alumni Hall	Pittsburgh	Pennsylvania	15260	Dr. George Novacky	412-624-8490

Universities Teaching Ada

	University	Courses Provided	Address	City	State	Zip Code	Point of Contact	Phone Number
144	University of Texas	Software Engineering	Department of Computer Science	Austin	Texas	78712	Laurie H. Werth	512-471-9535
145	Sam Houston State University	1-Ada 2-Ada: Object-Oriented Programming	P.O. Box 2206	Huntsville	Texas	77341	Dr. Burris Wuhslung Lu	409-294-1568 409-294-1837
146	Syracuse University	Graduate level courses		Syracuse	New York	13214		
147	University of New Orleans			New Orleans	Lo[illegible]anna			
148	Miami Univeristy of Ohio	The Ada Programming Language	Systems Analysis Department Kreger Hall	Oxford	Ohio	45056	Jim Kiper	513-529-1252
149	University of Scranton	Programming Languages	Computer Science Department	Scranton	Pennsylvania	18510	Dennis Martin	717-961-6115
150	East Tenn State Univ	1-Advanced Prog Techniques 2-Software Engineering 3-Systems Design	Comp Sci Dept Box 23830A	Johnson City	Tennessee	37614-0002	Suzanne Smith	615-929-6963
151	Notre Dame		Computer Science Department	South Bend	Indiana			
152	University of Tulsa	Comparative Programming Languages	Dept of Math & Comp Sci 600 South College	Tulsa	Oklahoma	74104	Travis Tull	918-542-6000 x2228
153	Central State University	1-Programming in Ada 2-Computer Networks	Dept of Comp Sci 100 N. University	Edmond	Oklahoma	73034	Bill McDaniel	405-341-2980
154	Oral Roberts University	Special Topics: Software Engineering	Dept of Math/Comp Science 7777 S. Lewis	Tulsa	Oklahoma	74171	Jeffrey Jackson	918-495-6701

Universities Teaching Ada

	University	Courses Provided	Address	City	State	Zip Code	Point of Contact	Phone Number
155	Beaver College	Modern Programming Languages: Ada	Dept of Computer Science & Mathematics	Glenside	Pennsylvania	19038	Mark Balcer	215-572-2984
156	Cheyney University of PA	Software Engineering Using Ada	Dept of Mathematics and Computer Science	Cheyney	Pennsylvania	19319	Jesse Williams	215-399-2435
157	Elizabethtown College	Comparison of Programming Languages	Department of Computer Science 1 Alpha Drive	Elizabethtown	Pennsylvania	17022	Ms. Barbara Tulley	717-367-1151
158	Clemson University	Programming Systems	Dept of Electrical/ Computer Engineering	Clemson	South Carolina	29634	Dr. James Leathrum	803-656-5930
159	Prairie View A&M	1-Introduction to Ada 2-Advanced Ada	Department of Computer Science	Prairie View	Texas	77446	N. Ravindran	409-857-2715
160	Slippery Rock University	Ada	Dept of Comp Sci Slippery Rock U	Slippery Rock	Pennsylvania	16057	Richard Hunkler	412-794-7133
161	Univ of Rhode Island	Software Engineering	Dept of Compu Sci and Statistics Tyler Hall	Kingston	Rhode Island	02881	Jan Prichard	401-792-2701
162	Louisiana Tech University	1-Software Methodology 2-System Design	Department of Computer Science	Ruston	Louisiana	71272	Prof. Margaret Schaar	318-257-2298
163	National University	5- Data Base Mgt. 6-Princ of H/W & S/W Integr 7-Expert Systems 8-V & V Techniques		San Diego	California	92108	Prof. Peter Sibley	619-563-7123
164	National University	9-Software Eng. Project I 10-Software Eng. Project II 11-Softwre Eng. Project III		San Diego	California	92108	Prof. Peter Sibley	619-563-7123
165	Mississippi State University	Courses offerred at the Junio and Senior level			Mississippi			

Universities Teaching Ada

	University	Courses Provided	Address	City	State	Zip Code	Point of Contact	Phone Number
166	Memphis State University	1-Ada Programming 2-Operating Systems	Department of Mathematical Sciences	Memphis	Tennessee	38152	David Vaught	901-454-2482
167	Middle Tennessee State University	Programming Languages	Comp Info Systems Box 50	Murfreesboro	Tennessee	37132	Dr. Nathan Adams	615-898-2362
168	Tennessee Technical University	Advanced Programming - Ada	Comp Sci Dept Box 5101	Cookeville	Tennessee	38505	Donald C. Ramsey	615-372-3691
169	State Technical Institute at Knoxville	Ada	Hardin Valley Road Box 22990 ATTN: CST Dept	Knoxville	Tennessee	37933	Gerald Wlaker	615-694-6468
170	University of North Texas	Introduction to Software Engineering	Dept of Comp Sci P.O. Box 13886	Denton	Texas	76203	Dr. Jeff Harris	817-565-2801
171	Texas Technical University	Structured Programming and Software Engineering	Dept of Comp Sci Mail Stop 3102	Lubbock	Texas	79409	Dr. James Archer	806-742-3527
172	University of Houston/Clear Lak	1-Ada Programming Lang 2-Software Design 3-Dev of Software Tools 4-Seminars in Software Eng.	Department of Computer Science	Houston	Texas	77058	Theodore Liebfried Dr. Charles McKay Dr. Anthony Lakkos Dr. Charles McKay	713-488-9480
173	East Texas State University	Survey of Programming Languages	Computer Science Department	Commerce	Texas	75428	Sandra Huerter	214-886-5409
174	McMurry College	Ada Programming with Applications	Computer Science Department	Abilene	Texas	79697	Louis Volt	915-691-6393
175	Univ. of Texas at Arlington	1-Introduction to Software Engineering with Ada 2-Adv. Software Engineering 3-Software Engineering in Ada	Comp Sci Eng Dept P.O. Box 19015	Arlington	Texas	76019	Dr. Paul C. Grabow	817-273-2348
176	Univ. of Alabama at Birmingham	Formal Specifications and Software Development	Sch of Natural Sci Dept of Comp & Info Sciences	Birmingham	Alabama	35294	Dr. Warren Jones	205-934-2213

Universities Teaching Ada

	University	Courses Provided	Address	City	State	Zip Code	Point of Contact	Phone Number
177	Texas Christian University	1-Ada Software Development and Programming 2-Ada Design and Development	Comp Sci Dept P.O. Box 32886	Fort Worth	Texas	76129	Ted Tenny Tom Nute	817-921-7166
178	Weber State College	Emerging Techniques in Computing	Computer Science Department	Ogden	Utah	84408-2401	David Hart	801-626-7093
179	Utah Valley Community College	Ada: A First Language	800 West 1200 South	Orem	Utah	84057	Dr. Harrington	801-226-5000
180	Vermont Technical College	1-Introduction Ada Programming 2-Advanced Ada Programming	Electrical & Electronic Eng. Technology Dept.	Randolph Center	Vermont	05061	Dr. Carl Brandon	802-728-3391
181	George Mason University	Real-Time Systems Design and Development	School of Info Tech and Engineering 4400 University Dr	Fairfax	Virginia	22030	Dr. Jorge Diaz-Herrera	703-323-2713
182	Old Dominion University	Ada Programming	Department of Computer Science	Norfolk	Virginia	23508	Hill Price	804-440-3915
183	Christopher Newport College	Ada Programming Language	Dept of Comp Sci 50 Shoe Lane	Newport News	Virginia	23606	Prof. Tean-Quay Lee	804-599-7065
184	Norfolk State University	1-Ada Programming I 2-Ada Programming II	Dept of Math & Computer Science 2401 Corprew Ave	Norfolk	Virginia	23504	George C. Harrison	804-623-8654
185	Hampton University	1-Introduction to Ada 2-Advanced Ada Programming	Department of Computer Science	Hampton	Virginia	23668	Robert A. Willis	804-727-5552
186	Gonzaga University	Programming Languages	Dept Math & and Computer Science 509 E. Boone	Spokane	Washington	99258	Brian Carlson	509-328-4220
187	Eastern Washington University	Advanced Programming in Ada	Computer Science Department	Cheney	Washington	99004	Dr. Ray Hamel	509-458-6260

Universities Teaching Ada

	University	Courses Provided	Address	City	State	Zip Code	Point of Contact	Phone Number
188	West Virginia University	1-Ada & Object-Oriented Desig 2-AI Applications of Ada 3-Introduction to Computing 4-Software Engineering	Dept of Stats & Comp Sci Knapp Hall	Morgantown	West Virginia	26506	Dr. Frances VanScoy	304-293-3607
189	West virginia Wesleyan College	Ada Programming	Dept of Math & Computer Science	Buckhannon	West Virginia	26201	Ron Klausewitz	304-473-8000
190	Marshall University	Software Engineering with Ada	Dept of Computer & Information Sciences	Huntington	West Virginia	25701	Kathleen Warner	304-696-5424
191	Beckley College	Introduction to Ada Programming	Dept of Comp Sci P.O. Box AG	Beckley	West Virginia	25802	Stephanie Ketz	304-253-7351 ext. 14
192	Alderson Braddice College	1-Computer Language: Ada 2-Software Engineering	Div of Natural Sci Dept of Comp Sci	Philippi	West Virginia	26416	Alicia Kime Gary Schubert	304-457-1700
193	West Virginia Institute of Technology	Special Tipics - Ada Programming	Department of Computer Science	Montgomery	West Virginia	25136	Don Smith	304-442-3361
194	Howard Unviersity	1-Advanced Prog. Langauges 2-Real-Time Systems	Systems & Computer Science School of Eng.	Washington	D.C.	20059	Don Coleman	202-636-6595
195	American University	Data Structures	Dept of Comp Sci and Information Sciences	Washington	D.C.	20016	Richard A. Holzsager	202-885-1470
196	Gallaudet University	Data Structures using Ada	Dept of Math/Computer Science	Washington	D.C.	20002	Howard L. Egan	202-651-5315
197	Colorado State University				Colordao			
198	Marquette University	1-Programming Languages 2-Ada Programming Language	Department of Math, Statistics & Computer Science	Milkaukee	Wisconsin	53233	Dr. George Corliss	414-224-7573

Universities Teaching Ada

	University	Courses Provided	Address	City	State	Zip Code	Point of Contact	Phone Number
199	Ball State University	Principles of Software Engineering	Dept. of Comp Sci Program in Comp Sci	Muncie	Indiana	47306	Prof. W.F. Brown	317-285-8644
200	Brigham Young University	Introduction to Software Design	Dept of Computer Science	Provo	Utah	84602	Prof. Scott Woodfield	801-378-2915
201	Florida Atlantic University	Software Engineering	Department of Computer Science	Boca Raton	Florida	33431	Dr. Neal Doulter	305-393-3180
202	National University	1-Princ of S/W Engineering 2-Intro to Appl Prog Lang-Ada 3-Adv. Appl Programming 4-Advanced Software Engin	School of Engineering and Computer Sciences Master of Science	San Diego	California	92108	Prof. Peter Sibley	619-563-7123
203	North Carolina State University	Software Engineering with Ada	Dept of Comp Sci Prog in Comp Sci	Raleigh	North Carolina	27695	Prof. K.C. Tai	919-737-7862
204	Southwest Texas State University	Advanced Software Engineering	School of Science Dept. of Comp Sci	San Marcos	Texas	78666	Dr. C.J. Hwang	512-245-3409
205	Stanford University	Oject-Oriented Design with Ada	School of Eng. Dept. of Comp Sci	Stanford	California	94305	Proj Stuart Reges	415-723-9798
206	State University of New York/ Binghamton	1-Software Engineering I 2-Software Engineering II	Thomas J. Watson School of Eng, Applie Sci & Tech Dept of Comp Sci	Binghamton	New York	13901	Proj Thomas Piatkowski	607-777-4802
207	University of Arizona				Arizona			
208	Mississippi Valley State				Mississippi			
209	New Mexico State University	Software Development	Department of Computer Science Prog in Comp Sci	Las Cruces	New Mexico	88003	Prof. Dan Dearholt	505-646-3724

Universities Teaching Ada

	University	Courses Provided	Address	City	State	Zip Code	Point of Contact	Phone Number
210	University of Colorado at Colorado Springs	Introduction to Software Engineering	Sch of Eng & Appl Science Dept of Comp Sci	Colorado Springs	Colorado	80933	Dr. Robert Sebesta	303-593-3325
211	University of New Mexico/Los Alamos	Intro to Software Engineering	Department of Computer Science	Los Alamos	New Mexico	87544	Ms Angela Coop	505-662-5919
212	Washington University	1-Prog Systems and Language 2-Software Eng Workshop 3-Modular Programming	Sever Inst of Tech Dept of Comp Sci	St. Louis	Missouri	63130	Dr. Grula-Catalin Roman	314-889-6190
213	National University	Software Engineering Master's Program where students are required to know Ada -two month Ada course		San Diego	California		Richard Ridhle	415-858-1551
214	North Arizona State	Courses for sophmores and juniors			Arizona			
215	University of Connecticut	Graduate Level courses			Conneticut			
216	George Washington University	Undergraduate level: 1-Operating Systems 2-Programming & Data Struc 3-Theory of Comp Translators	Computer Science Department		Washington, D.C.		Shmuel Rotenstreich Michael Feldman	202-994-5252 202-994-5253
217	George Washington University	Graduate level: 1-Design of Translators 2-Comparative Prog Languages 3- Concurrency & Parallelism	Computer Science Department		Washington, D.C.		Michael Feldman	202-994-5253
218	University of Connecticut	Graduate level courses	Computer Science Department		Connecticut			
219	Oklahoma State University				Oklahoma			
220	University of Missouri - Rolla	Introduction to Ada	Computer Science Department	Rolla	Missouri			

Universities Teaching Ada

	University	Courses Provided	Address	City	State	Zip Code	Point of Contact	Phone Number
221	Kansas State	Graduate Level courses	Computer Science Department		Kansas			
222	University of North Carolina	Undergraduate level course	Computer Science Department	Charlotte	North Carolina			
223	University of Washington	Both graduate and undergraduate courses	Computer Science Department		Washington			
224	Stockton State College			Stockton	New Jersey			
225	Manatee Community College	Introduction to Ada	5840 26th Street West	Bradenton	Florida	34207	Dr. Donald P. Purdy	813-755-1511
226	Capital University	Ada Programming	Computer Science	Laurel	Maryland		Jack Bieler	703-941-8888 301-596-0161

Appendix F - Software Design Paradigms

Software engineering currently employs a variety of paradigms in the development of software. A "paradigm" is a mechanism that illustrates a concept through the use of an example or idea that is commonly understood. These paradigms, which are used throughout the software lifecycle, provide a particular perspective of the software process. A couple of issues arise in the use of these paradigms. Is there an advantage to using the same paradigm consistently throughout the lifecycle? And secondly, is there a paradigm for software development that is superior to the others?

There are three major categories of paradigms we are considering: (1) object-oriented, (2) process-oriented, and (3) behavior or state-oriented. The object-oriented paradigm allows the software engineer to structure software around the conceptual objects of the system. Objects possess attributes and have specific functions associated with them. A process-oriented paradigm takes a functional view, highlighting system processes and data flows between those processes. A behavior-oriented paradigm provides a view based upon the system states. Objects and processes do not have to be explicitly defined in a state-based notation.

The idea of three complementary views or paradigms has been noted in both the design and requirements community. Buhr (Buhr,91) notes the existence of the structural, functional, and temporal "domains." These domains correspond to the categories of paradigms, where the structural is the object-oriented, the functional is the process-oriented, and the temporal is the behavior-oriented. Rumbaugh (Rumbaugh,91) also notes that a system can be viewed with an "object model, dynamic model, or functional model."

Techniques within the object-oriented paradigm are object-oriented design (OOD)(Booch,87) and object-oriented requirements analysis (OOA) (Coad,90). Popular techniques within the process-oriented paradigm are structured analysis (Yourdon,89) and structured design. Behavior-oriented techniques include finite state machines, Statecharts (Harel,87) and Petri nets.

One of the major advantages of using Ada is the ability to design software in an object-oriented fashion. This approach allows a software engineer to produce software that hides many of the "implementation details." Given the use of OOD, should we employ an object-oriented perspective during requirements? Not entirely. The object-oriented

paradigm serves a useful role in managing software complexity during the design and implementation stages. However, an object-oriented perspective alone is not sufficient to describe requirements adequately. OOA, like its counterpart, Structured Analysis, provides the requirements reader a picture of the system objects and processes. While this is useful, we still need a way of describing the behaviors required by the implemented system. For this, we use a state machine or Petri net. Structured Analysis and OOA use some form of a state machine (finite state machine, state-event-response table) for defining the timing and behavioral requirements of a system. This use of a state machine is not part of the primary notation for either of these techniques but is an augmentation.

In addition, the goals for the different phases are not the same. During design, we want to define a structure to our software that hides unnecessary detail, promotes reliability by defining interfaces explicitly, and supports modifiability by localizing the possible changes. During requirements, we want to ascertain and describe all the desired functionality, features, and behaviors of a system that are externally visible to the user(s) and/or to other systems. From a pure requirements standpoint, we should not know how the system will be implemented (Davis, 90).

Thus, we should employ a variety paradigms (i.e., perspectives) during the requirements definition phase. And the choice of paradigms(s) should be based upon the demands of the system itself, not necessarily the intended design and implementation technique.

References

Booch, Grady, Software Engineering with Ada, Benjamin/Cummings, Menlo Park, CA, 1987.

Buhr, R.J.A., et al., "Support for Specifying Temporal Behavior in Ada Designs," Ada Letters, Vol. 11 No. 3, ACM Press, Spring 1991.

Davis, Alan M., Software Requirements Analysis and Specification, Prentice-Hall, Englewood Cliffs, NJ, 1990.

Harel, David, "Statecharts: A Visual Formalism for Complex Systems," Science of Computing, pp. 231-74, 1987.

Rumbaugh, James et al., Object-Oriented Modeling and Design, Prentice-Hall, Englewood Cliffs, NJ, 1991.

Appendix G - Tables to Support Findings

Contents

Key for tables:

o	Support provided
s	Some support provided
o	Support expected within the next 18 months
B	Bridge to independent tool
T	Templates
n/a	Not applicable

Company	Contact	Product	Price (base)	Development Activities Supported										
				Planning	System Specification	System Simulation	Software Specification	Prototyping	Software Design	Code Generation	DataBase Design	User Interface Design	Reverse Engineering	Testing
ATI, Inc.	(212) 354-8280	superCASE, XL/superCASE	$8,000			B			o	o			o	
Ascent Logic Corp.	(408) 943-0630	RDD System Designer, DVF	$42,000		o	o						o		
Athena Systems, Inc.	(408) 730-2100	Foresight	$13,900			o	o	o						
CCC Co.	(215) 854-0555	NETworkbench	$25,000			o	o	B	o	o		o	o	o
CCC Co.	(215) 854-0555	MODEL	$25,000						o	o	o	o		o
CSC	(703) 876-1223	Design Generator	$995				o		o			o		
Cadre Technologies Inc.	(703) 875-8670	Teamwork family	$10,000	s		o	o		o	o		o	o	o
Cadware Group	(203) 397-2908	System Developer I	$499	o			o		o		o	o		
Cadware Group	(203) 397-2908	System Developer II	$3,499	o			o		o		o	o		
Computer Systems Advisors	(201) 391-6500	POSE family	$3,995	o			o		o	o	o	o	o	
Computer Systems Advisors	(201) 391-6500	SILVERRUN	$7,500				o				o	o		
Easyspec, Inc.	(713) 480-6606	Object Plus	$2,490				o		o	o	o		o	
Future Technologies, Inc.	(206) 939-7552	Envision	$7,500			o	o	B	o	s	o/B	B	o	
i-Logix, Inc.	(617) 272-8090	Statemate family	$75,000		o	o	o	o	o					
IDE	(703) 848-8808	StP family	$5,000				o		o	o	o			
Iconix	(703) 391-2771	PowerTools/Adaflow	$8,990				B	o/B		o	o	o		o
Index Technology Corp.	(800) 777-8858	Excelerator	$9,800	B			o		o/B		s	o/B		
Integrated Systems, Inc.	(408) 980-1500	AutoCode	$20,000		o	o				o		o		
KnowledgeWare	(703) 506-0823	ADW, /RAD	$17,000	s			o	o	o	o/B	o	o	o	
LBMS	(800) 333-6382	Structured Architect Workbench	$6,995				o							
LBMS	(800) 333-6382	System Engineer	$7,500	o			o		o	o	o	o	o	
LPS srl	39 11 885 934	KeyOne	$895						o	o	s		o	
Mark V Systems Limited	(818) 995-7671	ObjectMaker, Adagen, Cgen	$8,000				o		o	o		o	o	

Company	Contact	Product	Price (base)	Development Activities Supported										
				Planning	System Specification	System Simulation	Software Specification	Prototyping	Software Design	Code Generation	DataBase Design	User Interface Design	Reverse Engineering	Testing
Mentor Graphics, Inc.	(714) 660-8080	CASE Station	$25,000				o	o	o	o			o	o
Nastec	(800) 872-8296	CASE 2000	$9,900	o			o		o	B	B		o	
Popkin Software	(212) 571-3434	System Architect	$1,890	o			o/B	o	o	o		o	o	
ProMod, Inc.	(800) 255-2689	ProMod family	$10,000				o		o	o				
RJO	(301) 731-3600	Auto-G	$31,500		o	o	o		o	o				o
Reasoning Systems	(415) 494-6201	Refine	$9,900				o/B	o		o		o	o	o
SES	(919) 881-2144	SES/workbench	$36,000		o	o	B			o				
SPS	(212) 686-3790	EPOS	$14,785	o	o	o	o		o	o		o		
SPS	(407) 984-3370	Classic Ada	$4,500						o	o				o
Scandura Intelli,ent Systems	(215) 664-1207	re/NuSys Workbench	$2,800				o	o	o	o		o	o	o
Semaphore Tools	(508) 794-3366	Pilot	$5,000				o	o	o	o	o	o	o	
Softlab, Inc.	(415) 957-9175	Maestro	$13,000				o		o	B				
Software Systems Design, Inc.	(714) 625-6147	AISLE toolset	$37,500				B		o				o	o
Systemetica Ltd.	44 202 297292	HOOD-SF	7,000 st				o		o	o				
Systemetica Ltd.	44 202 297292	SSADM-SF	7,000 st				o		o		o			
TRW	(205) 830-3606	DCDS	Free	s	o	o	o	o	o					
Teledyne Brown Engineering	(703) 352-8500	TAGS/RT	$19,000	s	o	o	o		o					
Texas Instruments	(703) 849-1481	IEF/IEW	?	o			o		o		o	o	o	
Verilog S.A.	(301) 220-2430	AGE	$50,000	o	o	o	o	o	o	o			o	o
Visible Systems Corp.	(617) 969-4100	VA Workbench	$3,395				o		o	o	o	o		
Yourdon, Inc.	(415) 871-2800	Analysis/Design Toolkit	$1,995				o		o		o	o		
Yourdon, Inc.	(415) 871-2800	Cradle	$1,995				o		o	o		o		

TABLE 1. CASE Tools

Tool	Requirements Extraction	Requirements Traceability	Software Development																DataBase Dev			If Dev	
						Spec Methods						Des Methods				Code Gen			Methods				
			Resource alloc	Timing Info	Simulation	Structured Analysis	SA augmented	Object-Oriented Analysis	Other	Prototyping	Auto DB Population	Structured Design	Object-Oriented Design	Other	Auto Design Gen	Ada	C++	Other	Chen	Other	DB Schema/Code Gen	Screen/Forms Design	Prototyping
ADW, /RAD		o		o	o		o		o	o	o	o		o	o	o		T		o	o	o	o
AGE		o	o	▪	o			o	o		o		o			o	o	o		▪	o	▪	
AISLE family		o										o	o			o							
Auto-G			▪	▪	o				o		o					o							
AutoCode		o	o	o	o				o		o					o	o	o					
CASE 2000			o	▪		o	o		o		▪	o		o	o	o		B	o		o	o	
CASE Station						o	o				o	o						T	o			o	
Classic Ada											o		o			o							
Cradle	o	o	o	o		o	o				o	o				o		o	o			o	
DCDS		o	o	o	o				o	o	o			o		o						o	
Design Generator		o				o	o				o								o				
EPOS		o	o	▪	o	o	o		o		o	o		o	o	▪		▪		o		o	
Envision	▪	▪			o	o	▪			o	o	o						T	o		B	B	B
Excelerator		o				o	o		o	o	o	o		o	o				B	o	o	o	o
Foresight			o	o	o	o	o			o	o												
HOOD-SF		o	o					o			o		o		o	o							
IEF/IEW																							
KeyOne									o		o			o	o					o	o	o	o
MODEL						B			B			o	o			o	o	o					
Maestro	o	?		?		o			o		o	o						B					
NETworkbench			o	o	o	B	B			o	o			o		o		o		o	o	▪	▪
ObjectMaker						o	o	o	o		o	o	o	o		o	o	o	o	o	o	o	o
Object Plus		o						o	o				o			o	o	o		o	o	o	

Tool	Requirements Extraction	Requirements Traceability	Software Development																DataBase Dev			If Dev	
						Spec Methods						Des Methods				Code Gen			Methods				
			Resource alloc	Timing Info	Simulation	Structured Analysis	SA augmented	Object-Oriented Analysis	Other	Prototyping	Auto DB Population	Structured Design	Object-Oriented Design	Other	Auto Design Gen	Ada	C++	Other	Chen	Other	DB Schema/Code Gen	Screen/Forms Design	Prototyping
POSE						o					o	o						B	o	o	o	o	o
Pilot				o				o		o	o		o				o	o		o	o	o	o
PowerTools/AdaFlow		o			B	o	o				o	o	o		o	o	o	o	o	o	o		
ProMod toolset		o				o	o				o	o	o	o	s	T		T					
RDD-100 family	B	o	s	o	o				o	o					s					o		s	
Re/NuSys Workbench			o		o				o					o		o	o	o		o		o	o
Refine		o			o	o	o	o		o	o	o				o	o	o		o		o	o
superCASE		B		s		B	B				o	o	o			T		T					
SA Workbench		o	s			o	o		o		o												
SSADM-SF		o				o			o		o	o								o			
SES/workbench		o	o	o	o	o	o	o	o		o					o	o					o	
SILVERRUN						o					o							o		o		o	o
StP		o	o			o	o				o	o		o		T	B	T	o	o	o	o	o
Statemate		o	o	s	o				o	o	s			o		o		o				s	o
System Architect	o	o	o			o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
System Developer I						o	o		o		o	o							o	o	o	o	o
System Developer II	o	o		o		o	o		o			o							o	o	o	o	o
System Engineer		o				o	o		o		o	o		o	s	o	o	o		o	o	o	o
Teamwork/RqT	o	o	o		o	o	o		o		o	o	o		o	o	B	o		o	s	o	
TAGS/RT	o	o	o	o	o				o	o				o									
VA Workbench						o					o	o			o			o	o		o	o	o
Yourdon ADT	s	s				o	o				s	o							o		o		

TABLE 2. Methodology Support

Product	Platforms			Imp. Lang			Vendor Support			Mark. Since (USA)	Cust. Base	Internal Storage				Report Generation						Adap	
																Style			O/P Format				
	PC	Workstation	Mainframe	Ada	C++	Other	Training/Cons	User Group	Newsletter			Database	Repository	Multi-user	Network	Fixed	User-definable	2167 support	PostScript	Desktop Pub II	Plotting	Methodology	Data Model
ADW, ADW/RAD	o					o	o	o	o	1990	>3k inst		o	o	o		o	o	o			s	
AGE		o				o	o		o	1987	>1k lic			o	o		o		o	o			
AISLE family	o	o				o	o	o		1985	45 org	o		o	o		o	o	o	o			
Auto-G	o	o	o			o	o	o		1989	25 inst*			o	o	o		o	o		o		
AutoCode	o	o			o	o	o	o	o	1986	>600 inst	o		o	o	o	o	o	o	o			
CASE 2000	o	o				o	o	o	o	1981	Not avail.	o		o	o		o	o	o	o		s	
CASE/CodeLink Stations		o			o		o	o	o	1984	>3k users	-	-	o	o		o	o	o			s	
Classic Ada		o	o	o			o		s	1989	>50 inst	o		o	o	o	o						
Cradle		o				o	o		o	1990	6-7 org*	o	o	o	o		o	o	o		o		
DCDS		o		o			o		o	1987	>200 inst	o	o	o	o		o	o	o			s	o
Design Generator	o					o				1987	<100 inst					o			o				
EPOS	o	o	o			o	o	o	o	1984	500 lic*	o		s			o	o	o				
Envision		o				o	o	o	o	1986	>600 inst	o	o	o	o		o					s	
Excelerator family	o	o			o		o	o	o	1984	100k inst	o	s	o	o		o	o	o	o	o	s	s
Foresight		o			o		o			1988	20 org	o	o	o	o		B		o	o	o		
HOOD-SF	o	o		?	?	?	o	?	?	?	?	-	o	s	o	o	o		o	o	o	o	o
IEF/IEW		o	o		o		o		s	1986	>350 org	o	s		o		o		o				o
KeyOne	o	o	o			o	s			1987	>600 inst			o	o		o	o	o		o		
MODEL			o	o		o	o			1981	<5 inst	-	-	-	-		o						
Maestro II	o	o	o			o	o		o	1978	23k lic*	?	o	o	o	o	?	?	o			o	
Networkbench		o	o	o		o	o			1990	5 inst	o	o	o	o		o						
ObjectMaker family	o	o		o		o	o	o	o	1986	500 inst	o	o	o	o	o	o	B	o	o	o	o	o
Object Plus	o					o	o		s	1989	700 lic	o	o	o	o		o	o				s	
POSE	o	o				o	o		o	1982	>2k users*	o		o	o		o				o		

Product	Platforms			Imp. Lang			Vendor Support			Mark. Since (USA)	Cust. Base	Internal Storage		Multi-user	Network	Report Generation						Adap	
																Style			O/P Format				
	PC	Workstation	Mainframe	Ada	C++	Other	Training/Cons	User Group	Newsletter			Database	Repository			Fixed	User-definable	2167 support	PostScript	Desktop Pub If	Plotting	Methodology	Data Model
Pilot	o	o			o		o		o	*1991*	n/a	o	o	o	o		o	o		o		o	o
PowerTools/AdaFlow	o	o				o	o			1986	1.5k lic			o	o		o	o		o			
ProMod family	o	o	o			o	o			1985	500 lic	o		o	o	o		o	o				
RDD-100 family	o	o				o	o	o	o	1988	250 lic	-	B	o	o		o	o	o	o		s	o
Refine		o			o	o	o		o	1985	>100 lic	o	o		o		o		o			o	o
re/NuSys Workbench	o	o				o	o			1989	100 lic	-	*B*	o		o					s		
superCASE family		o	o			o	o			1987	>100 inst	o		o	o		o	o	o			s	
SA Workbench	o					o	o			1990	300 lic	o	o				o	o	o				
SES/workbench		o			o	o	o	o	o	1989	>100 inst	-	-	o	o		s		o				
SILVERRUN	o				o		o	o	o	1988	>3k lic	o		*o*	*o*			o					
SSADM-SF	o	o		?	?	?	o			?	?	-	o	s	o	o	o		o	o	o	o	o
Statemate family	o	o	o			o	o			1987	700 lic		o	o	o		o	o	o	o	o		
SiP		o		o	o	o	o	o	o	1985	4k inst	o	o	o	o		o	o	o	o	o	s	s
System Architect	o					o	o			1988	>5k lic	o		o	o		o		o	o		o	
System Developer I	o					o	o			1984	5k users	o		o	o	o			o			o	
System Developer II	o					o	o			*1991*	n/a	o	o	o	o		o		o			o	
System Engineer	o			?	?	?	o		o	1990	n/a	o	o	o	o	?	?	?	?	?	?	?	?
TAGS/RT		o				o	o	o	o	1984	Not avail.	o	o		o	o	s	B	o	o			
Teamwork family	o	o				o	o	o		1982	15k lic	o	o	o	o		o	o	o	o	o		s
VA Workbench	o					o	o		o	1985	>3k inst	o	o	o	o	o			o				
Yourdon A/D Toolkit	o					o	o		o	1984	4k lic	o				o			o		o		

TABLE 3. General Information

Company	Contact	Product	Subject CASE
Cadware Group, Ltd.	(203) 397-2908	Foundry	System Developer family
Index Technologies	(800) 777-8858	Customizer	Excelerator
Mark V Systems Ltd	(818) 995-7671	Tool Development Kit	ObjectMaker
Mark V Systems Ltd	(818) 995-7671	Menu Customization Kit	ObjectMaker
Reasoning Systems	(415) 494-6201	Refine	Product family
Systematica Ltd.	44 202 297 292	VSF Analyst/Designer Workbench	VSF Methods Workbench

TABLE 4. Customization and Meta CASE Tools

Company	Contact	Product	Language Support	
			Available	Planned
Cadware Group, Ltd.	(203) 397-2908	User Interface Prototyper	COBOL	
CinCom Systems Ltd.	(800) 543-3010	MANTIS	COBOL, PL/1	
Cortex Corp.	(617) 622-1900	CorVision	COBOL	
ExperTelligence	(805) 967-1797	Action!	C	C++
Intersolve	(800) 777-8858	PVCS, APS	COBOL	
KnowledgeWare	(703) 506-0823	IEW/GAMMA	COBOL	
Micro Focus, Inc.	(415) 856-4119	COBOL/2 Workbench	COBOL	
Netron, Inc.	(416) 636-8333	CAP	COBOL	
Relational Team Concepts	(713) 622-7400	TOP*CASE	Oracle	
Pansophic	(800) 323-7335	TELON, TELON/PWS	COBOL, PL/1	
Pansophic	(800) 323-7335	Panel Painter	COBOL, PL/1	
SINC, Inc.	(201) 391-6500	Flexgen	COBOL	
SSA, Inc.	(708) 850-9192	AS/SET	RPG/400	
Sage Software, Inc.	(800)547-4000	APS Development Center	COBOL	
Software A&E	(703) 276-7910	SNAP/KES (formerly Spectrum)	C	
Software AG N. Am	(703) 860-5050	Predict	Natural	
Software One, Ltd.	44 0628 850444	Clarion	?	
SysCorp International	(800) 727-7837	MicroSTEP	C	
Unify Corp.	(916) 920-9092	ACCELL	COBOL	
Unisys	(215) 993-6166	LINC II	?	
Vleermuis Software Research bv	31 30 31 04 26 fax	GUI_MASTER	C++	

TABLE 5. Application Generators

From CASE Tool / To CASE Tool	ADW	Auto-Mate Plus	Adagen	Design/1 Case Tool	Data Analyst	Excelerator	IEF/IEW	IEW/GAMMA	StP	Teamwork	Saber-C C++ Gen	START	QASE
ADW, /RAD				o		o	o						
IEW	o	o	o	o	o	o	o	o		o			
AISLE family						o			o	o			
MODEL										o			
Maestro								o					
NETworkbench										o			
PowerTools													o
superCASE						o							
SES/workbench									o				
StP											o	o	
System Architect						o	o						
System Developer I										o			
System Developer II										o			
Teamwork									o	o			

TABLE 6. Bridges Between CASE Tools

Company	Contact	Product	Language Support	
			Available	Planned
ASA, Inc.	(214) 245-4553	Hindsight	C	Ada, C++
ATI, Inc.	(212) 354-8280	superCASE SCI	FORTRAN	
Bachman Information Systems	(617) 273-9003	Bachman Product Set	COBOL	
Catalyst Group	(703) 698-5100	XPERT series	COBOL	
IDE	(703) 848-8808	CDE		Ada, C, C++, Pascal
InterPort Software Corp.	(703) 425-6425	InterCASE	COBOL	
Intersolve	(800) 777-8858	PVCS, APS	COBOL	
Language Technology, Inc.	(508) 741-1507	(IEW Tool)		COBOL
McCabe	(800) 638-6316	BAT, CodeBreaker	Ada, C, COBOL, FORTRAN, Pascal	
Nastec	(800) 872-8296	Source/Re	COBOL	
ParcPlace Systems	(415) 691-6700	ObjectWorks/C++	C++	
Procase	(408) 727-0714	SMARTsystem	C	C++
Reasoning Systems	(415) 494-6201	CLS	C	
Reasoning Systems	(415) 494-6201	Ada/RevEng	Ada	
SPS	(212) 686-3790	RE-SPEC	FORTRAN, Pascal	Ada, C, COBOL
ViaSoft, Inc.	(602) 952-0050	Via/Center	COBOL	

TABLE 7. Independent Reverse Engineering Tools

Company	Contact	Product	Function	Language Support
ABRAXAS Software Inc.	(800) 347-5214	CODE CHECK	Stle analysis	C, C++
Cadre Technologies Inc.	(703) 875-8670	SAW	Coverage/perf analysis	Ada, C
Computer Associates	(203) 627-8923	TRAPS	Regression testing	Independent
Donatech Corporation	(515) 472-7474	Realtime Testware	Regression testing	Independent
Dynamics Research Corp.	(508) 475-9090	AdaMAT	Quality analysis	Ada
EVB Software Engineering Inc.	(800) 877-1815	DYN	Complexity analysis	Ada
General Research	(805) 964-7724	AdaQUEST	Coverage, quality analysis	Ada
Intermetrics, Inc.	(714) 891-4631	TST	Dynamic analysis support	Ada
McCabe	(800) 638-6316	Start	DFD-driven testing	Independent
McCabe	(800) 638-6316	ACT	Complexity analysis	Ada, C, COBOL, FORTRAN, Pascal
Nokia Data	358-31-237317	TBGEN	Test bed generation	Ada
Nokia Data	358-31-237317	TCMON	Coverage analysis	Ada
Programming Environments Inc.	(201) 918-0110	T	Test data generation	Independent
RTP Software Ltd	(0252) 711414	MALPAS	Static analysis	Ada, Pascal
Set Labs	(503) 289-4758	UX-METRIC	Quality analysis	Ada, C++, C
Set Labs	(503) 289-4758	PC-METRIC	Quality analysis	Ada, C, others
Software Research Inc.	(415) 957-1441	SMARTS family	Regression testing	Independent
Software Research Inc.	(415) 957-1441	TCAT series	Coverage analysis (branch)	Ada, C, COBOL, FORTRAN, Pascal,
Software Research Inc.	(415) 957-1441	TCAT-PATH	Coverage analysis (path)	Ada, C, FORTRAN, Pascal
Software Research Inc.	(415) 957-1441	SCAT	Coverage analysis (system)	Ada, C, A
Software Research Inc.	(415) 957-1441	TSCOPE	Coverage animation	Coverage analysis
Software Research Inc.	(415) 957-1441	TDGEN	Test data generation	Independent
Software System Design	(714) 625-6147	TestGen	Coverage analysis	Ada, C
Teledyne Brown Engineering	(205) 726-1613	ACAT	Complexity analysis	Ada
Teledyne Brown Engineering	(205) 726-1613	SMART	Quality assurance	Ada
Verilog S.A.	(301) 220-2430	Logiscope	Coverage analysis	Ada, C, COBOL, FORTRAN, Pascal
XA Systems Corp.	(800) 344-9223	PATHVU	Quality analysis	COBOL

TABLE 8. Stand-Alone Testing and Measurement Tools

Company	Contact	Product
AGS Management Systems	(800) 678-8484	firstCASE
ASTEC	(301) 441-9036	Camera
Atherton Technology	(301) 961-1526	Backplane
Cadre Technologies Inc.	(703) 875-8670	Teamwork/IPSE toolkit
Cincom Systems, Inc.	(800) 888-0115	AD/Advantage
General Research Corp.	(805) 964-7724	SLCSE
IBM		AD/Cycle
Pansophic	(800) 323-7335	TELON/Teamwork

TABLE 9. Integration Frameworks

Company	Contact	Product	Type	Comment
Adpac Corp.	(415) 974-6699	Design	Analysis/Design tool	Sending 5/13
Arthur Anderson	(312) 5070-5161	Design/1 CASE Tool	Analysis/Design tool	Sending 4/26
Carleton University	(613) 788-5718	TimeBench	Analysis/Design tool	Prototype
Cognos	(617) 229-6600	Powercase	Analysis/Design tool	Sending 5/13
D. Appelton Company	(213) 546-7575	IDEF/Leverage	Analysis/Design tool	No answer
Michael Jackson Software	(44)71 286-1814	Jackson Workbench	Analysis/Design tool	No answer
Thought*Tools		SCOOP-3	Analysis/Design tool	No answer
On-Line Software Inter	(201) 592-0009	Casepac	Analysis/Design tool	Sending 5/13
Sterling		miniAssist	Analysis/Design tool	Sending 5/5
Toes Software	(800) 777-4316	Application Xcellence	Analysis/Design tool	Sending 5/13
Westmount Technology	(31)15-610815/(914)294-661	ISEE- TSEE, RTEE	Analysis/Design tool	Sending 5/14
Apollo (now HP)	(800) 227-6556	DSEE	Configuration management	
CaseWare, Inc.	(714) 754-0308	Amplify	Configuration management	
Pansophic	(800) 323-7335	PAN/L [illegible]	Configuration management	
ProMod, Inc.	(800) 255-2689	ProMod/CM	Configuration management	
Procase	(609) 452-8848	Procase	Configuration management	
Softool	(805) 683-5777	CCC family	Configuration management	
Software Main & Dev Sys	(508) 369-7398	Aide de Camp	Configuration management	
SQL Systems International	England 44-279-641021	PCMS*ADA	Configuration management	
ASYST Technologies	(800) 361-3673	The Developer	Database tool	
Informix Software, Inc.	(415) 322-4100	Informix-ESQL/Ada	Database tool	
Ontologic	(508) 667-2382		Database tool	
Oracle Systems Corp	(415) 506-7000	Case* family	Database tool	
SQL Solutions	(416) 249-2246	Deft	Database tool	
Advanced Logical Software	(213) 653-5786	Anatool	Diagram editor	
Ascent Technologies, Inc.	(415) 940-1550	MetaView	Diagram editor generator	
Cadware Group, Ltd.	(203) 397-2908	Sylva	Diagram editor	
Digital Insight	(303) 674-5232	Robochart	Diagram editor	
Meta Software Corp.	(800) 227-4106	MetaDesign, Design/IDEF	Diagram editor	
Software Originals, Inc.	(800) 873-6873	MacSTILE	Diagram editor	
TNO	31 15 697 071	Configurable Graphical Editor	Diagram editor generator	
Tata Consultancy Services	(408) 720-9584	Essay	Diagram editor	

Company	Contact	Product	Type	Comment
Caine, Faber, & Gordon, Inc.	(818) 449-3070	PDL/81	PDL tools	
Data General	(508) 366-8911		PDL tools	
Encore Computer Corp.	(301) 499-4700		PDL tools	
Flexible Computer Corp.	(214) 869-1234		PDL tools	
Gilmore Aerospace	(404) 728-0312		PDL tools	
GTE Government Systems Corp.	(617) 449-5000		PDL tools	
IBM SID	(301) 493-1448		PDL tools	
Incremental Systems Corp.	(412) 621-8888		PDL tools	
Intelligent Choice, Inc.	(213) 379-9680		PDL tools	
Intermetrics, Inc.	(617) 661-1840	Bryon	PDL tools	
Loral/Rolm Mil-Spec	(408) 423-7701		PDL tools	
Phoenix International	(213) 568-1740		PDL tools	
RAMTEC	(201) 477-8248	PDL Tool KIT(TM)	PDL tools	
Sanders Associates	(603) 885-9208		PDL tools	
SofTech	(617) 890-6900		PDL tools	
ABT Corp	(212) 219-8945	Project Workbench	Project management	
American Manangement Systems	(703) 841-6000	Life-Cycle Productivity Systems	Project management	
Claris		MacProject II, SmartForm Manager	Project management	
Deloitte, Haskins, & Sells	(704) 377-3560 x3131	4Front	Project management	
Index Technology Corporation	(800) 777-8858 x739	PC Prism	Project management	
Project Software & Development, Inc	(301) 231-8660	Project/2	Project management	
Software Publishing Corp.		HTPM	Project management	
AST, Inc.	(303) 790-4242	Qase (PerSpective)	Sys perf analysis	
Active Memory Technology, Inc.	(714) 261-8901	DAP 500 Program Manag/Sim		
Chen & Associates	(504) 928-5765	ER-Designer	Database tool	
Cullinet Software, Inc.		IDMS/Architect	Database tool	
Digital		Epitool	Expert system development	
EVB Software Engineering Inc.	(800) 877-1815	GRACE library		
GSI-Danet, Inc.	(703) 471-7130	OSIPRO	OSI development	
Holland Systems Corp.		Logical Database Design (LDD)	Database tool	
IDDK Software		Intelligent Database Design (IDDK)	Database tool	
IWG Corporation	(619) 223-5444	Poplink	Communication analysis	
Information Engineering Systems Ltd.		USER	Expert system development	

Company	Contact	Product	Type	Comment
Interact	(212) 696-3700	Integrator	CAE tool	
Interactive Software Engineering, Inc.	(805) 685-1006	Eiffel	Programming environment	
JADE Simulation Inter. Corp.	(804) 744-5849	JADE family	Simulation environment	
Mass Tech, Inc.	(205) 539-8360		Ada libraries	
Polyhedron Software, Ltd.	(44) 865 300579	SPAG	Formatter	
Programming Research, Ltd.	(44)81 942 9242	Flint	Rev engineering	No US contact
Quintus Computer Systems, Inc.	(415) 965-7700	Prolog Integrated Environment	Programming environment	
Simulation Software	(519) 657-8229	GP	Simulation environment	
Tartan Laboratories, Inc.	(412) 856-3600	Ada Scope	Ada analysis tool	
Unicad, Inc.	(800) 331-3729	UIMS, X-Pression	User interface tool	
Unirel	+39 55 301279	Unirel Openlook Toolkit	User interface tool	
Wolverine Software Corp.	(703) 750-3910	GPSS/H	Programming environment	
Xinotech Research	(612) 379-3844	Program Composer	Ada analysis tool	

TABLE 10. Other Tools

Company	Contact	Product	Comment
Ad CAD, Inc.		STATEMENT1	Now i-Logix
Ada Technology Group, Inc.	(202) 387-5715		# disconnected
Andyne Computing, Ltd.		ACPVision, ERVision	No contact point
Associative Design Technology, Ltd.		PadTech	No contact point
TBHA Computer Pty. Ltd.		Semantic Data Dictionary (SDD)	No contact point
CASET	(714) 496-8670	IPSYS Tool Building Kit	No answer
CGI Systems, Inc.	(703) 684-6166	PacBase, PacBench, PacDesign	No answer
Cap Gemini Software Products		*MULTI-PRO*	No contact point
Charles Stark Draper Laboratory		DARTS	No contact point
Computer Assoc Inter	(516) 227-3300	*CA- series*	Sending 5/13
Computer Data Systems	(202) 921-7000	Scan/COBOL, Super Structure	Not contacted
Computer Corporation of America		ACCOLADE	No contact point
Compuware Corp		CATI tools (COBOL)	No contact point
Datamat	(619) 431-5610		No answer
Dialogic Systems Corp.		Dialogic Development Center Workbench	No contact point
Eclectic Solutions Corp.		COINS	No contact point
Eliot Consultants	(213) 439-7021		
Excel Software		MacDesigner	No contact point
General Dynamics		TASKIT Tasking Ada Sim Kit	No contact point
Hamilton Technologies, Inc.		001/AXES	No contact point
IGL Technology, Inc.	SPECIF-X		No contact point
International Computers Ltd.		Quick Build Workbench	No contact point
Interprogram B.V.		BLUE	No contact point
Language Technology	(508) 741-1507	Recoder, Inspector	Not Ada, C++
Leading Software Technologies		The Intelligent Assistant TIA	No contact point
M. Bryce & Associates		PRIDE-ISEM	No contact point
Manager Software Prod, Inc.	(617) 863-5800	Manager series	No answer
Matterhorn, Inc.		HIBOL	No contact point
McDonnell Douglas Information Sys.	(800) 225-7760	ProKit Workbench	Not contacted
Menlo Business Systems, Inc.		Foundation Vista	No contact point
Meta Software		Design/family	No contact point
Phoenix Technologies Ltd	(617) 551-4000		No longer supported
Ready Systems	(800) 228-1249	ART series, CARDtools	IPSE

Sapiens Inter	(212) 366-9394	several	No answer
Sequent Computer	(503) 526-4153		# disconnected
Servi Logic	(415) 748-6200	GemStone	Not contacted
Six Sigma Case	(202) 643-6911	Canonizer	Not contacted
Software Design Tools		Xtools	No contact point
Software Leverage	(617) 648-1414		No products
Software Productivity Research		SPQR/20 Estimator	No contact point
Syscon Corporation	(619) 296-0085	family	Proprietary
SystemOID, Inc.		Consoi	No contact point
TASC	(617) 942-5000	AdaGRAPH	Wrong #
TXT Ingegneria Informatica	39 2 27991991		No US contact
Telelogic Europe	011 32 3 647 3670	SDT	No US contact
Ultraware, Ltd.		Development Management System (DMS)	No contact point
Visual Software, Inc.	(301) 230-3200	vsDesigner, etc.	No longer supported
Westinghouse Electric Corp.		SIDE	No contact point

Information From: Gonen Ziv (212) 354-8280, May 7 1991.

Address: Advanced Technologies, Inc, 305 5th Avenue, Suite 2420, New York, NY 10118

Tool Summary: Back end CASE tool.

1. **Hardware Platforms:** VMS based for VAX mainframe, microVAX, VAX clusters etc.
2. **Products:** superCASE and superCASE SCL licensed per machine.
   i. superCASE from $8,000 to $90,000.
   ii. XL/superCASE bridge to Excelerator/RTS, provides requirements traceability $8,500.
   iii. superCASE SCI reverse engineering $5,000 to $25,000.
3. **Tool Implementation Language:** Mainly C
4. **Vendor Support:** Technical support line, training, consultancy.
5. **Marketed Since:** 1987.
6. **Size of customer base:** Over 100 installations.
7. **Methodologies/functions supported:**
   i. **Software design:** OOD Buhr, SC methods. Capture of timing information in annotations but not used. Interface consistency checked.
   ii. **Code generation:** Templates for Ada, C, FORTRAN, PL/1, PL/M, Jovial.
   iii. **Maintenance:** Re-engineering for FORTRAN.
8. **Documentation generation:** 2167A support, user-definable formats.
9. **Project management support:** Configuration management built-in and standard interface to external CM tools. Security/control access.
10. **Environment Characteristics:** Multi-user, network support.
11. **Database:** Data dictionary implemented under DEC RDB. Import/export, split/merge.
12. **Links to other tools:** See XL/superCASE.
13. **Output formats:** PostScript.
14. **User interface:** Command line, menu, on-line help, some undo. Database query facility.
15. **Adaptability:** Customizable editor.
16. **Planned enhancements:** Port to UNIX, by summer '92.
17. **Collaboration with other organizations:** Negotiating with IDE (StP).

Information From: John Cox (408) 943-0630, May 8 1991.

Address: 180 Rose Orchard Way, Suite 200, San Jose, CA 95134

Tool Summary: The Requirements Driven Development System Designer (RDD-100) is based upon the early steps of DCDS, providing an improved graphical user interface. Object-oriented approach to support library for re-usable components.

1. **Hardware Platforms:** Sun, Apollo workstations, Apple Macintosh PCs, VAXstation.
2. **Components:** Maintenance primary support $7,000, secondary support $5,000.
   i. System Designer. Equivalent to DCDS System Requirements Engineering Methodology (SYSREM) and it's System Specification Language (SSL), $36k for single user, $44,700 for network license. Volume discounts available.
   ii. RDD Design Verification Facility (RDD-DVF) for specification simulation, $11,365 for single user, $13,207 network. Provides deadlock, resource utilization, system performance, communication constraints verification and analysis. Available version 3.0.
3. **Tool Implementation Language:** Smalltalk
4. **Vendor Support:** Training, consultancy. Starting support group and newsletter.
5. **Marketed Since:** 1988, currently RDD-100 Version 2.02, version 3.0 to be released July '91.
6. **Size of customer base:** Approx. 250 licenses across 16 organizations.
7. **Methodologies/functions supported:**
   i. **System specification and design:** Some semi-automatic requirements extraction from source document. Information modeling. Some allocation of functions to hw, sw, subsystem components, some timing information captured but not all used. Traceability of system requirements and decisions. Simulation facility developed for SDIO through GE, productized for version 3.0.
   ii. **Implementation:** Forms/screen design via customizable schema.
8. **Documentation generation:** User-definable formats, also 2167 and Mil-STD-490.
9. **Project management support:** Security/control access.
10. **Environment Characteristics:** Network support but not on-line sharing between multiple users.
11. **Database:** Database import/export via ASCII, also export contextdoc pic-ed (Mentor Graphics). Database split/merge. Using external repositories (DEC, Mentor Graphics). Allows alternative designs to be stored.
12. **Output formats:** PostScript.
13. **User interface:** Menu and mouse, windowing, some undo. Database browser/query facility.
14. **Adaptability:** User-definable documentation via modification/creation of programs. User definable hierarchy charts generated from database. Additional diagnostics can be created by the report generator. User definable entities, relationships, and attributes to existing schema and to create new schema.
15. **Planned enhancements:**
   i. Version 3.0 introduces stimulus-response graphs at the system level.
   ii. Support for Interleaf.
   iii. Port to HP9000 and other HP machines, IBM RISC/AIX by end of '91.

iv. Working with 3rd party for knowledge-based support for requirements extraction.

16. **Collaboration with other organizations:**

i. DEC and Mentor Graphics.
ii. Potentially also Cadre, Iconix and others (phase new products in, starting 3rd quarter '91).

Information From: (408) 730-2100

Tool Summary: Front-end CASE, desk top simulation and modeling system for specifying and analyzing real-time embedded software.

1. **Hardware Platforms:** Sun/UNIX and HP workstations with X-Windows.
2. **Components:**
   i. Graphical Model Editor.
   ii. Model Analyzer.
   iii. Concept Prototyper.
   iv. Library elements: reusable functions and operations, mathematical and logic, signal processing, timing and validation, data manipulation, electronic I/O panel.
3. **Tool Implementation Language:** C++
4. **Tool Price:** $13,900. Training at Athena from $500 per day for 1 user to $3,000 for 6 to 10 users, on site from $1,350 for 2 days. 30 day free evaluation.
5. **Vendor Support:** Training, consultancy.
6. **Marketed Since:** September 1988. Release 2.0 due out mid-May '91.
7. **Size of customer base:** 20 customers, some of whom have multiple copies.
8. **Software specification:** Merge of Ward-Mellor and Hatley-Pirbhai methods with explicit timing information and Ada-like mini-specs. For static analysis check syntax/semantics, diagram balancing, execution readiness, diagrams/data dictionary. Interactive/batch simulation with environment model showing hardware, software, and firmware with external events. Functional and constraint modeling, tests for reachability, non-determinism, deadlock conditions, and usage of transitions. Executable model for rapid prototyping with debugging and tracing. Animation. Can include Ada code and, in version 2.0 (1) external functional calls to pull in existing C code, (2) mini-spec I/O, and (3) bidirectional translator to/from Ada and executable mini-specs, to support import of existing code. Automated database population/change propagation.
9. **Documentation generation:** via FrameMaker.
10. **Environment Characteristics:** Network support via LAN.
11. **Database:** Proprietary object management system with published data formats. Database accessed by user-written application programs.
12. **Output formats:** ASCII (during simulation), PostScript, Nroff, FMT, Runoff, Interleaf, some plotting, HPGL.
13. **User interface:** Menus and mouse, on-line help, on-line documentation, windowing, some undo.
14. **Adaptability:** General-purpose editor.
15. **Standards conformance:** X-Windows, Extended Systems Modeling Language.
16. **Planned enhancements:** User-modifiable libraries.

Information From: Mitch Bassman (703) 876-1220, John Sheffler (703) 876-1223, May 8 1991.

Tool Summary: Functions as an expert assistance that automatically translates requirements into a design generation. Knowledge-based data dictionary. Modeless operation with browsers. Object-oriented implementation supports life cycle traceability. Implements CSC's Digital System Development Methodology.

1. **Hardware Platforms:** IBM PC/AT or compatible under DOS.
2. **Tool Implementation Language:** Smalltalk/V286 from Digitalk.
3. **Tool Price:** $995
4. **Vendor Support:** Support not routinely provided.
5. **Marketed Since:** 1987, Version 2.1 released May '90.
6. **Size of customer base:** <100 installations
7. **Methodologies/functions supported:**
   i. **Software specification:** SA, Ward-Mellor methods. Chen for information modeling. Checks diagram/data dictionary consistency, prevents invalid input. Traceability. Automated database population/change propagation.
   ii. **Software design:** Design methods/diagrams: SD generated from requirements. Checks syntax/semantics, database/diagram consistency, complexity analysis. Forms/screen design.
8. **Documentation generation:** Customize contents (not format), no 2167A support.
9. **Project management support:** Some configuration management.
10. **Database:** Data dictionary implemented as file system. Import/export facility, with split/merge.
11. **Output formats:** PostScript.
12. **User interface:** Windowing, menus and mouse, on-line help, some undo. Browser/query facility.
13. **Adaptability:** Free-form text/graphics.

Information From: (703) 875-8670, May 8 1991.

Tool Summary: Environment that spans the design and implementation phases with real-time debug and verification tools. Supports automated transition of design to code, and helps to automate the maintenance of test information on-line as part of the CASE database.

1. **Hardware Platforms:** Sun, Apollo, DEC, HP workstations. Teamwork/OS/2 IBM PS/2 or Compaq under OS/2 includes Cadre's IPSE toolkit to allow adaptability such as customizing menus, accessing the database. RISC/AIX-based platforms. Compiler independent.
2. **Products:** Core environment $10,000 for 1st seat and $1,200 each additional. OS/2 version $6,500 with RT extensions extra $1,750. C/Rev and FORTRAN/Rev each $8,500. Ada/Rev $2,775. Maintenance 15%.
   i. Teamwork/IM information modeling $1,750.
   ii. Teamwork/SA for Structured Analysis $1,750.
   iii. Teamwork/SD for Structured Design $1,750.
   iv. Teamwork/ADA graphic editor for Ada program design,
   v. Teamwork/DPI document preparation interface,
   vi. Teamwork/ACCESS database utility access,
   vii. Teamwork/Menus for tailoring/extending Teamwork menus,
   viii. Teamwork/ABS an Ada source builder,
   ix. Teamwork/CSB a C source builder,
   x. Teamwork/RqT requirements traceability (previously SAIC's THOR), $15,000 for first, $7,500 for each additional.
   xi. Teamwork/SIM simulation (like Statemate). Token based simulation, $12,000 for basic interactive version, with batch and additional performance analysis facilities $19,000.
3. **Tool Implementation Language:** Mainly C.
4. **Vendor Support:** Hot-line, training, consultancy, users group.
5. **Marketed Since:** 1982, currently version 4.0.
6. **Size of customer base:** 15,000 copies.
7. **Methodologies and functions at different development stages supported:**
   i. **System specification:** Hardware/software allocation via RqT.
   ii. **Software specification:** Requirements extraction from natural English using RqT. Gane-Sarson, Yourdon-DeMarco, Ward-Mellor SA methods, and Jackson diagrams. Automatic inheritance for DFDs. Syntax/semantic, parent-child diagram balancing, consistency between diagram types, database/diagram consistency checking SIM provides simulation with performance analysis. Meller/Schlaer and ERDs for information modeling. Automated database population/change propagation. Traceability.
   iii. **Software design:** Yourdon-Constantine, Booch-Buhr and Project Technologies object-oriented methods. Show changes needed for normalization to support database design.
   iv. **Code generation:** SADMT, Ada, C, (C++ through Saber-C). Forms/screen design.
   v. **Testing:** Via Cadre's SAW product for coverage and performance analysis.
   vi. **Maintenance:** Re-engineering for C, FORTRAN.
8. **Documentation generation:** User-definable formats and 2167A.
9. **Project management support:** Configuration management, own package or via Sun's NSE, VAX/s CMS. Baselining, security/control access. Status reporting using metric from DeMarco's Bang complexity rating.

10. **Database:** Object management system, multi-tiered. Import/export, split/merging.
11. **Environment Characteristics:** Multi-user support, network support through LAN Manager (heterogenous and external control), multiple projects.
12. **Links to other tools:**
    i. Import from StP.
    ii. Athena and Softbench integration environments.
    iii. SQL report writer to access data dictionary information (3rd party).
    iv. GE tools from Ada Programmers Workbench reimplemented in Teamwork.
    v. ADAS from Research Triangle Institute.
13. **Output formats:** ASCII, PostScript, HPGL. Interface to Interleaf, Context, Scribe, Bookmaster, WordPerfect.
14. **User interface:** Windowing, menus/mouse, color, database query facility, undo facility. Database browser, on-line help.
15. **Adaptability:** Free-form graphics. User-definable database entries.
16. **Standards conformance:** CDIF.
17. **Planned enhancements:**
    i. Automatic transition from SA to SD.
    ii. FORTRAN reverse engineering.
    iii. Teamwork/T for software-based testing.
18. **Collaboration with other organizations:**
    i. General Electric Research and Development Center.
    ii. Associated with Project Technology.
    iii. PanSophic.

Information From: Rich Giordano (800) CADWARE, May 20 1991.

Tool Summary: Rule-based approach with open architecture.

1. **Hardware Platforms:** IBM PC
2. **Products:**
   i. SmartCASE basic method support without data dictionary $299.
   ii. System Deverlop I is centralized around the diagram editor, with a data dictionary/repository implemented in DB3 $499.
   iii. System Deverlop II centralized around the repository (proprietary database) to provide more flexibility $3499.
   iv. IE Information Exchange customization option (rather than a formal option). Includes IA Interaction Access option.
   v. Foundry metatool to customize the development environment (e.g., methods and user-interface) based on RuleTool, a technique using the diagram editor to create own rule-based methods $4999.
   vi. User Interface Prototyper for prototyper and COBOL source code generation $499. Available with both System Developer I and II, for II supports use of a mouse.
3. **Tool Implementation Language:** C with 8-10% assembler.
4. **Vendor Support:** Hotline, training, consultancy.
5. **Marketed Since:** System Developer I 1984, System Developer II out in June 1991.
6. **Size of customer base:** System Developer I 5000 users.
7. **Methodologies and functions at different development stages supported:**
   i. **Software specification:** Gane-Sarson, DeMarco-Yourdon, Ward-Mellor methods, also flow charts. Shlaer-Mellor, ERDs for information modeling. Requirements extraction, traceability, capture of timing information in II. Automated database population and change propagation.
   ii. **Software design:** Constantine method. Prototype for DB3 database design.
   iii. **Code generation:** Forms/screen design in COBOL
8. **Documentation generation:** User-definable formats.
9. **Project management support:** Configuration management, project planning, status reporting, change reporting, security/control access in System Developer II.
10. **Environment Characteristics:** Multi-user and network support.
11. **Database:** Merge, import and export with System Developer II.
12. **Output formats:** ASCII, PostScript, other.
13. **User interface:** Menu/mouse, on-line help.
14. **Adaptability:** Methodology tailoring. Can add menu options. Cadware Ascii Netrual Diagram Interchange (CANDI) files allow definition of own diagrams, can access by CASE tool or own code for analysis etc.
15. **Planned Enhancements:** X-Windows and OS/2 support.
16. **Collaboration:** IBM's AD/Cycle.

Information From: Evan Lock (215) 854-0555, May 20 1991.

Address: 2401 Walnut Street, Suite 402, Philadelphia, Pennsylvania 19103

Tool Summary: Uses rules and equational specification to generate engineering, real-time, distributed parallel processing software, supports testing and maintenance. Built-in intelligence for logical checking, design optimization, and self-documentation. Rapid prototyping and development. Changing name to Distributed Application Workbench. See also MODEL.

1. **Hardware Platforms:** VAX/VMS and IBM (VM/CMS, MVS/TSO) mainframes, Sun, DEC, IBM workstations.
2. **Products:** Technology transfer package (4 month license, 10 days training, 20 days consulting) for $30,000 plus travel. Range from $25,000 to $150,000 depending on environment. 25% extra for additional language. 15% annual maintenance. Components:
   i. Builder to generate Ada.
   ii. Simulator to generate Ada and C.
   iii. Manager to represent distributed run-time environment.
   iv. Configurator integrates system components to generate programs controlling initiation/termination and managing communication and control.
   v. Compiler to generate complete source language programs and produce test data for validation and debugging.
   vi. Report/Screen Generator taking pictorial input to specify reports and displays.
   vii. Test Data Generator with built-in random functions, user specifies testing rules.
3. **Tool Implementation Language:** PL/1, C, Ada, proprietary non-procedural language.
4. **Vendor Support:** Training, consultancy.
5. **Marketed Since:** 1990
6. **Size of customer base:** 4 or 5 initial sites (some government).
7. **Methodologies/functions supported:**
   i. **Software design:** Accepts DFD input from StP (DeMarco-Yourdon, Ward-Mellor, Hatley, Gane-Sarson) or textually entered in non-procedural form (rules, formulae, operations, functions, declarations). Hardware/software allocation, timing information. Simulation for performance analysis. Relational operation optimization for database design (sequential, ISAM, VSAM, SQL). Consistency/completeness, circular logic checking, optimization.
   ii. **Code generation:** Ada, DCL, JCL, C, PL/1. Forms/screen design via Painter.
   iii. **Testing:** User-specifiable test data generation (random provided).
   iv. **Maintenance:** Re-engineering for Ada, FORTRAN, C.
8. **Documentation generation:** User-definable formats. Data for 2167 available but no formats.
9. **Environment Characteristics:** Multi-user and network support via linked CASE.
10. **Database:** Repository.
11. **Links to other tools:** Cadre's Teamwork, Softool's CCC, IBM's VM/SE.
12. **Output formats:** ASCII.
13. **Planned enhancements:**
    i. Automated database population/change propagation.
    ii. Analyze to determine worst case time and show if satisfy timing requirements.

iii. Port to UNIX environments.
iv. Generation of FORTRAN.
v. Reverse engineering, currently working on FORTRAN and LISP.
vi. Generating programs for parallel processing.
vii. Accept object-oriented input.

14. **Collaboration with other organizations:** IBM for AD/Cycle.

Information From: (215) 854-0555

Tool Summary: Back-end CASE for design through maintenance. Accepts DFDs or non-procedural specifications as input. Performs I/O and memory optimization.

1. **Hardware Platforms:** IBM mainframe, VAX/VMS
2. **Tool Implementation Language:** Ada, C.
3. **Tool Price:** $25,000 to $150,000
4. **Vendor Support:** Training, consultancy. Support Group? Newsletter?
5. **Marketed Since:** 1981
6. **Size of customer base:** Mainly used in-house, less than 5 installations.
7. **Methodologies and functions at different development stages supported:**
   i. **Software design:**
      a. Methods/diagrams: SD and OOD, depends on front-end case. Forms/screen design. Consistency, completeness, circular logic checking.
   ii. **Code generation:** Ada, C, PL/1. Report/screen generation.
   iii. **Testing:** Automated test data generation either by user specified rules or random.
8. **Documentation generation:** User-definable formats. 2167A information available, no report formats.
9. **Project management support:** via front-end case.
10. **Environment Characteristics:** via front-end case.
11. **Database:** via front-end case, separate database not maintained.
12. **Links to other tools:** Interface to Teamwork, StP, potentially DEC's DecDesign.
13. **Output formats:** ASCII.
14. **User interface:** via front-end case.

Information From: Irene Nechaev (800) 537-4262

Address: 50 Tice Blvd., Woodcliff Lake, NJ 07675

Tool Summary: Picture Oriented Software Engineering (POSE) for systems planning and business area analysis, analysis, design, construction of information systems.

1. **Hardware Platforms:** IBM PC-XT, PC-AT, PS/2 or compatible, under DOS, OS/2. Macintosh.
2. **Products:** POSE alone $2,665; with FlexGen $3,995.
   i. Data model toolkit, any single module $595, toolkit for $1195:
      a. POSE-DMD Data Model Diagrammer
      b. POSE-DMN Data Model Normalizer
      c. POSE-LDD Logical Database Designer
      d. POSE-DBA Database Aid
   ii. Process model toolkit, any single module $595, toolkit for $1195:
      a. POSE-DCD Decomposition Diagrammer
      b. POSE-DFD Data Flow Diagrammer
      c. POSE-SC Structure Chart Diagrammer
      d. POSE-ACD Action Chart Diagrammer
   iii. POSE-SRP Screen Report Prototyper $595.
   iv. POSE-PMD Planning Matrix Diagrammer for business analysis/planning $595.
   v. Data Model Bridge (DMB) for uploading data models to KnowledgeWare's IEW $595.
   vi. LAN support $595.
3. **Tool Implementation Language:** COBOL
4. **Vendor Support:** Training, consultancy, twice yearly newsletter.
5. **Marketed Since:** 1979 in Europe, 1982 in USA. Preparing to release POSE Version 4.2 with reverse schema engineering, increased import/export functionality, complete data model integration and advanced utilities and input.
6. **Size of customer base:** User base of over 2,500 worldwide.
7. **Methodologies/functions supported:**
   i. **Software specification:** Yourdon, Gane-Sarson methods. Diagram balancing, consistency. Information engineering using Chen, Merise. Libraries for reuse of objects. Automated database population/change propagation.
   ii. **Software design:** Constantine method. Database design.
   iii. **Code generation:** COBOL through FlexGen. Schema generation for various database including DB2, SQL. Forms/screen design with prototyping.
   iv. **Maintenance:** Reverse schema engineering to allow importing existing database schemas to populate the DMD data dictionary for new applications.
8. **Documentation generation:** User-definable report generation.
9. **Project management support:** Security/control access, project planning, status reporting, change reporting. Configuration management.
10. **Environment Characteristics:** Network support but not multi-user.
11. **Database:** Data dictionary implemented as a database with published interfaces. Database split/merge. Import/export function for exchange of information with other CASE tools. Also

ASCII file generation.

12. **Links to other tools:**
    i. Generates code via link to FlexGen (from SINC, Inc.) which provides 4GL programming language, rapid prototyping, source code generation, user query, and report tools.
    ii. DMB for uploading data models to KnowledgeWare's IEW.
    iii. Export via ASCII to code generators, some existing interfaces.
    iv. IBM's CSP application generator.
13. **Output formats:** HPGL, ASCII.
14. **User interface:** Menu and mouse, color, windowing. Database browser/query facility, on-line help.
15. **Adaptability:** Free-form text/graphics.
16. **Planned enhancements:**
    i. MS Windows and IBM OS/2.
    ii. Multi-user version end '91 or early '92.
17. **Collaboration with other organizations:** Conformance with IBM's Ad/Cycle.

Information From: Irene Nechaev (800) 537-4262

Address: 50 Tice Blvd., Woodcliff Lake, NJ 07675

Tool Summary: SILVERRUN series support rule-based building and refining of data models, generation of SQL, and building/validating DFDs.

1. **Hardware Platforms:** Mac PC
2. **Components:** It consists of a Relational Data Moduler (RDM) module, a Data Flow Diagrammer (DFD) module, and an Entity Relationship Expert (ERX) module. Preparing Release 2.0.5. operates under X-Windows, OS/2. Each of the 3 modules costs $2,500.
3. **Tool Implementation Language:** C++
4. **Vendor Support:** Training, consultancy, hot-line, newsletter. Users group being established.
5. **Marketed since:** 1988
6. **Size of customer base:** 3000 licenses
7. **Methodologies and functions at different development stages supported:**
   i. **Software specification:** Supports Gane-Sarson, Yourdon-DeMarco with ERDs for information modeling.
   ii. **Software design:** Database design with schema generation for Ingres, DB2. Screen/forms prototyper.
8. **Documentation generation:** User-definable formats.
9. **Database:** Data dictionary implemented as database.
10. **Output formats:** ASCII.
11. **User interface:** Menu and mouse, windowing, color.
12. **Adaptability:** Free-form text/graphics.
13. **Planned enhancements:**
    i. Integration with POSE.
    ii. Generation of C code, late 1991.
    iii. Multi-user, network support, later 1991.

Information From: Eric Rivas (713) 480-3233, May 21 1991.

Address: 17629 El Camino Real, Suite 202, Houston, TX 77058

Tool Summary: Backend CASE tool to support requirements definition, objects analysis, and code generation, does not support graphical analysis of application problem space.

1. **Hardware Platforms:** IBM AT
2. **Products:** Basic system $1,990, with Ada code generator $2,490. Volume discounts available.
3. **Tool Implementation Language:** C
4. **Vendor Support:** Training, consultancy, hot-line, bulletin board.
5. **Marketed Since:** 1989
6. **Size of customer base:** 700 licenses
7. **Methodologies and functions at different development stages supported:**
   i. **Software specification:** CORE method with application-tailored requirements templates. Object-oriented Analysis using the Coad/Yourdon method. Information matrix analysis. Traceability. Auto database population/change propagation.
   ii. **Software design:** Object-oriented Design. Schema generation for DB2, Oracle, SQL/D, dBASE, Paradox, and others.
   iii. **Code generation:** Ada, C++, C, Turbo Pascal.
   iv. **Maintenance:** Re-engineering for C and C++.
8. **Documentation generation:** Customizable and 2167A templates.
9. **Project management support:** Version control.
10. **Environment Characteristics:** Multi-user and network support.
11. **Database:** Object-oriented repository implemented as a database. Import/export in flat files and Common Delimited ASCII. Database split/merge.
12. **Output formats:** ASCII.
13. **User interface:** Menu and mouse, windowing, on-line help, database browser/query facility.
14. **Adaptability:** Some methodology tailoring.
15. **Planned enhancements:**
    i. X-Windows/Motif version.
    ii. Inheritance.
    iii. General-purpose graphical editor.

Information From: Leon Stucki (206) 939-7552, 23 may 1991.

Tool Summary: Formerly DesignVision by Ken Orr Institute.

1. **Hardware Platforms:** IBM PS/2 under OS/2.
2. **Tool Implementation Language:** C
3. **Tool Price:** Single user $7,500. Volume discounts available.
4. **Vendor Support:** Training, consultancy, support group, newsletter.
5. **Marketed Since:** 1986
6. **Size of customer base:** Around 600 installations.
7. **Methodologies and functions at different development stages supported:**
   i. **Software specification:** SQL interface provides some support for requirements extraction. Structured Analysis, with limited support for real-time extensions. Chen information modeling. Automated database population/change propagation.
   ii. **Software design:** SC.
   iii. **Code generation:** User-definable templates for some C generation. Schema generation via link to Olivetti products, tool provides some itself.
8. **Documentation generation:** User-definable formats.
9. **Project management support:** Security/control access.
10. **Environment Characteristics:** Multi-user and network support.
11. **Database:** Object-oriented repository implemented as database. Import/export facility.
12. **Links to other tools:**
    i. Link to Olivetti products for forms/screen design and schema generation.
    ii. Link from Brackets to Envision (Envision to Brackets planned).
13. **Output formats:** ASCII.
14. **User interface:** Menu and mouse, windowing, color, on-line help. Database browser/query facility.
15. **Adaptability:** Free-form text/graphics, some methodology tailoring.
16. **Planned enhancements:**
    i. Link to MicroSoft's Project for project management support.
    ii. Reverse engineering.
    iii. Link to Olivetti products for prototyping.
    iv. Simulation.
    v. Integrate Brackets with Envision.
17. **Collaboration with other Organizations:** IBM AD/Cycle.

Information From: May 6, 1991.

Address: 22 Third Avenue, Burlington, MA 01803

Tool Summary: Workstation-based graphical support for simulation and prototyping. Executable specification for real-time software, screen display forms. Test data used to emulate system environment and uncompleted portions of system. Color animation of diagrams.

1. **Hardware Platforms:** Sun with UNIX and SunOS, VaxStation with MicroVMS and UIS software, Apollo/Aegis with DomainIX. VAX/VMS, RISC-based Sun and DEC workstations, IBM PC/AIX.
2. **Products:** Each with kernel (3 graphics editors) and training for 2 people. Maintenance 15%.
   i. Statemate Analyzer $25,000.
   ii. Statemate Prototyper to generate code $30,000 (for either Ada or C).
   iii. Statemate Documentor for customized output includes Statemate Dataport to access outside elements and database, $20,000.
   iv. EXPRESS VHDL.
3. **Tool Implementation Language:** C
4. **Vendor Support:** Training, consultancy, technical support line.
5. **Marketed Since:** 1987
6. **Size of customer base:** Approx 700 copies.
7. **Methodologies and functions at different development stages supported:**
   i. **System specification:** System definition and specification, system requirements analysis and design (with EXPRESS VHDL for hardware specification), system integration and testing, validation testing. Simulation with state reachability, deadlocks, race conditions.
   ii. **Software specification:** David Herel's method with activity charts, data dictionary entries, state charts (concurrency and hierarchy, extension of state transition diagrams), module charts (physical system architecture). Some timing information, concurrency. Consistency/completeness checks of model. Automatic change propagation. Dynamic and behavioral validation, interactive/batch simulation, dynamic reachability and non-determinism testing, no dynamic timing or hardware allocation. Traceability.
   iii. **Software design:** Module charts (not SC). Traceability between design elements and forms (formal and informal textual information such as requirements list). Forms editor,
   iv. **Code generation:** Ada, C
8. **Documentation generation:** Text and graphics, user-definable and built-in templates (including 2167A templates).
9. **Project management support:** Configuration management, logging and versioning of files, security/control access, status reporting, change reporting.
10. **Environment Characteristics:** Multi-user, no replication.
11. **Database:** Repository of ASCII files used like native DBMS (InterBase). DATAPORT facility via C routines for import/export of ASCII data, provides bridge to other tools. Database split/merge.
12. **Links to other tools:**
    i. DesignAid: Network support using IBM PC-Network and Novell Advanced NetWare.

ii. Uses RDB from MicroVAX, Interbase from Sun and Apollo.

13. **Output formats:** ASCII, PostScript, Interleaf, troff, nroff, HPGL.

14. **User interface:** Menu and mouse, windowing, color, on-line help, 1 level of undo. Menu-driven query facility for database.

15. **Adaptability:**
Graphic editors are rule-based with automatic syntax checking.

16. **Standards conformance:** EXPRESS VHDL (1076 compliant VHDL).

17. **Planned enhancements:** Design to test link for performance analysis, end of '91.

Information From: Lesley Mangeri (703) 848-8808

Tool Summary: Open architecture called Visible Connections with published interfaces.

1. **Hardware Platforms:** DEC VAXstation, Sun, HP/Apollo workstations, IBM RISC, and others under UNIX, X-Windows.
2. **Products:** $5,000 to $12,000
   i. OOSD/Ada Release 1.0. Release 1.1 will include code generation from designs, 2167A support, X-Windows support (summer '91), and reverse engineering 92.
   ii. OOSD/C++ with graphical design editor, expected end '91.
   iii. CDE Phase I released 1990. Reverse engineering and code generation in Phase II. Integrated between design and construction tools.
   iv. StP Integrated Structured Environment with Document Preparation System with 2167 and user-definable report templates. Document browsing capability, interface with external work processing systems. Mixing text/graphics. Comes with each of above modules.
   v. Rapid prototyping tool.
3. **Tool Implementation Language:** C, C++, Ada.
4. **Vendor Support:** Training, quarterly newsletter, consultancy, support group, hot-line.
5. **Marketed Since:** 1985. Currently release 4.3.
6. **Size of customer base:** 4000 installations.
7. **Methodologies and functions at different development stages supported:**
   i. **Software specification:** Gane-Sarson, Yourdon-DeMarco, Hatley methods. No explicit timing or other quantitative performance information, replication, resource allocation. Chen and Jackson data structure diagrams for information modeling. Diagram and decomposition checking, consistency with database and between diagram types. Automated database population and change propagation on demand. Traceability.
   ii. **Software design:** Structure charts, mini-specs. Supports Wasserman's User Software Engineering for interface design and prototyping. Parameter checking for static analysis. Database design with SQL schema generation for various relational databases including DB2, Informix, Ingres, Interbase, Oracle.
   iii. **Code generation:** User-definable source code templates for Ada, C, Pascal, PDL for data and type declarations from design descriptions. Structured Chart Editor templates for COBOL. RAPID/USE code for user interface development.
   iv. **Testing:** For SA/SD portion via bridge to McCabe's tools.
8. **Documentation generation:** User-defined and 2167A templates.
9. **Project management support:** Security/control access.
10. **Environment Characteristics:** Multi-user and network (heterogeneous) support. Multiple project support.
11. **Database:** Object management library (repository) implemented as relational database, user-definable schema with data independent interface to data dictionary. Database split/merge, import/export with defined data formats.
12. **Links to other tools:**
    i. Atherton's Software Backplane.
    ii. 4GLS

iii. Interleaf and FrameMaker publishing.

13. **Output formats:** PostScript, troff, UNIX pic, raster.

14. **User interface:** Menu and mouse, windowing, on-line help, undo. Database browser.

15. **Adaptability:** Object Annotation Editor to associate properties and values with diagram objects based on user-defined annotation templates. Annotation information extracted from data dictionary via Object Management Language, Documentation Preparation System, or Troll DBMS facilities. Special tool for limited methodology tailoring.

16. **Standards conformance:** CDIF.

17. **Planned enhancements:**

    i. RISC/AIX platforms 3rd quarter 90, single license $5,000 to $21,000.
    ii. Expect generation of C++ (through Saber-C) next year.
    iii. Reverse engineering.

18. **Collaboration with other organizations:**

    i. Group Bull for their internal use.
    ii. Saber Software (for C coding, testing and re-engineering).
    iii. Informix Software, joint marketing agreement. SQP support.

Information From: Neil McCoy (703) 391-2771, May 7 1991.

Address: 2800 28th Street, Suite 320, Santa Monica, CA 90405

Tool Summary:

1. **Hardware Platforms:** All on Macintosh PCs, FreeFlow under DOS windows and Sun/UNIX. AdaFlow Sun/UNIX by fall 1991, other environments by end of year.
2. **Products:** PowerTools/RT, PowerTools/MIS, PowerTools/Ada each $4,995. PowerTools/Engineering $5,995. PowerTools/AdaVantage, PowerTools/LifeCycle $6,995. Training approx $500 a day, on methodology via 3rd party. Components:
   i. AdaFLOW hierarchical Buhr/Booch diagram editing with dictionary and language sensitive editing support, $1,995.
   ii. Free Flow support for DeMarco/Hatley.
   iii. Fast Task real-time SA extensions.
   iv. DataModeler for modeling and logical database design.
   v. QuickChart shows partition of software into modules (Constantine).
   vi. SmartChart structure chart generator.
   vii. PowerPDL translates pseudo-code into trees needed for SmartChart and generates formatted documentation.
   viii. ASCII Bridge merges multiple dictionaries and import/export facility.
   ix. CoCoPro.
3. **Tool Implementation Language:** Pascal and C.
4. **Vendor Support:** Training, consultancy.
5. **Marketed Since:** 1986
6. **Size of customer base:** 1500 copies
7. **Methodologies/functions supported:**
   i. **Software specification:** Can import requirements specification from Teamwork. DeMarco and Hatley/Ward-Mellor. Schlaer-Mellor OOA methods, Chen, Martin methods and IDEF1X, ERA editor for information modeling. Consistency, diagram balancing, database/diagram consistency checking. Traceability in AdaFLOW via comments in data dictionary. Automated database population/change propagation.
   ii. **Software design:** Constantine SD with Page-Jones extensions, Structured Object-Oriented Design (SOOD) in AdaFlow. PDL with document generation. DataModeler builds textual source files containing SQL, COBOL, or other source language data definitions for database design.
   iii. **Code generation:** QuickChart for C, C++, etc. (Pascal, Modula-2, LISP, Prolog, FORTRAN, PDL, Jovial). AdaFlow for Ada.
   iv. **Maintenance:** Re-engineering via SmartCheck, PDL for software developed using tools.
8. **Documentation generation:** User-defined and 2167A templates.
9. **Database:** Data dictionary implemented as file system, together with diagrams maintained as integrated encyclopedia. Multiple typing in data dictionary.
10. **Project management support:** CoCoMo cost modeling. Security/control access, configuration management via ASCII Bridge, export after date stamping.

11. **Environment Characteristics:** Multi-user, network support.
12. **Database:** Import/export to DBMS via ASCII Bridge. Split/Merge.
13. **Links to other tools:**
    i. See ASCII Bridge.
    ii. Teamwork for requirements.
14. **Output formats:** ASCII, Interleaf. In Mac environment support WordPerfect and such.
15. **User interface:** Menu and mouse, windowing, color, undo facility, database browser.
16. **Standards conformance:** CDIF
17. **Planned enhancements:**
    i. Publish and subscribe to replace cut and paste and allow automatic updating.
    ii. All tools under DOS Windows and Sun/UNIX. Release on multiple platforms e.g., combination of UNIX and DOS environments.
    iii. Requirements traceability tool, fall '91.
    iv. Potentially link to Advanced Systems Technology, Inc.'s QASE RT for simulation.
18. **Collaboration with other organizations:**
    i. Joint marketing venture with Meridian for purchase with Meridian Ada Vantage compiler.
    ii. IBM Ad/Cycle.

Information From: Julie Kelly (800) 777-8858, hot-line (800) 888-4203. May 7 1991.

Address: One Main Street, Cambridge, MA 02142

Tool Summary: Planning, analysis, design, construction and re-engineering of information systems, supporting overview of a database and interacting application.

1. **Hardware Platforms:** IBM PC/DOS, VAXstation/VMS.
2. **Products:** Maintenance $882 per copy.
   i. Excelerator/IS, includes XLDictionary for integration project information $9,800.
   ii. Excelerator/RTS, includes XLDictionary for integration project information $9,800.
   iii. XL/DOC add-on for documentation generation to user-specified formats/scripts $4000.
   iv. PC Prism supports both IS and RTS, computer aided system planning $8000.
   v. Excelerator for Design Recovery for re-engineering of COBOL. Taking off market.
   vi. Customizer package to tailor Excelerator, modify graphs, screen descriptions $12,500.
   vii. XL/Quickstart provides on-line assistance for using Excelerator.
   viii. IDEF/LEVERAGE, a custom version of Excelerator to automate IDEF modeling.
3. **Tool Implementation Language:** C++
4. **Vendor Support:** Publishes CASE magazine. Training, consultancy, hot-line, support group and newsletter.
5. **Marketed Since:** About 1984. Currently release 1.9.
6. **Size of customer base:** 100,000 installations.
7. **Methodologies/functions supported:**
   i. **Software specification:** Yourdon, Gane-Sarson, Ward-Mellor, Hatley, SSADM methods. Chen and Merise ERDs for information modeling. Diagram balancing, syntax/semantics, database/diagram consistency checking. Automated database population/change propagation. Traceability of engineering and user requirements.
   ii. **Software design:** Constantine charts, Jackson structure diagrams. Verifies normalization to support database design.
   iii. **Code generation:** Transform database record descriptions into BASIC, C, COBOL, PL/1. Forms/screen design with prototyping in Basic, C, COBOL, PL/1.
8. **Documentation generation:** Customizable and user-definable formats, 2167A support.
9. **Project management support:** Access control, assignment to project tasks, workbreakdown structure diagrams, presentation graphs.
10. **Environment Characteristics:** Central project dictionary. Multi-user, network support. Database split/merge facility, multiple project support. Access to database by XL/Programmer Interface. Export to dBASE II, and other databases.
11. **Links to other tools:**
    i. Bridge to IBM CSP and JAD, DB2. Rep (PC Prism).
    ii. 4FRONT integration framework from Deloitte & Touche.
    iii. Bridge by XL/Interface to TELON for prototyping or MicroFocus COBOL/2 Workbench.
    iv. Bridge to Sage's APS Development Center.
    v. XL-XPRESS bridge to PSL/PSA.
    vi. Interface to Aldus PageMaker, GDDM, Ventura Publisher.
    vii. Softool's CCC.

viii. Applied Business Technology's Project Workbench.
ix. Interface to other application generators for COBOL.
x. Interface to 4GL MANTIS, PowerHouse.

12. **Output formats:** PostScript, HPGL. Interleaf for VAX version.
13. **User interface:** Menu and mouse, windowing, color, some on-line help. Database query/browser.
14. **Adaptability:** Free-form text/graphics via Customerizer package.
15. **Standards conformance:** SAA next version.
16. **Planned enhancements:**
    i. Improved static analysis, executable specs with Petri-nets.
    ii. Support for OS/2.
17. **Collaboration with other organizations:**
    i. IBM partner, AD/Cycle.
    ii. Merged with Sage, supporting APS application generator. (Sage now called Intersolve.)

Information From: Bruce Donadt (508) 393-1231, May 8 1991.

Address: 2500 Mission College Blvd., Santa Clara CA 95054-1215

Tool Summary: Graphical environment for mathematically-based design of real-time control systems with design capture, simulation and code generation in Ada, C, Fortran. Automates development of real-time software from SYSTEM_BUILD's high-level graphical design. 2 and 3D plotting.

1. **Hardware Platforms:** VAXstation, HP/Apollo, SUN workstations, IBM PC.
2. **Components:** Single-user workstation from $20,000 to $43,000. Multiple licenses multiple by factor of 1.4, and factor of 2.4 for multi-user licenses. This purchases full support and use of software for 1 year, must renew at 20% each subsequent year.
   i. SYSTEM_BUILD for graphical modeling and simulation of nonlinear, continuous, event driven and sampled-data systems. Includes Case Extension Module, RT/Expert Module, RT/Fuzzy Module. Simulation enhancements include Interactive Animation Module, HyperBuild Module, RemoteSim Module.
   ii. MATRIXx Analysis and Design for interactive control system analysis and design.
   iii. Xmath scientific and engineering mathematics, graphics, and programming.
   iv. AutoCode Real-Time Code Generation generates code directly from high-level SystemBuild block diagrams in Ada, C, FORTRAN.
   v. AC-100 Implementation and Testing supports testing of control software and hardware.
3. **Tool Implementation Language:** C++ (and others for math routines).
4. **Vendor Support:** Newsletter, training, consultancy, support group, hot-line.
5. **Marketed Since:** SYSTEM_BUILD since 1983, AutoCode (SystemBuild + code generation module) since '86. Currently release 2.04.
6. **Size of customer base:** 600-700 installations.
7. **Methodologies/functions supported:**
   i. **System specification:** Graphical model, ST, global data stores, finite state machines. Information modeling. Hardware/software allocation. Simulation with timing information, can include code in any compilable language. Automated database population/change propagation. Traceability.
   ii. **Code generation:** Ada, C, FORTRAN.
8. **Environment Characteristics:** Multi-user and network support.
9. **Database:** No import/export. Data dictionary implemented as database. Database split/merge. User Code Block interface allows Ada, FORTRAN, C modules to be added to the library.
10. **Output formats:** PostScript, Interleaf.
11. **User interface:** Menu and mouse, windowing, color, on-line help.
12. **Standards conformance:** Next release X-Windows under Motif.
13. **Planned enhancements:**
    i. Document generator (summer '91) will provide user-definable templates and 2167 documentation aids.
    ii. Open architecture allowing import/export from/to other CASE tools.

Information From: Brenda Watkins (703) 506-0823 x7040, Jeff Wiley for technical support.

Address: 3340 Peachtree Road, N.E., Atlanta, GA 30326

Tool Summary: Set of integrated rule-based CASE tools running on micros designed to develop applications for mainframe IBM environments. Tools integrated round central object-oriented encyclopedia, likely to be kernel of IBM's repository product. Re-use support.

1. **Hardware Platforms:** IBM PS/2, OS/2 with Presentation Manager.
2. **Products:**
   i. Application Development Workbench (ADW) comprises the Design Workstation, Construction Workstation, Planning Workstation, and Analysis Workstation. The Starter Kit is $15,000. ADW/MVS operates in a mainframe environment (MVS/TSO), an open architecture framework that can be used with PWS CASE tools, IEW, and ADW.
   ii. ADW/RAD for application animation and automated generation of design information from specification. Uses object-oriented methods and a non-procedural specification language. Purchased separately costs $1,500, or with ADW/DOC for $2000. Executes on IBM PS/2. It focuses on a tactical or business area analysis project and the associated analysis and design to drive application development of the business model. It can be driven by the process and data models defined by the ADW/Analysis Workstation. Application Animator for iteratively prototyping the specification. Application Design Generator to generate the application design (screen layouts, action diagrams, structure charts and data structures) into the ADW/Design Workstation (2nd release). Initial version targeting text-base applications, subsequently GUI applications.
   iii. ADW/DOC for documentation support. Purchased separately costs $1,500, or with ADW/RAD for $2000.
   iv. GAMMA COBOL generator $209,300 for first license.
   v. Repository Enablement Facility provides a bridge between KnowledgeWare's encyclopedia and RM/MVS.
   vi. IEW Starter Kit is $15,000.
3. **Tool Implementation Language:** C
4. **Vendor Support:** Training, consultancy, newsletter, hot-line, support group.
5. **Marketed Since:** IEW since 1985, ADW since 1990.
6. **Size of customer base:** 55k copies, >3k sites.
7. **Methodologies/functions supported:**
   i. **Software specification:** Yourdon-DeMarco, Gane-Sarson, Ernst-Young methods. James Martin's Object Oriented Analysis, and ERDs for information modeling. Simulation via ADW/RAD. Syntax/semantics, diagram balancing, database/diagram consistency, consistency with planning stage checking; the Knowledge Coordinator around the encyclopedia ensures referential integrity, consistency, etc. Traceability. Automated database population and change propagation.
   ii. **Software design:** SC and module action diagrams generated from specification. Screen/forms design and prototyping. Generate SQL Data Definition Language, COBOL for database.
   iii. **Code generation:** Templates for C, Ada, COBOL, FORTRAN, Pascal, PL/1, and others.
   iv. **Maintenance:** Re-engineering from COBOL.

8. **Documentation generation:** User-definable and 2167A templates via ADW/DOC.
9. **Project management support:** Audit trail, security/control access, some project planning.
10. **Environment Characteristics:** Multi-user, network support via LAN.
11. **Database:** Repository with split/merge, import/export facility.
12. **Links to other CASE tools:**
    i. Mark V's Adagen/KW001 interface extensions for Ada generation for IEW/AWS.
    ii. Software One Ltd. interface from Auto-Mate Plus to IEW/ADW, from Teamwork to IEW/ADW, and between IEF and IEW.
    iii. Barton Group interface IEF to IEW or ADW, and with INGRES/Pansophic.
    iv. Fina Oil interface from Excelerator to IEW and between Design/1 CASE Tool and IEW.
    v. Computer Associates interface with Architect.
    vi. Cortex Ltd. interface from IEW/DWS to CorVision.
    vii. EDS interface from IEW/AWS (soon IEW/ADW) to Pacbase.
    viii. Comp. Eng. Cons. bi-directional interface for IEW/ADW and CEC's Analyst Workbench.
    ix. Software AG interface from IEW/ADW to Predict (also Excelerator to Predict).
    x. U.S. Sprint interface from Prokit Workbench to IEW.
13. **Links for reverse engineering:**
    i. InterCASE for transfer of data to IEW/AWS and IEW/DWS.
    ii. Utilities for database reverse engineering.
14. **Links to code generators:**
    a. TELON code generator for COBOL and PL/1.
    b. Barton Group working on bi-directional interface between IEW/AWS and Bachman's Data Analyst. Also Bachman interface from IEW/AWS to Data Analyst.
    c. Ernst & Young interface from IEW/DWS into Microfocus Workbench for generation of object code from IEW's COBOL.
    d. Bi-directional interface between IEW and Uniface (4th gen application development system).
    e. Bonner & Moore Consulting interface to Netron's Cap.
    f. Interface to Clarion code generator.
    g. APS/IEW PC Interface for bridge from IEW/AWS to Sage's APS. Bi-directional IEW/DWS interfaces by John Deere.
    h. SAA interface from IEW/DWS to AS/SET code generator for RPG/400.
    i. KnowledgeWare's bi-directional interface to IBM's CSP and own COBOL generator.
    j. Pro-C code generator for C.
15. **Output formats:** ASCII, PostScript.
16. **User interface:** Menu and mouse, windowing, color, on-line help with hypertext. Database browser/query facility.
17. **Adaptability:** Free-form text/graphics, some methodology tailoring.
18. **Standards conformance:** IBM SAA, National Language Support (NIS).
19. **Planned enhancements:**
    i. Real-time extensions to be released in January 1992.
    ii. C generation in 1992.
20. **Collaboration with other organizations:** IBM AD/Cycle.

Information From: Giovanna Petrone 39 11 831.1830, FAX 39 11 812.1235

Email: giovanna@lps@i2unix.uucp

Address: Via Napione 25, 10124 Torino, Italy

USA distributors for Ada products: (703) 648-1551

Tool Summary: For detailed, programming, and documentation of software projects using Ada, C, C++, FORTRAN, COBOL, Pascal, and others. Uses hypertext technology. Formerly DUAL and KEYLINE.

1. **Hardware Platforms:** DEC VAX/VMS, Sun and Apollo workstations, IBM PS/2 and RISC systems, PC, HP series 9000.
2. **Products:** The full KeyOne package (for Ada) starts at $895 for IBM PC. C++ package starts at $2,850 on workstations. Ranges up to $21,400 for Ada or C++ on VAX 8974, 8840, 8978, 6360, 6333, 8842. Maintenance is 15% of license price, with updates during maintenance period costing $300.
   i. KeyFlex hybrid editor ranges from $295 (Ada) and $1,800 (C++) to $15,000.
   ii. KeyDesign syntax directed editor for design.
   iii. KeyDoc structured documentation generator.
   iv. Off-the-shelf translators for Pascal to Ada, Ada PDL to C, HOOD PDL to Ada or C.
   v. Intermodule navigation for KeyOne for Ada 15% of Ada license price.
   vi. DoD 2167A documentation support 15% of license price.
   vii. SQL extension to standard languages (C, COBOL, Ada) 10% license price.
3. **Tool Implementation Language:** C
4. **Vendor Support:** Consultancy, training, hot-line.
5. **Marketed Since:** DUAL introduced in 1982, KeyOne in 1987.
6. **Size of customer base:** >600 installations
7. **Methodologies and functions at different development stages supported:**
   i. **Software design:** Step-wise refinement with James Martin action diagrams. Automated database population/change propagation?
   ii. **Code generation:** Ada, C, C++, Pascal, FORTRAN, COBOL.
   iii. **Maintenance:** Re-engineering for Ada, C, C++, FORTRAN, Pascal.
8. **Documentation generation:** User-definable formats, 2167A templates.
9. **Project management support:** Security/control access.
10. **Environment Characteristics:** Multi-user, network support.
11. **Database:** Data dictionary implemented as file system. Import/export?
12. **Output formats:** PostScript.
13. **User interface:** indowing, on-line context-sensitive help, undo facility.
14. **Planned enhancements:** Translators are being developed for Jovial to Ada, FORTRAN to Ada or C, Ada to HOOD PDL reverse translator.

Information From: Grace Farenbaugh (818) 995-7671, May 7 1991.

Address: 16400 Ventura Blvd., Suite 303, Encino CA 91436

Tool Summary: Code generation and reverse engineering for Ada, C, C++. Extensibility a major feature. Designed to facilitate rule-based integration with other methods/tools.

1. **Platforms:** IBM PC/DOS, MACs, and under UNIX/Windows for any workstation.
2. **Products:** As a whole, ObjectMaker CASE Tool (analysis/design, menu customization, and 1 language) $8,000. Volume discounts available. Maintenance 15% source price.
   i. ObjectMaker Analysis and Design, drawer, database repository, and methods support $5,000.
   ii. ObjectMaker Tool Development Kit (TDK) provides access to rules for extensive customization $25,000.
   iii. Menu customization kit for menus and acceleration keys $1,500.
   iv. Adagen language module for Ada code generation and reverse-engineering $3k.
   v. Cgen language module for C, C++ code generation and reverse-engineering $3k.
3. **Tool Implementation Language:** C, Prolog, Ada.
4. **Vendor Support:** Training, consultancy. Starting a support group and newsletter.
5. **Marketed Since:** AdaGen since 1986, ObjectMaker Version 1.8 since April '91.
6. **Size of customer base:** 500 seats, 80 organizations.
7. **Methodologies/functions supported:**
   i. **Software specification:** Yourdon, Ward-Mellor, Hatley, Coad-Yourdon methods. Block, F-net, R-net, and Petri-net diagrams. Chen, Schlaer-Mellor for information modeling. Diagram balancing, syntax/semantics, database/diagram consistency checking. Automated database population/change propagation.
   ii. **Software design:** Many, including Constantine, Booch/Buhr methods. Some support for database design, not fully automated.
   iii. **Code generation:** Ada, C, C++
   iv. **Maintenance:** Re-engineering for Ada, C++ available July '91.
8. **Documentation generation:** Fixed. 2167A via DOCGEN2167 runing on PCs and Mac, own support available by end of '91.
9. **Environment Characteristics:** Multi-user and network support for UNIX version.
10. **Database:** Reposity, import/export. Published interfaces and split/merge by end '91.
11. **Output formats:** ASCII, PostScript, Interleaf, HPGL, Troff, nroff, FrameMaker, WordPerfect.
12. **User interface:** Menu and mouse, windowing, color, on-line help, undo. Database browser via forms/tables component later this year.
13. **Adaptability:** Tool kit allows additions or modifications of methods, graphical notations, database schema, and user interface, including custom languages and framework support.
14. **Standards conformance:** CDIF, PCTE.
15. **Planned enhancements:**
   i. Schema generation.
   ii. More hardware platforms.
   iii. User definable report formats and full support for 2167A.

Information From: John di Fernandos (503) 685-4830, May 7 1991.

Address: 17052 Jamboree Blvd., Irvine, CA 92714

Tool Summary: Graphics modeling environment with engineering analysis, planning, simulation, and real-time code generation, optimization, and automated documentation. With MATRIXxCAE for CAE/CASE integration. Formerly TekCASE.

1. **Hardware Platforms:** Apollo workstations, OSF/Motif.
2. **Products:** $25K to $40K for a single workstation.
   i. CASE Station.
   ii. CodeLink Station.
   iii. DOC technical publishing.
3. **Tool Implementation Language:** C++
4. **Vendor Support:** Training, consultancy, support group, newsletter.
5. **Marketed Since:** 1984, Version 2.0
6. **Size of customer base:** >3k users
7. **Methodologies/functions supported:**
   i. **Software specification:** Youron-DeMarco, Ward-Mellor, Hatley methods, with ERDs for information modeling. 70 rule-based checking facilities. Automated database pop/change.
   ii. **Software design:** SC with prototyping and forms/screen design.
   iii. **Code generation:** Code frames for C.
   iv. **Testing:** Debugging, coverage and performance analysis.
   v. **Maintenance:** Re-engineering from C.
8. **Documentation generation:** Report generation, 2167A support.
9. **Project management support:** Version management via Design Manager.
10. **Environment Characteristics:** Multi-user and network support.
11. **Database:** Use host's file system, store data in an intermediary ASCII format.
12. **Output formats:** PostScript, other.
13. **User interface:** Menu and mouse, windowing, color, on-line help.
14. **Adaptability:** Methodology tailoring (only things such as changing error messages).

Information From: (800) 333-6382

Tool Summary: Open architecture. Evolved from PSL/PSA which now provides repository facilities. Formerly marketed by Meta Systems, now bought out by LBMS.

1. **Hardware Platforms:** IBM PC
2. **Products:** SA Workbench $6,995. Metabase Import/Export Utilities for interface between QuickSpec, SA Workbench and PSL/PSA.
3. **Tool Implementation Language:** C
4. **Vendor Support:** Hot-line, training, consultancy, newletter.
5. **Marketed Since:** PSL/PSA since 1975, Workbench since April 1990.
6. **Size of customer base:** 300 licenses
7. **Methodologies and functions at different development stages supported:**
   i. **Software specification:** Can accept input from QuickSpec of system specification in Microsoft Windows. SA, Ward-Mellor methods with traceability. DFDs can be created from PSL information. Information modeling. Static analysis of diagram balancing and consistency. Some resource allocation. Automatic database population, change propagation.
8. **Documentation generation:** User-definable formats, 2167A templates.
9. **Database:** Repository, bridge to PSL/PSA. Proprietary object-oriented database. Split/merge, import/export facility, published interfaces.
10. **Links to other tools:** Wordprocessing and desktop publishing systems.
11. **Output formats:** ASCII.
12. **User interface:** Menu and mouse, windowing, color, on-line help, undo. Database query facility only through reports.

Information From: Maria Campbell (313) 663-6027

Tool Summary: Systems Engineer is a rewrite of Auto-Mate Plus. Open-architecture for desk-top based development with adherence to Dynamic Data Exchange and Object Linking and Embedding interface standards to tool extension.

1. **Hardware Platforms:** IBM PS/2, network under NETBIOS compatible LAN.
2. **Components:** System Engineer $7,500.
   i. SE/Open component for integration of Systems Engineer with other tools.
   ii. Applications Engineer generates applications using input from System Engineer. Based on Jackson Technology.
   iii. Information Manager supports integration and control of multiple System Engineer workgroup SQL databases across an origanization. Also key component of LBMS REVENG.
   iv. REVENG reverse and re-engineering toolset applies to C, COBOL, FORTRAN. Dynamic analysis capabilities based on instrumentation are being added.
   v. Strategic Planner supports business and strategic data modeling and planning to produce a phased strategic IT plan.
   vi. Project Engineer for project planning and estimating, extensions will include progress monitoring and an expert system to act as an advisor and validator of project plans.
   vii. On-Line Methods based on hypertext and hypergarphics to provide support for development.
3. **Tool Implementation Language:** C
4. **Vendor Support:** Training, consultancy, hot-line, newsletter.
5. **Marketed Since:** Auto-Mate Plus first released in 1985. System Engineer since Febuary 1990, current version 2.1S.
6. **Size of customer base:** 12,000 users in Europe and USA.
7. **Methodologies/functions supported:**
   i. **System Specification:** Problem requirements and solutions analysis. Traceability. System structure diagrams.
   ii. **Software Specification:** DFDs, entity life history, data modeling diagrams. Automated database population/change propagation.
   iii. **Software Design:** Functional decomposition. Automated generation of pseudo code, knowldge-based normalization and automated logical to physical design. Screens/form design with prototyping.
   iv. **Code Generation:** COBOL, PL/1, Ada, C.
8. **Documentation generation:** No user-definable formats, 2167A information available but not formatted.
9. **Project management support:** Security/control access, version control, project planning.
10. **Environment Characteristics:** Multi-user, network support.
11. **Database:** Repository implemented as database.
12. **Links to other tools:** SSADM Version 4.
13. **Output formats:** PostScript, ASCII, Interleaf, HPGL.
14. **User interface:** Menu/mouse, windowing, color, on-line validation, on-line tutorial, help. Browser/query facility.

15. **Adaptability:** Free-form text/graphics and some methodology adaptability.

16. **Standards conformance:** CDIF, IRDS, AD/Cycle, Common User Access (CUA) graphical user interface.

17. **Planned enhancements:**

    i. OS/2 Presentation Manager support and Information Manager Integration, 2nd quarter 1991.
    ii. Improved windows based data design module, enhancements to design tools, e.g., data modeling, and full Applications Engineer Integration, 3rd quarter 1991.
    iii. GUI painter to generate C for Windows and Presentation Manager.
    iv. Object orientation approach.
    v. Generation of 100% GUI application code, through enhancement of System Engineer to support C and C++.
    vi. Matrix handling for enhanced data modeling, JSP support.

Information From: Mike Skiles (800) 872-8296

Tool Summary: Project manager workbench, requirements management and analysis system, structured analysis and design. Nastec was previously Transform Logic Corp.

1. **Hardware Platforms:** DEC VaxStation, IBM PC, AT, PS/2 and compatibles.
2. **Products:** Volume discounts available. Annual maintenance $1056 per copy, includes technical support line, maintenance and enhancement releases. On-site training $680 per day.
   i. DesignAid $6,900. Data modeling option $1500. Real-time modules $1500.
   ii. AutoDraw.
   iii. Source/Re for reverse engineering of COBOL.
   iv. (RTrace now marketed by different company. User-definable categories and attributes. VAX-based relational database. Support VMS security features.)
3. **Tool Implementation Language:** Pascal, C.
4. **Vendor Support:** Seminars and workshops (on-site and at Nastec's Corporate Training Center), video-based training program, consultancy, support group/newsletter, hot-line.
5. **Marketed Since:** DesignAid approx 1981, AutoDraw since 1987.
6. **Size of customer base:** Information not available.
7. **Methodologies and functions at different development stages supported:**
   i. **Software specification:** Yourdon-DeMarco, Gane-Sarson methods with real-time modeling option for Ward/Mellor and Hatley, Jackson diagrams. Resource allocation to architectural components. Timing information as annotations. Chen data modeling (optional) for information modeling. ERD rule-based validation. Syntax/semantics, diagram parent-child balancing, text/diagram consistency, model consistency checking. Automated database population, no change propagation.
   ii. **Software design:** Warnier-Orr, N-S, process flow, HIPO, structure charts (option via AutoDraw for automatic generation), flow charts, decision tables, mini-specs. Supports normalization for database design. Validates Structured English against data dictionary.
   iii. **Implementation:** Code generation via Transform and TELON. Forms/screen design.
   iv. **Maintenance:** Re-engineering from COBOL.
8. **Documentation generation:** User-definable, 2167A formats.
9. **Project management support:** On-line estimation, risk assessment, management reporting, project status, review process using electronic mail, on-line task assignment, automatic status reporting, project planning and definition. Security/control access. Change reporting.
10. **Environment Characteristics:** Multi-user, remote access to database on host or LAN file server.
11. **Database:** Data dictionary implemented as database and file systems, with published interfaces and split/merge.
12. **Links to other tools:**
    a. Nastec's Transform repository.
    b. Desktop publishing via Pc-Paint or DEC Runoff.
    c. DesignAid: HostLink allows access to a database and document files (graphics and text) on an IBM host computer.
    d. PanSophic's TELON COBOL Generator.
    e. Chen and Associates SchemaGen.

f. SafeSpan: DesignAid bridge to PSL/PSA.
g. JaDesign: support for IBM's Joint Application Design (JAD) methodology.

13. **Output formats:** Published interfaces DEC VAXDocument with Encapsulated PostScript, Interleaf TPD for VAX, Nastec's NRunoff interface for EC Runoff, Xerox Ventura Publisher and Aldus PageMaker for PCs. ASCII text files.

14. **User interface:** Menu and mouse, color, on-line help, undo facility. SQL-based access to dictionary, browser.

15. **Adaptability:** Free-form text/graphics. Keyboard macros for customized functions and utilities.

Information From: Marilyn Hansen (800) 255-2689, May 6 1991.

Address: 23685 Birtcher Drive, Lake Forest, CA 92630

1. **Hardware Platforms:** DEC VAX/VMS, VAXstation, IBM PC/MS-DOS, PS/2 and compatibles, Sun/UNIX, HP 9000 workstations.
2. **Products:**
   i. ProMod/SART requirements analysis with real-time extensions. Includes ProMod/2167A report generator. PC version $3,000, VAX ranging from $3,500 to $30,000.
   ii. ProMod/TMS traceability matrix system for requirements and other development items through design $500 to $10,000.
   iii. ProMod/MD object-oriented design with architectural and detailed design, PC version $3,500, VAX ranging from microVAX $10,000 to $35,000. Includes ProMod/DC design charts.
   iv. Pro/Source source code generation in Ada and C $1,500 to $5,000.
   v. ProCap source code refinement and maintenance $1,000 to $1,500.
   vi. ProMod/CM change and configuration control, VAX only $500.
   vii. Re/Source reverse engineer code to design. (Not released in USA.)
3. **Tool Implementation Language:** Converting from Pascal to C.
4. **Vendor Support:** Training and consultancy via 3rd party.
5. **Marketed Since:** In-house use since 1980, marketed in the US since 1985.
6. **Size of customer base:** 100 users, 500 licenses in USA, 10K in Germany.
7. **Methodologies/functions supported:**
   i. **Software specification:** Yourdon-DeMarco, Hatley methods. Syntax/semantics, database/diagram consistency checking and diagram balancing. Automated database population/change propagation. Traceability.
   ii. **Software design:** Automated transform to SC from requirements, will be able to edit this transformation in next version. OOD, Constantine methods, modular hierarchy chart, or function network chart. Language independent pseudo-code.
   iii. **Code generation:** Ada, C, Pascal templates (control structures).
8. **Documentation generation:** Customizable formats, 2167A support.
9. **Environment Characteristics:** Database split/merge. Multi-project support.
10. **Database:** Data dictionary implemented by proprietary database, ASCII file import/export to other CASE tools.
11. **Output formats:** ASCII, PostScript.
12. **User interface:** Menu and mouse, windowing on VAX, on-line help, some undo.
13. **Planned enhancements:** Version 2 is under development, parts expected 3rd quarter '91.

Information From: John Moses (212) 571-3434, May 6, 1991.

1. **Hardware Platforms:** PC based tool runs under MS-Windows. IBM PC and compatibles under Microsoft windows.
2. **Tool Implementation Language:** C
3. **Tool Price:** $1,395 volume discounts available. Network version $1,545. Annual support $250/$340. OOD module $495, annual support $50.
4. **Vendor Support:** Training, consultancy, user manual includes tutorial.
5. **Marketed Since:** Since June 1988, currently Release 2.1.
6. **Size of customer base:** Over 5000 copies, approx. 7 copies per customer.
7. **Methodologies/functions supported:**
   i. **Software specification:** Requirements extraction from natural English, potentially including user-definable attributes. Gane-Sarson, Yourdon-DeMarco, Ward-Mellor methods. Optional OOD with hardware/software allocation using Booch's architectural diagram. ERDs for information modeling. Automatic diagram leveling, balancing with syntax/semantic and database/diagram consistency checking. Traceability, also testplan tracking. Automated database population/change propagation.
   ii. **Software design:** Structure charts, module specs automatically generated from mini specs. Also flowcharts, decomposition charts. Normalization and schema generation.
8. **Documentation generation:** User-definable reports, SQL custom reporting system, some desktop publishing features, matrix reporting facility, graphics. Have information needed for 2167A documentation but not yet produce these reports explicitly.
9. **Project management support:** Project planning, status reporting, change reporting, defect reporting.
10. **Environment Characteristics:** Network support, supporting 3Com, Novell, Token Ring, STARLAN and others under DOS. Data dictionary using dBASE III Plus format. Published interfaces, i.e., open architecture data dictionary/encyclopedia using dBASE III Plus file formats. Multi-user support. Database split/merge.
11. **Interfaces:** Import through ASCII and common delimiter published interface. Import command to populate requirements specification. Bulk in ASCII format (to populate data dictionary or requirements specifications). Export reports to dBASE III and spreadsheet.
12. **Links to other tools:** Spreadsheet also
    i. Currently interface with IEF/IEW and Excelerator by ASCII and Common Delimiter format. In 3rd quarter '91 a standard interface to System Architect will be supported with bridges to these tools.
13. **Output formats:** ASCII, Encapsulated PostScript. Interface to desktop publishing systems.
14. **User interface:** Menu, mouse and keyboard, windowing, some use of color. Context sensitive on-line help and novice facility. Database browser/query facility through report generation.
15. **Adaptability:** User-defined attributes test plan, on-line rules. User definable attributes for dictionary, definable attribute edit rules. User-defined attribute system (metadata) available for analysis including system variables and various system calculated metrics. User-definable diagram types using available icons.

16. **Planned enhancements:**
    i. Code generation for C and COBOL in 4th quarter '91, Ada, C++ in 2-3rd quarter 92.
    ii. Re-engineering beginning with COBOL in 3rd quarter '91.
    iii. Security/control access 3rd quarter '91.
    iv. OS/2 and AIX (RISC) version.
    v. Rapid prototyping support 4th quarter '91 for COBOL and C.
    vi. SQL server interface.
    vii. Methodology extensions for Constantine's object-oriented notation and Coad/Yourdon design editor for checking diagram consistency.
    viii. Support for C++.
    ix. Forms/screen design 3rd quarter '91, with prototyping in COBOL.
17. **Collaboration with other organizations:** Tool assistance program with IBM. Will conform to IBM's repository formats. Support of IBM AD/Cycle 1st quarter 92.

Information From: Bjorn Hemdal (301) 731-3600

Tool Summary: Methodology independent with isomorphic, interchangeable graphic and text forms.

1. **Hardware Platforms:** Sun, DEC VAXstation, Apollo workstations, VAX systems via conventional terminals, Atari PCs.
2. **Components:**
   i. Auto-G comprised of graphic editor and underlying database.
   ii. Sema semantic analyzer or diagnostic facility.
   iii. Sadmt translator from specification lanaguge to SADMT.
   iv. Dbutil design file manager.
   v. T-print for translating graphical to textual representation.
   vi. T-parse for translating textual to graphical representation.
   vii. Special utility programs, such as plot generators.
3. **Tool Implementation Language:** Currently C, planning Ada or C++ for next version.
4. **Tool Price:** $31,500 for 1st license.
5. **Vendor Support:** Training, consultancy, hot-line. Support group in UK, USA as needed.
6. **Marketed Since:** 1987 in Europe, 1989 in USA.
7. **Size of customer base:** 25 active users in Europe.
8. **Methodologies/functions supported:**
   i. **Specification:** Single formal notation that can be checked for correctness, completeness, and consistency. No explicit resource allocation. Capture of complete logical behavior and performance aspects. Concurrency, replication, timing. No traceability. Automated database population/change propagation.
   ii. **Code generation:** Ada, SADMT, C.
9. **Document Generation:** Fixed formats.
10. **Project management support:** Configuration management, but relies on operating system support for file access and time-date stamping. Extensive versioning and view capabilities.
11. **Environment Characteristics:** Multi-user, network support.
12. **Database:** Data dictionary implemented as flat file system (looking at object-oriented database for next version). Import/export as ASCII coded, T language statements.
13. **Output formats:** Primarily plotting. ASCII, PostScript.
14. **User interface:** Menu and mouse, windowing (in Sun, DEC, HP environments), some on-line help, undo. Query facility for locating instances on G diagrams. Data items or structure definitions dumped to file for external prcoessing.
15. **Planned enhancements:**
   i. 2167 report generation, perhaps user-definable formats.
   ii. Datadic data dictionary program to provide selective data dictionary query facility.
   iii. AI-based help facility.
   iv. Generation of C++ (perhaps in 4th quarter 1991).
   v. In next version, due 3rd quarter 1991, simulation and test harness capability.

Information From: Gordon Kotik (415) 494-6201, May 20 1991.
Wants a copy, FAX (415) 494-8053.

Tool Summary: Software Refinery is an interactive knowledge-based programming environment to prototype complex applications using a high-level, rule-based, executable specification language, synthesize LISP code, customize to create knowledge-based environments tailored for specification of application areas, reuse knowledge in the form of rules and logic formulas.

1. **Hardware Platforms:** Sun/SunOS, Symbolics, HP, TI Explorer and MicroExplorer workstations. X-Windows, GNU Emacs.
2. **Components:** REFINE license from $9,900 for Sun to $12,900 for Symbolics, volume discounts available. Annual maintenance contracts $900, preferred customer maintenance $3,400, university maintenance $500. Training $2,500 for first 4 at Reasoning Systems, $8,000 on-site.
   i. High-level, wide-spectrum executable specification language with compiler to transform specification into Common LISP, syntax system to integrate REFINE with existing computer languages and to create new languages and debugging system for monitoring execution of REFINE programs and creating customized debugging tools.
   ii. Knowledge base of objects including programs, logical assertions, and documents, allows user-definable object types.
   iii. C Language Subsystem reverse engineering $1,900 to $2,600.
   iv. Ada/RevEng a REFINE application currently handling 50% Ada language syntax, producing abstract syntax trees, structure charts, hypertext-style Ada source code inspector.
   v. RERUN: REFINE runtime environment to execute refinery application. >From $2,500 for Sun to $3,200 for Symbolics.
   vi. RECAST: platform on which to build C applications, includes knowledge-based representations for C programs. For development of communication systems with network modeling, reconfiguration, and simulation with automated generation of conformance tests via OSI guidelines. Interactive graphics development using state machine diagrams. $1,900 to $2,600.
   vii. INTERVISTA toolkit for building graphical user interfaces under X Windows.
   viii. User Interface Toolkit for creating interactive graphics tools used to graph (re-engineer) C, COBOL, JCL software.
   ix. DIALECT generates program language parsers and printers from grammars. Has been used for Ada, C, and others.
3. **Tool Implementation Language:** REFINE (moving to C++, 1992).
4. **Vendor Support:** Training, maintenance, consultancy, newsletter, hot-line.
5. **Marketed Since:** July 1985
6. **Size of customer base:** Over 100 licenses.
7. **Methodologies and functions at different development stages supported:**
   i. **Software specification:** Object-oriented diagrams and DFDs. No concurrency, replication, timing information, or resource allocation. Information modeling using object-oriented approach. Traceability. Syntax validation, checking for dead code. Executable specification language with assertions, supports checking for communication protocols deadlock, livelock, unreachable and unused states.
   ii. **Code generation:** Common LISP code, Ada, C, FORTRAN. Forms/screen design.
   iii. **Testing:** static analysis tools for C.

iv. **Maintenance:** Re-engineering for Ada, C, FORTRAN.

8. **Documentation generation:** User-definable formats.

9. **Project management support:** Configuration management.

10. **Environment Characteristics:** Knowledge base restoration to previous state saving, and sharing. No multi-user support, but network support.

11. **Database:** Repository implemented as database, with import/export, published interfaces. Support for generation/analysis of competing designs, save/restore knowledge based, sharing of knowledge base (no merging). Editor and file system interface based on EMACS text editor.

12. **Output formats:** PostScript.

13. **User interface:** Menu/mouse, windows, color, textual specification, menu-based knowledge-base browser and editor, on-line help. On-line documentation with browser, keyword search capability, and on-line index.

14. **Adaptability:** Knowledge base allows user-definable object types. General purpose object-oriented database, and syntactical transformation tools to adapt meaning of icons. General purpose graphics editor. Ability to create, say, natural language query language, object schema for storing decisions and reasons. Static analysis capabilities can be created in terms of rules and patterns. Free-form text/graphics.

15. **Planned enhancements:**

    i. C++ analyzers by end of '91.
    ii. CDIF and X-Windows conformance.
    iii. Translation of StP data, structure charts, and Petri-nets into REFINE and hence code generation.

Information From: Wayne Hansley (919) 881-2144, May 1991.

Tool Summary: Design specification, modeling, and simulation tool for both hardware/software systems. Interfaces to popular CASE tools for performance analysis. Can embed C code to be executed, workbench supports all C data types and storage classes. Formerly PAWS.

1. **Hardware Platforms:** Sun/UNIX, HP/Apollo, DEC VAXstation workstations.
2. **Components:** Basic workstation version $36,000.
   i. SES/design for graphical construction of system designs, behavior specified in C.
   ii. SES/sim translates a design specification into an executable simulation model, the simulation language is an object-oriented superset of C and C++.
   iii. SES/scope animation modules for observing and debugging an executing simulation model.
   iv. SES/graph.
3. **Tool Implementation Language:** C, moving to C++.
4. **Vendor Support:** Training, consultancy, hot-line, support group and newsletter.
5. **Marketed Since:** PAWS/GPSM introduced late 1970's. SES/workbench marketed since March 89. Currently version 2.0, 2.1 due out summer '91.
6. **Size of customer base:** Installed in over 100 locations worldwide.
7. **System specification:** Object-oriented approach using directed graphs, block diagrams, DFDs (Ward-Mellor or Hatley) and flow charts for specification. Supports object types, methods, instances, references and type inheritance. Objects can have multiple dimensions and can be referred to by pointers. Hardware/software allocation. Capture of timing/behavioral information via annotations on diagrams, used in simulation. Transaction-oriented, discrete event simulation, automatically generated from system design, for performance analysis. Can attach assertions for checking design correctness. Traceability. Forms/screen design.
8. **Documentation generation:** Statistical reports generated by user-specifiable forms.
9. **Project management support:** Configuration management.
10. **Environment Characteristics:** Multi-user, network support.
11. **Database:** No underlying database.
12. **Links to other tools:** IDE's StP.
13. **Output formats:** PostScript.
14. **User interface:** Menu/mouse, windows, hypertext-like on-line help, on-line reference manual, undo.
15. **Planned enhancements:**
    i. Ports to other machines underway.
    ii. Summer '91 version will include enhanced debugging, color, graphical output.
    iii. Ada, C++ supported '92.
    iv. VHDL ASCII standard.

Information From: Lois Valley (407) 984-3370

Tool Summary: Back end CASE tool.

1. **Hardware Platforms:** VAX/VMS, Sun/UNIX, Apollo and others UNIX-based systems. X-Windows.
2. **Products:**
   i. Classic-Ada Toolset $2,000, with Persistence Toolset $3,000.
   ii. Classic-Works interactive browsing capability $500.
   iii. ClassLook set of class reusable libraries to inherit capability to create X-Window environments $1,000.
3. **Tool Implementation Language:** Ada
4. **Vendor Support:** Training, consultancy, bulletin board.
5. **Marketed Since:** 2 years
6. **Size of customer base:** > 50 sites.
7. **Methodologies and functions at different development stages supported:**
   i. **Software design:** OOD methods with automated database population/change propagation.
   ii. **Code generation:** Ada.
   iii. **Testing:** Syntax and semantic Classic-Ada and Ada analysis. Automatic message tracing for debugging and performance analysis.
8. **Environment Characteristics:** Multi-user and network support.
9. **Database:** Data dictionary implemented as open database.
10. **User interface:** Command line with on-line help.

Information From: Steven (212) 686-3790, May 8 1991.

Address: 14 E. 38th Street, 14th Floor, NY 10016

Tool Summary: For real-time, process control systems. Language-independent. Code translation for Fortran. Reuse of knowledge, design information, and planning details.

1. **Hardware Platforms:** Sun, Apollo, HP, DEC workstations, VAX/VMS, IBM-PC/AT/MS-DOS, Intel/iRMX, Siemens. Planned AT&T/MS-DOS, Motoral/UNIX, Data General MV series/AOS/VS.
2. **Components:** $14,785 up to $100,000 for
   i. EPOS Code Generation Tool System. Currently Pascal, FORTRAN, Ada, PEARL.
   ii. EPOS-R for requirements specification
   iii. EPOS-S specification language and design system for system design specification using stepwise refinement. Combines graphics with PDL.
   iv. EPOS-P project specification e.g., project structure, work structure, work packages, project schedules.
   v. EPOS-A Analysis Support Package for consistency/completeness, interface, lack of ambiguity checking.
   vi. EPOS-M Management Support Package for project control, cm, progress reporting.
   vii. EPOS-D Documentation Package for automated documentation generation.
   viii. EPOS-C Communication System for user-friendly communication command system with interactive editing.
   ix. RE-SPEC reverse engineering for EPOS design specifications, from Pascal, FORTRAN.
3. **Tool Implementation Language:** Proprietary.
4. **Vendor Support:** Training, consultancy, support group, quarterly newsletter.
5. **Marketed Since:** 1984 in the USA, early 1980's in Europe.
6. **Size of customer base:** 500 copies in USA.
7. **Methodologies/functions supported:**
   i. **System specification:** System design using hardware blocks, module connection. Traceability. Syntax, completeness/consistency checking. Simulation.
   ii. **Software specification:** Ward-Mellor, Hatley methods with data/control flows, data structure, Petri-nets. Some capture of timing/behavioral information. Jackson diagrams for information modeling. Syntax/semantics and consistency checking. Prototyping for screens only. Automated database population/change propagation.
   iii. **Software design:** Function, event, module, data flow/structure, and device oriented diagrams. Consistency checking between diagrams and spec, between Ada programs and specs.
   iv. **Code generation:** C, 70-85% of Fortran, Pascal, 60-70% Ada code for concurrent systems.
8. **Documentation generation:** User-definable formats, with 2167A support.
9. **Project management support:** Project planning/scheduling with automated report generation in text/graphics. Project structure diagram, PERT and Gantt charts, current progress diagrams, work breakdown plans, network diagram, milestones. Status and change reporting. Configuration management.
10. **Environment Characteristics:** Some multi-user support.

11. **Database:** Proprietary with import/export in ASCII. Split/merge.
12. **Links to other tools:** Graphic input with CORE graphics editor, GOSS, Perspec?
13. **Output formats:** ASCII, PostScript.
14. **User interface:** Menu and mouse, windowing (on VAX under X, Sun/UNIX, and HP9000), on-line help. Database query.
15. **Planned enhancements:**
    i. Porting to PCs, Macintosh, IBM PS/2 OS/2. Porting to MS Windows for PC, available '92.
    ii. RE-SPEC for COBOL, C, Ada.
    iii. Configuration management.

Information From: Jean Baker (215) 664-1207, 17 May 1991.

Address: 1249 Greentree Lane, Narberth, PA 19072

1. **Hardware Platforms:** IBM PC, SUN Sparc, RS6000 under X-Windows.
2. **Components:** re/NuSys Workbench from $2,800 to $12,600. Components can be purchased individually.
   i. ScanFlow Designer $995.
   ii. Simulator for debugging and visual test coverage $2,800.
   iii. Program Generator for Ada, Pascal, C, COBOL, FORTRAN $3,600.
   iv. Implementor $2,800.
3. **Tool Implementation Language:** C
4. **Vendor Support:** Training, consultancy.
5. **Marketed Since:** 1989
6. **Size of customer base:** 100 licenses
7. **Methodologies and functions at different development stages supported:**
   i. **Software specification:** Flowform diagrams. Also used to support information modeling. Hardware/software allocation. Consistency checking.
   ii. **Software design:** Pseudocode with checking options for C, COBOL, FORTRAN, Pascal, Ada. Screen prototyping.
   iii. **Code generation:** Ada, C, Pascal, FORTRAN, COBOL. C++ in September 1991.
   iv. **Maintenance:** Re-engineering for Ada, FORTRAN, Pascal, COBOL. C++ after September.
8. **Documentation generation:** For printing hardcopy. User-definable formats. 2167A information available but not templates.
9. **Project management support:** Use a component approach that supports team working. No central repository, information stored in flowforms.
10. **Environment Characteristics:** Network support.
11. **Output formats:** ASCII.
12. **User interface:** Command line, and menu, windowing, on-line help, some undo.
13. **Adaptability:** Via 4GL to create high level languages.
14. **Planned enhancements:** Working with other vendors to provide links to repositories/libraries.

Information From: Ted Cannie (508) 794-3366, May 14 1991.

Tool Summary: Full life cycle support using object-oriented approaches, with open architecture and repository. Due for release in September '91.

1. **Hardware Platforms:** IBM PCs under MS-Windows, and Sun/UNIX under X-Windows.
2. **Products:** PC version $5,000, Unix $5,500.
3. **Tool Implementation Language:** C++
4. **Vendor Support:** Training, consultancy, newsletter.
5. **Marketed Since:** Prerelease versions will be made available to selected sites.
6. **Size of customer base:** Not applicable.
7. **Methodologies and functions at different development stages supported:**
   i. **Software specification:** Single diagram type supporting OOA/OOD using Booch, Coad/Yourdon, and Semaphore OO Notation. Also supports ER. Diagrams can be annotated with text. Completeness/consistency checking of database. Automated database population/change propagation.
   ii. **Code generation:** C++.
   iii. **Maintenance:** Re-engineering for C++.
8. **Documentation generation:** Via SQL interface to repository.
9. **Project management support:** Security/control access, configuration management, version control.
10. **Database:** Object-oriented repository with access via SQL interface. Split and merge.
11. **Output formats:** ASCII.
12. **User interface:** Menu and mouse, windowing.
13. **Adaptability:** Methodology tailoring via user-defined rules for completeness/consistency checking of a model. Future versions will incorporate inferencing techniques based on forward and backward chaining and pattern matching. Allow adding enities and attributes to the repository.
14. **Planned enhancements:**
    i. Multi-user, version 2 planned for 1st quarter 1992.
    ii. Code generation and reverse engineering of additional languages, likely Ada.
    iii. Timing diagrams for explicit capture of timing information.
    iv. Animation of specification.
    v. Schema generation for database design and forms/screen design.
    vi. Explicit support for 2167A documentation.
    vii. Interface to text publishing systems such as Interleaf.
15. **Collaboration with other Organizations:** Potentially with Saber-C.

Information From: Dan, Ernie Moore (415) 957-9175, May 10 1991.

Tool Summary: Graphical modeling tools, support for multiple methodologies, distributed intelligence, open architecture. Object-oriented, distributed, repository-based CASE.

1. **Hardware Platforms:** Maestro II Workstation (MVS). MS-DOS with own windowing manager and multi-tasking software. IBM PC/PS/2 compatibles with workstation connected to UNIX-based file server on DEC VAX or Philips machines through Ethernet.
2. **Components:** Tool price for single user $13,000.
   i. Object Management System (OMS) provides meta model, allows customizing data model, or integrating SoftLab and 3rd party tools. Data associated with software development process is stored in a repository organized by OMS. It provides access rights, versions and variants, distributed data storage and access, elementary and user-defined transactions. Processor and geographical distribution, with copy of data model on all servers.
   ii. MGEN application generator expected second half '90.
   iii. DDT Diagram Design Tool.
   iv. LDT Layout Design Tool.
   v. GED Graphics Editor.
   vi. TEXT Text Editor.
   vii. CMS Configuration Management System.
   viii. PMS Project Management System.
   ix. COMM Communication Packages.
3. **Tool Implementation Language:** PROLAN, C-like.
4. **Vendor Support:** Training, consultancy, newsletter.
5. **Marketed Since:** Maestro I introduced in 1978. Maestro II marketed since autumn 1989 in Europe, January 1990 in USA.
6. **Size of customer base:** 23,000 Maestro I workstations worldwide.
7. **Methodologies/functions supported:**
   i. **Software specification:** SA, LSDM, SSADM methods. Merise for information modeling. Automated database population/change propagation.
   Capture of timing/behavioral information? Traceability?
   ii. **Software design:** SD method. Schema generation for database design.
   iii. **Code generation:** Either by 2-way interface with generators via the data dictionary, or by knowledge-based generators that produce logic and control code, screen definitions, database definitions and schema. Uses generator engine with spec based and knowledge base parts. Currently have knowledge base support for IBM DB2, COBOL, working with HP for HP9000 and others with C.
8. **Documentation generation:** User-definable formats? 2167A support?
9. **Project management support:** Own text editor/word processor, office automation software (electronic mail, diary, etc.), Workbreakdown structure. coordination and communication. Workbreakdown structure. Configuration management, versioning, audit trail, change rollback, change reporting, defect reporting, security/control access.
10. **Environment Characteristics:** Multi-user support, network (heterogeneous) support via LAN, Ethernet.

11. **Database:** Server-based object-oriented repository, C interface. Database split/merge.
12. **Links to other tools:**
    i. Communication packages to link Maestro II to variety of common machines including IBM, Siemens, DEC VAX, Bull, ICL, and any UNIX computer.
    ii. Interfaces to IEW, and Micro Focus COBOL.
    iii. Trimarand, Inc. code generator METAgen in PC/LAN environment, knowledge-based generator embedded in Maestro II. Expect release mid 90.
    iv. Aeon for requirements extraction from natural English.
13. **Output formats:** Postscript. Essentially all UNIX file system devices.
14. **User interface:** Menu and mouse, windowing, on-line help (hypertext). Database browser/query facility.
15. **Adaptability:** Designed to be fully extensible and customizable. Programmable user-interface. Modifiable graphic notation for diagrams.
16. **Standards conformance:** IRDS, AD/Cycle.
17. **Planned enhancements:**
    i. UNIX, OS/2 based workstations, HP and IBM hardware.
    ii. Object editor, inheritance, and more object facilities such as functions, subtyping.
    iii. Object-oriented query language.
    iv. Additional DBMS interfaces, including DB2, Predict.
    v. Check-in/-out capabilities.
18. **Collaboration with other organizations:** IBM with AD/Cycle.

Information From: Thomas Radi (714) 625-6147

Address: 3627 Padua Avenue, Claremont CA 91711

Tool Summary: Set of tools to take real-time structured analysis input and support design and testing. C version (CISLE) available.

1. **Hardware Platforms:** VAX and MicroVAX, Sun, DEC, Apollo workstations, PCs, others.
2. **Components:**
   i. ADADL Ada-based PDL, $5,000 to $18,800.
   ii. DocGen document generator for MIL-STD documentation $4,600 to $17,000.
   iii. TestGen Ada design and code testing tool $4,600 to $17,000s
   iv. GrafGen graphical Ada design system $7,000 to $10,500.
   v. ASE Ada/ADADL syntax directed editor $1,390 to $7,800.
   vi. ARIS Ada/ADADL RTSA requirements interface system interfaces with Teamwork to create first cut at an Ada program structure working from DFDs, $7,500 to $14,500.
   vii. AIEM on-line debugging and analysis tools $5,200 to $15,200.
   viii. QualGen quality metrics $4,600 to $17,000.
   ix. RETT requirements traceability $4,600 to $17,000.
3. **Tool Implementation Language:**
4. **Vendor Support:** Training, consultancy, support group meetings at Tri Ada.
5. **Marketed Since:** 1985
6. **Size of customer base:** 46 organizations
7. **Methodologies and functions at different development stages supported:**
   i. **Software design:** Input from RTSA database compatible with Teamwork, Excelerator, StP, Structured Architect. Produces OOD Booch/Buhr-like diagrams, uses templates for documentation and pseudo-code design. Provides structure charts, quality and complexity analysis. Forward and backward traceability from requirements to design, code, and tests. Automated database population/change propagation.
   ii. **Code generation:** Ada
   iii. **Testing:** Design review expert assistant, unit test strategy generator, test effort estimator, test coverage analyer.
   iv. **Maintenance:** Re-engineering of Ada.
8. **Documentation generation:** User-definable formats and 2167A support.
9. **Project management support:** Project planning, status reporting, Security/control access.
10. **Environment Characteristics:** Multi-user, network support.
11. **Database:** Data dictionary implemented as database.
12. **Links to other tools:** Teamwork, Excelerator, StP, Structured Architect.
13. **Output formats:** Compatible with Interleaf, RUNOFF, nroff/troff and other word processors.
14. **User interface:** Text based. Database browser/query facility.
15. **Adaptability:** User-expandable interfaces to the database.

Information From: Chuck Williams (301) 224-3710

Tool Summary: The Virtual Software Factory (VSF) is a meta-CASE tool. Intended to support integration at the information level rather than the tool level. Addresses method and design database integration. Providing for verifiabiltiy, traceability, and tailorability across the life cycle. Available instances: HOOD-SF and SSADM-SF. Other methods implemented by Systematica and VSF users include CORE requirement capture method, and Mascot3/Ada (British MOD standard for real-time systems development).

Supports meta-modeling constructs such as multiple inheritance across hierarchies, multiple design databases, automatic translation between methodologies, and specification and enforcement of rules for methodologies. Schemas can be described using the VSF formalisms. Engineer specifies: (1) required documentation, say 2167, via MWB; (2) traceability model between design objects or earlier/later project phases; (3) filter mechanism to implement checking rules for static diagnostics, underlying formalism is a decidable second-order logic. VSF comes with a high-level, internal logic specification language resembling PROLOG, supports beliefs, belief generation rules, pre/post-conditions, etc. No simulation. Built-in file manager, design databases created by VSF are stored in a VSF specific-format. Documents stored/retrieved using a hypertext approach. Design fragments can be conserved to another tool whose output can then be merged (with conflict checking) back into the workbench. Host environment is a shell around VSF, user uses the configuration and project management tools available in the host environment. Not multi-user. Does merge design information into a central database via VSF merge facility.

1. **Hardware Platforms:** Sun, DECstation workstations, IBM PS/2 under OS/2, IBM RS6000, VAXstation.
2. **Components:** $200,000, Systematica are also paid a percentage of licence fee from CASE tools developed with VSF.
   i. Methods Engineering Workbench (VSF/MWB). Primarily textually-oriented to define graphics environment for the workbench. Used to define methodologies and configure the design environment.
   ii. Analyst Workbench (VSF/AWB). Graphical and textual editors that were predefined for methodologies in the MWB.
3. **Tool Implementation Language:** Ada, approx. 300,000 lines of source code.
4. **Vendor Support:** Training, Consultancy.
5. **Marketed Since:** March 88.
6. **Size of customer base:** 60-70 in Europe.
7. **Planned enhancements:** Version for IBM PS/2.
8. **Collaboration with other organizations:**
   i. DEC.
   ii. COGNOS, Inc.
   iii. Focus.
   iv. IBM for AD/Cycle.

Information From: 44 202 297292

Tool Summary: Instantiation of VSF/AWB.

1. **Hardware Platforms:** Sun, DECstation workstations, IBM PS/2 under OS/2, IBM RS6000, VAXstation.
2. **Product:** 7,000 pounds
3. **Tool Implementation Language:** Ada.
4. **Vendor Support:** Training, Consultancy.
5. **Marketed Since:** 1988 in Europe, just starting in USA.
6. **Methodologies/functions supported:**
   i. **Software Specification:** DFDs, DSDs for information modeling, entity life history diagrams. On-line validation of user actions. Consistency and completeness checking with diagram/database consistency checking.
   ii. **Design:** Dialogue design. Database design through 3rd normal form.
   iii. **Code Generation:** Some.
7. **Documentation generation:** User definable formats only achievable through tailoring using the methodology workbench. 2167A information present but not formatted.
8. **Project management support:** QA support, problem reporting.
9. **Environment Characteristics:** Multi-user, network support.
10. **Database:** Central repository implemented as IKBS, separate partial knowledge bases on workstations can be implemented as database or by file systems as appropriate for environment.
11. **Output formats:** ASCII, PostScript, interface to desktop publishing systems.
12. **User interface:** Menu, mouse, windowing. Navigation.
    On-line help/undo facility?
13. **Adaptability:** Methodology tailoring via VSF.
14. **Standards conformance:** SSADM British government standard for EDP system development.
15. **Planned enhancements:** 2167A documentation support.

Information From: Chuck Williams (301) 224-3710

Tool Summary: Instantiation of VSF/AWB.

1. **Hardware Platforms:** Sun, DECstation workstations, IBM PS/2 under OS/2, IBM RS6000, VAXstation.
2. **Product:** 7,000 pounds sterling, $17,000.
3. **Tool Implementation Language:** Ada.
4. **Vendor Support:** Training, Consultancy.
5. **Marketed Since:** 1988 in Europe, just established USA affiliate.
6. **Size of customer base:** None in USA.
7. **Methodologies/functions supported:**
   i. **Software Specification:** Object-oriented methods.
   ii. **Design:** Ada PDL.
   iii. **Code Generation:** Ada.
8. **Documentation generation:** User definable formats only through tailoring with the methodology workbench. 2167A information available but not formats.
9. **Environment Characteristics:** Multiple projects supported. Multi-user, network support.
10. **Database:** Split, merge.
11. **Output formats:** ASCII, PostScript, HPGL, interface to desktop publishing systems.
12. **User interface:** Mouse, windowing, on-line help. Browser.
13. **Adaptability:** Methodology tailoring via VSF.
14. **Standards conformance:** HOOD defacto standard for European aerospace Ada development.
15. **Planned enhancements:** 2167A documentation support.

Information From: Jan Smedley (205) 837-2400

1. **Hardware Platforms:** Sun, VAX
2. **Tool Implementation Language:** Ada
3. **Tool Price:** Free
4. **Vendor Support:** Training, newletter, consultancy, hot-line.
5. **Marketed Since:** Available since 1987.
6. **Size of customer base:** >200 installations
7. **Methodologies and functions at different development stages supported:**
   a. **System specification:** F-net, IDEF diagrams. Hardware/software allocation. Simulation. Traceability. Automated database population. Capture of timing information?
   b. **Software specification:** Various diagrams.
   c. **Software design:** Various diagrams.
   d. **Code generation:** Ada
8. **Documentation generation:** User-definable formats, 2167A templates.
9. **Project management support:** Configuration management, status reporting, change reporting.
10. **Environment Characteristics:** Network support.
11. **Database:** Repository implemented as ERA database. Split/merge, import/export.
12. **Links to other tools:**
13. **Output formats:** ASCII for 2167A documentation, PostScript for graphs.
14. **User interface:** Menu and mouse, windowing, color, on-line help, some undo. Database browser/query facility.
15. **Adaptability:** Free-form text/graphics. Some methodology tailoring.
16. **Planned enhancements:**
    i. Multi-user support.
    ii. X-Windows.
    iii. Potentially OOD support.

Information From: Cathy Chou (703) 352-8500, May 10 1991.

Tool Summary: For definition, analysis, and simulation of system designs based on Engineering Block Diagrams.

1. **Hardware Platforms:** Apollo/Aegis, Sun/UNIX, Dec VAXstation/Ultrix workstations. IBM PS/2.
2. **Products:**
   i. TAGS $6,500. Includes:
      a. Input/Output Requirements Language (IORL),
      b. Diagnostic Analyzer (DA),
      c. Automated Configuration Management (CM),
      d. Simulation System with simulation compiler and Executable Ada Code Generator (ECG) are no longer marketed.
   ii. Requirements Validation Tool Suite (RVTS). Currently on IBM PC compatibles under DOS, being ported to X-Windows and Ultrix. Requirements stored in a relational database. Supports automatic extraction of natural language-based requirements statement and their cataloguing into a hierarchical database for sorting, analysis, tracing, design mapping, and report generation. Multi-user network environment with centralized database manager. Output formats: ASCII text files. User interface: menus. Requirements Tracer (RT) second generation RVTS, marketed since: December 1990. $12,500 for 1st seat, $6,500 thereafter.
3. **Tool Implementation Language:** C
4. **Vendor Support:** Training, consultancy, forming support group, newsletter.
5. **Marketed Since:** TAGS since 1984.
6. **Size of customer base:** In the hundreds.
7. **Methodologies/functions supported:**
   i. **System specification:** Functional decomposition with object-oriented. RT can import an ASCII text file and extract requirements from this. With traceability and resource allocation.
   ii. **Software specification:** Own methods. Capture of timing/behavioral information. No information modeling. Syntax/semantics, diagram balancing, database/diagram consistency checking.
   iii. **Software design:** Control flow diagrams.
   iv. **Code generation:** No longer marketed.
8. **Documentation generation:** Not in TAGS, with user-definable formats in RT. 2167A support via other documentation tools.
9. **Project management support:** Configuration management, change reporting, version identification, time stamping. Security/control access, some status reporting, defect reporting.
10. **Environment Characteristics:** Multi-user and network support.
11. **Database:** Central. RT import/export in ASCII, TAGS uses library routines accessed with user-defined C and FORTRAN programs. No database split/merge. Data dictionary has no textual descriptions.
12. **Links to other tools:** Interleaf and Mentor Graphic's Context publishing software.
13. **Output formats:** PostScript, Interleaf for 2167A.

14. **User interface:** Menu/mouse, windowing, on-line help, some undo. Database browser/query facility,
15. **Planned enhancements:** Port to IBM's AIX operating system.

Information From: Dick Taylor (703) 849-1481.

Tool Summary: For planning, analysis, design, construtions, and maintenance.

1. **Hardware Platforms:** PC, workstation for development, mainframe for code generation.
2. **Products:** Price?
3. **Tool Implementation Language:** C++
4. **Vendor Support:** Training, consultancy, hot-line, bulletin board.
5. **Marketed Since:** 1986
6. **Size of customer base:** Over 350 users.
7. **Methodologies/functions supported:**
   i. **Software specification:** DFDs, ERs, action diagrams. Automated database population/change propagation.
   ii. **Software design:** SCs, screen/forms design.
   iii. **Code generation:** Code and screen generation. Schema generation.
   iv. **Testing:** COBOL generation for testing based on diagrams.
   v. **Maintenance:**
8. **Documentation generation:**
9. **Project management support:** Security/control access, history tracking, version control.
10. **Environment Characteristics:** Multi-user and network support.
11. **Database:** Encyclopedia implemented as object-oriented database. Check-in, check-out, split/merge, import/export facility.
12. **Output formats:**
13. **User Interface:** Menu/mouse, windowing, color.
14. **Planned enhancements:**
    i. CUI compliance on SAA platforms.
    ii. New diagram facilities.
    iii. Reverse engineering.
    iv. Automated first cut at design.
15. **Compatibility:** With Ad/Cycle.

Information From: Mark Luciw (301) 220-2430, May 10 1991.

1. **Hardware Platforms:** HP 9000, HP/Apollo, Sun, VaxStations. UNIX and X-Windows.
2. **Products:**
   i. AGE $50,000 for single-user, volume discounts available. Includes:
      a. ASA for requirements analysis and system validation, includes ASA-ED editing tool, ASA-PM modeling, ASA-PG test generation.
      b. GEODE for designing and code generation, includes GEODE-ED editor, GEODE-SM simulator, GEODE-RT run time generator.
      c. MCAG linking module for traceability.
   ii. Logiscope for software quality analysis.
3. **Tool Implementation Language:** Pascal, C
4. **Vendor Support:** Training, consultancy, newsletter.
5. **Marketed Since:** 1990 (as AGE), ASA and GEODE for 3 to 4 years.
6. **Size of customer base:** Over 1000 copies.
7. **Methodologies/functions supported:**
   i. **System specification:** SADT/IDEF method with resource allocation and some capture of timing information. Consistency, functional decomposition checks. Simulation. Traceability.
   ii. **Software specification:** SADT Datagrams for information modeling. Automated database population/change propagation.
   iii. **Software design:** SDL notation.
   iv. **Code generation:** C.
   v. **Testing:** See Logiscope.
   vi. **Maintenance:** See Logiscope.
8. **Documentation generation:** User-definable formats.
9. **Project management support:** Some security/control access, change reporting via tracing facility.
10. **Environment Characteristics:** Multi-user, network, multi-project support.
11. **Database:** Data dictionary as part of ASA, implemented as file system. All information maintained in ASCII files. Import/export facility, split/merge.
12. **Output formats:** PostScript. Interface to Interleaf and FrameMaker.
13. **User interface:** Menu and mouse, windowing, some color, on-line help, some undo. Database browser/query facility,
14. **Standards conformance:** SDL/CCITT, X Windows.
15. **Planned enhancements:**
   i. Generation of Ada code by June '91.
   ii. Object-oriented support through LOVE programming support environment, will be made available as part of AGE and will generate C++.
   iii. Tie in user-interface toolkits.

Information From: (617) 890-2273, May 21 1991.

1. **Hardware Platforms:** IBM PC
2. **Products:**
   i. Professional $2,795, or with prototyper $3,395.
   ii. LAN Professional (3 nodes) $7,895.
3. **Tool Implementation Language:** Mainly C.
4. **Vendor Support:** Training, consultancy, newsletter.
5. **Marketed Since:** 1985.
6. **Size of customer base:** >8000 users, >3000 installations.
7. **Methodologies and functions at different development stages supported:**
   i. **Software specification:** Yourdon-DeMarco, Gane-Sarson methods. Chen, ER diagrams for information modeling. Diagram balancing, consistency checking (diagrams are validated as created). Automatically populated database and change propagation.
   ii. **Software design:** Yourdon-Constantine, Page-Jones methods with automatic generation from specification and design complexity measurement. SQL generation for database design. Screen prototyping.
8. **Documentation generation:** Fixed document types, some contents can be customized. 2167A information available but not formatted.
9. **Project management support:** Security/control access.
10. **Environment Characteristics:** Multi-user and network support. Multi-project.
11. **Database:** Server-based repository implemented as file system and database with published interfaces. Split/merge.
12. **Output formats:** PostScript, tiff, ASCII, other.
13. **User interface:** Menu and mouse, windowing, on-line help, undo facility. Database browser/query facility.
14. **Planned enhancements:**
    i. Scheme extraction from database.
    ii. Code generation for C and COBOL later in '91.

Information From: David Stephenson (703) 758-1501

Address: 1501 Broadway, New York, NY 10035

Tool Summary: Primarily for business software.

1. **Hardware Platforms:** IBM PC-AT, PS/2 and compatibles, DOS.
2. **Components:** Tool price $1,995 single user. User Interface Generator option for screen prototyping and code generation no longer marketed.
3. **Tool Implementation Language:** Mainly C.
4. **Vendor Support:** Technical support line, training, consultancy, newsletter.
5. **Marketed Since:** 1984, currently Version 6.1.
6. **Size of customer base:** 4000 copies.
7. **Methodologies/functions supported:**
   i. **Software specification:** Some requirements extraction. DFDs, ST, etc. diagrams. Diagram balancing, database/diagram consistency checking. Traceability only through to process specs. Chen for information modeling. No automated database population, but notification of needed database changes.
   ii. **Software design:** SC method. Schema generation for DB3.
8. **Documentation generation:** Fixed report formats, merges text/graphics. No 2167A support.
9. **Environment Characteristics:** Single-user, not recommended for use on a network.
10. **Database:** Data dictionary implemented as DB3. Split/merge facility.
11. **Output formats:** ASCII, PostScript, HPGL.
12. **User interface:** Menu and mouse, color, on-line help/tutorial, undo facility. Database browser/query facility.
13. **Adaptability:** Free-form graphics.

Information From: David Stephenson (703) 758-1501, May 14 1991.

Tool Summary: For real-time software.

1. **Hardware Platforms:** UNIX under X.
2. **Components:** Tool price $1,995 single user, $2,495 multi-user version. Includes Code Generator (CGEN) for Aqa, C, Pascal.
3. **Tool Implementation Language:** Mainly C.
4. **Vendor Support:** Technical support line, training, consultancy, newsletter.
5. **Marketed Since:** 1990 in USA, currently Version 4.
6. **Size of customer base:** 20-30 customers in Europe, 6-7 USA.
7. **Methodologies/functions supported:**
   i. **Software specification:** DFDs, ST, etc. with requirements extraction. Hardware/software allocation and capture of timing information. Chen information modeling. For static analysis syntax/semantic checking, diagram balancing, database/diagram consistency. Automated database population and flagging for needed changes. Traceability through to code.
   ii. **Software design:** Structure charts and module specs.
   iii. **Code generation:** Ada, C, Pascal.
8. **Documentation generation:** 2167A and user-definable formats.
9. **Project management support:** Configuration management, access control, change reporting.
10. **Environment Characteristics:** Multi-user and network support NetBIOS compatible networks, e.g., Novell, 3Com.
11. **Database:** Repository implemented as database. Database split/merge.
12. **Output formats:** PostScript, HPGL, HPLaserjet (PCL).
13. **User interface:** Menu and mouse, windowing, context-sensitive on-line help, undo facility. Database browser/query facility.
14. **Adaptability:** Free-form graphics.
15. **Planned enhancements:** Support for simulation/prototyping.

—— End Included Message ——

APPENDIX B. Final Report (SEI Substudy)

Special Report
SEI-91-SR-4
June 1991

A Comparison of Ada 83 and C++

Nelson H. Weiderman

Technology Division

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

This work is sponsored by the U.S. Department of Defense. The views and conclusions contained in this document are solely those of the author(s) and should not be interpreted as representing official policies, either expressed or implied, of Carnegie Mellon University, the U.S. Air Force, the Department of Defense, or the U.S. Government.

Table of Contents

A Comparison of Ada 83 and C++

Executive Summary

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 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-1815A-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 criteria [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 valida-

tion 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 lifecycle 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 Wasnington 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. LTC 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. Language 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

APPENDIX C. Final Report (CTA Substudy)

This appendix contains a survey and analysis of productivity and cost data on Ada and C++ Software Development programs.

SPECIAL STUDY

SURVEY AND ANALYSIS OF PRODUCTIVITY AND COST DATA ON Ada AND C++ SOFTWARE DEVELOPMENT PROGRAMS

System Design and Analysis Support (SDAS)
Contract #F19628-87-D-0016-000112

CDRL #DI-S-3591 A/T

Prepared For:

HQ Electronic Systems Division/SR-1
Space and Missile Warning Systems Program Office
Integrated Tactical Warning and Attack Assessment Program Office
Hanscom AFB, MA 01731-5000

26 June 1991

CTA INCORPORATED
175 Middlesex Turnpike
Bedford, MA 01730

TABLE OF CONTENTS

SURVEY AND ANALYSIS OF PRODUCTIVITY AND COST DATA ON ADA AND C++ SOFTWARE DEVELOPMENT PROGRAMS

1.0 TASK OVERVIEW

1.1 Purpose

The purpose of this task was to provide technical data and analysis results to ESD/SR-1 and SAF/AQK to support Government decision making for future software programs in Ada and C++ languages.

1.2 Objective

The objective of this special study was to perform a survey of productivity and cost data for Ada and C++ software development programs.

1.3 Background

Portions of the Integrated Tactical Warning and Attack Assessment (ITW/AA) System-of-Systems (to include the Cheyenne Mountain Upgrade (CMU) program) are currently being developed using the Ada programming language. The majority of the ITW/AA programs require a software intensive development effort. As such, it is critical that correct decisions regarding programming language choice be made to ensure life-cycle cost effectiveness. A study was necessary to collect life-cycle cost and productivity data, the results of which can be used as input into devising acquisition strategies for future ITW/AA programs.

1.4 Scope

The scope of this task covers: 1) surveying and analyzing productivity and cost data on Ada and C++ software development programs, and 2) providing a report of comparative cost and performance analyses between Ada and C++.

2.0 METHODOLOGY

2.1 Approach

The approach to acquiring software productivity and cost data for C++ and Ada was straightforward. We developed an initial list of likely sources of data based upon staff knowledge and established relationships. Telephone contacts with these individuals led to additional possibilities resulting in a list of over eighty contacts. Using the SOW list of desired data elements as a framework, telephone interviews were used to capture what data was available from each source. Selected quantitative results from these interviews were forwarded to SAF/AQK as acquired in order to provide and early characterization of the data obtainable. Some of the contacts represented organizations that have collected productivity, cost, and quality data from institutional projects. Because of the sensitive proprietary nature of such data, open sharing of the details was rare. Aggregate data from these sources was acquired in many instances. Identification and description of these databases is a part of our results, and may be useful in subsequent studies.

2.2 Data Sources

Table 2-1 lists the name, organization, phone number, and area of interest for all those individuals actually contacted. An indication of additional contacts suggested by this person is also offered. Some of the contacts have established databases of software productivity and cost data as shown. These databases are described more fully in Section 3.3 of this report.

2.3 Data Collection

CTA INCORPORATED collected a large sample size of data on the productivity and cost of producing code in Ada and C++. We placed the greatest emphasis on the C++ portion of the data collection because the language is newer than the Ada language and there were fewer completed programs to evaluate. The anticipated size for the Ada productivity and cost analysis was on the order of 300 programs while the C++ sample size was anticipated to be approximately 50 programs.

Data was collected from software houses and from our corporate experience base. We collected and documented background information on programs which used either software language and their productivity and cost parameters. Quantitative and qualitative data was also collected to support our data analysis phases. We also found that our interviews led to new data sources for our contact list. Updates to the list were presented at our weekly status briefings.

	A	B	C	D	E	F	G
1	Potential	Contacted	Data	NAME	ORGANIZATION	Area	PHONE NUMBER
2	80	62	8				
3	1			Ahern, Dennis	Westinghouse/Aerospace SW	SIGAda Reuse WG Chair	(301) 993-6234
4	1			Agresti, Bill	MITRE		(703) 883-7577
5	1	1		Apgar, Hank	MCR (Silver Study)	SSD Database/POC Capt Martin (213)	(805) 964-9894
6	1	1		Bachman, Bret	Rational		(408) 496-3700
7	1	1		Babel, Phil	ASD	SDCCR	(513) 255-3656
8	1			Basili, Vic	SEL/ U. of Maryland	Ada/Fortran Database	
9	1	1		Bauer, Paul	US West	Technology	(303) 541-4088
10	1	1		Bizoke, George	Lockheed	Proprietary/AF request may be honored	(408) 743-1106
11	1	1		Booch, Grady	Rational	C++ Data Components	(303) 987-1874
12	1			Brenner, Neal	Tecolote Research(West)		
13	1	1		Card, Dave	CSC	New Lead/Rajich/Aug Database	(301) 572-3815
14	1	1		Carleton, Anita	SEI	Measurement Group	(412) 268-7718
15	1	1		Caswell, Deborah	HP	SQA, etc.	(415) 857-6374
16	1	1		Cheadle, Bill	MMC	Ada (C) Cost Database	(303) 971-6740
17	1	1		Chrusciki, Andy	Rome Laboratories	SW QA	(315) 330-4476
18	1	1		Comewau, Greg	C++ Vendor	Renew contact	(718) 945-0009
19	1	1		Conners, Nellie	Rational	Stanfins -R/ Productivity Paper	(408) 496-3692
20	1	1		Costenbader, Jay	GSFC	C++ Project	(301) 286-5292
21	1	1		Crafts, Ralph	Ada Strategies	Ada Information	(304) 725-6542
22	1	1		Dean, Joe	Tecolote Research, (East/ESD)		(617) 275-3014
23	1			Defense System Mgmt College		Project Management Training	Ft. Belvoir, Va
24	1			Dion, Ray	Ratheon	Cost of Quality/No contact	(508) 440-2236
25	1	1		Dominy, Bob	GSFC	C++ Activity	(303) 286-4196
26	1	1		Duvall, Lorraine	Syracuse University	DACS History	(315) 479-8624
27	1	1		Edwards, Steven	GSFC	C++	(301) 286-6676
28	1	1		Ellis, Walter	FAA/IBM	Life-Cycle Cost Data	(301) 640-2889
29	1	1		Ferens, Dan	AFIT	Model "Executor"	(513) 255-4845
30	1	1		Fisher, Kurt	CONTEL		(703) 818-5240
31	1			Frazer, Tom	IDA		(703) 845-2132
32	1	1		Gaffney, John	SPC		(703) 742-7116
33	1	1		Gallerath, Dan	Gallerath Associates	Modeling/Seer/Jensen Model	(213) 670-3404
34	1			Gelprin, Dave	S/W Practices		(612) 541-1431
35	1	1		Grady, Robert B	HP	Metrics	(408) 746-5103
36	1			Graham, Dave	Hewlett Packard	Ada Project	(408) 447-5733
37	1	1		Grass, Judy	AT&T/Bell Labs		(908) 582-6905
38	1	1		Gross, Steve	Nav. Center for Cost Analysis	Database(Silver)	(703) 746-2311
39	1	1		Hetzel, Bill	SW Quality Engineering	SQA	(800) 423-8378
40	1	1		Hilton, Jane	AT&T	Software Development	(908) 457-2490
41	1	1		Holman, Richard	Hewlett Packard	C++ Project	(408) 447-7312

Table 2.1: Data Source List

	A	B	C	D	E	F	G
42	1	1	1	Jensen, Randall	Computer Economics, Inc.	SW Sizing Model	(213) 827-7300
43	1	1	1	Jones, Capers	SPR, Inc.		(617) 273-0140
44	1	1		Kahoutek, Henry	HP/Ft. Collins	General Quality	(303) 229-3169
45	1			Levine, Trudy	Fairleigh-Dickinson U.	SIGAda/Reuse Components	(201)692-2000
46	1	1	1	Loesh, Bob	JPL/STI	Ada Project/FAA Weather	(818) 397-7522
47	1	1		Mancl, Dennis	AT&T/Unix System Labs	Experience with C++	(908) 580-4391
48	1	1	1	Martin, Capt. Dale	Space Division, SMCR	Survey Data	(213) 363-0067
49	1	1		Markham,Dennis (?)	AdaIC	Ada Usage Database	(703) 685-1477
50	1	1	1	McGarry, Frank	GSFC/SEL	Ada/Fortran Database	(301) 286-6846
51	1			McGarry, Jack	NUSC	Metrics Activity	(401) 841-3834
52	1	1		Miller, Katheran	IITResearch	Cost Model Accuracy	(301) 459-3711
53	1			Moore, Bob	MICOM/Huntsville	Possible Database	?
54	1			Najberg, Andy	TASC (ESD)	Number disconnected	(617) 944-6850
55	1	1		Nyberg, Karl	Consultant	Ada Commentary / Service	(703) 281-2194
56	1	1		Peterson, Brock	Rational	X-Lib/Ada: OO Programming	(408) 496-3600
57	1	1		Pfleeger, Shari	CONTEL	C++ Metrics/Planning..No data	(703) 818-4498
58	1	1		Potter, Marshall	Asst. Sec. Navy:S/W Master Plan		(202) 695-2843
59	1	1		Pullen, Karen	MITRE/ESD	Mitre Cost Center	(617) 271-2301
60	1	1		Radzisz, Barbara	DACS	SW Databases	(315) 336-0937*
61	1	1		Rajich, Zaclav	Wayne State	C++ Instructor	(313) 769-7528
62	1	1	1	Reifer, Don	RCI, Inc	Cost Model/Database	(213) 373-8728
63	1	1		Rozum,Jim	SEI	Measurement Program	(412) 268-7770
64	1	1		Roush, Bernie	NASA/JSC	COCOMO Calibration	(713) 483-9092
65	1			Rone, Kyle	IBM	Reuse/Ada Components	(713) 282-8399
66	1	1		Schneidewind, Norm	Naval Post Graduate School	IEEE QA Metrics Chair	(408) 646-2719
67	1			Seidowitz, Ed	GSFC	Ada Design	(301) 286-7631
68	1	1	1	Silver, Aaron	Martin Marietta	Cost Models/Data	(303) 971-6730
69	1	1		Solomond, John	Director, AJPO	Ada Applications	(703) 614-0208...11
70	1	1		Stanfield, Jack	Loral/COS	Ada projects	(719) 594-1437
71	1	1		Stark, Mike	GSFC	Ada design	(301) 286-5048
72	1	1		Sulgrove, Bob	NCR	IEEE Productivity Std.	(513) 445-1064
73	1	1	1	Sweet, Wm	Quantitative SW Mgmt, Inc	SLIM Productivity Model, +	(703) 790-0055
74	1	1		Sylvester, Dick	MITRE	SW/HW Engineering	(617) 271-2301
75	1	1		Tibideau, Bob	GRC	C++ Use	(205) 922-1941
76	1			Van Duyn	Xerox	Process Controller	
77	1	1		Vasilescu	CTA Consultant	MIS	
78	1	1		Vendl, Dean	NCR	CAD Applications	(303) 223-5100 x3
79	1			Wiener, Richard	UCCS	SW Development/Languages	(719) 687-2517
80	1	1		Wild	Cadre	C++/Ada Maintenance	(401) 351-5950x1
81	1			Wolverton, Ray	TRW	Cost Modeling/Data	(213) 414-5515
82	1	1		Zimmerman, John	SW Productivity Research	Modelling	(617) 273-0140

Table 2.1: Data Source List (Cont'd)

2.4 Data Analysis

2.4.1 Productivity and Cost Data Analysis

The productivity and cost data analysis effort was designed to extract and evaluate data from both Ada and C++ completed programs. As we collected detailed information on these programs, we tabulated the results to answer the specific questions listed in SOW paragraphs 3.1.1 a thru q. Specifically:

a. Overall Effort - Man-years and dollars associated with design; with integration; with test; with implementation.

b. Overall Productivity - Man-years and dollars associated with the total project divided by initial LOC projection; by final LOC experience; by final memory space utilization.

c. Scrap and Rework Rate/Cost - Design effort, coding effort, and associated integration/test activity (including cost/effort associated with documentation and manuals) which was unintentionally discarded/replaced.

d. Errors - Errors per KLOC at conclusion of design; integration; test; implementation.

e. Comparison of cost at completion with initial prediction of cost.

f. Comparison of elapsed calendar time at completion with initial prediction of schedule.

g. Percent of total code reused (reused lines as percent of total lines):

- from within the project
- from outside the project
  - same language
  - different language

h. Number of lines of code as compared with projected lines for predecessor language. (Note: LOC may be better shown as memory space.)

i. Investment in general purpose software development tools and environments, including associated training.

j. Investment in language peculiar tools, including compilers and test/verification tools, including associated training.

k. Personnel costs associated with direct design; with direct coding; with direct integration and test; with direct implementation.

l. Indirect personnel costs.

m. Other costs.

n. Total cost of development and implementation.

o. Projected or proven functional operational benefits from system implementation (savings and cost avoidance, shown separately).

p. Ratio of language peculiar costs to total costs; to savings and cost avoidance.

q. Other relevant data as available.

2.4.2 Ada and C++ Comparative Analysis

The second phase of analysis was to compare the Ada and C++ productivity, cost and performance information collected in the survey above. Once the data was categorized as directed by SOW paragraphs 3.1.1 a thru q, a matrix was developed showing how each of the parameters compared according to the software language used. Our initial intent was to discover and present characteristics of Ada and C++ which would give either an advantage in some applications.

2.5 Reports and Schedule

2.5.1 Plan of Accomplishment Briefing and Plan of Accomplishment Report

Figure 2.1 was the schedule for the survey task. The POA Briefing was presented on 2 May 1991 at ESD/SR-1, Hanscom AFB, MA. The POA Briefing documented the results of work completed under SOW paragraph 3.1. The POA Document proposed a schedule for completion of the remainder of the task (Figure 2.1). Three copies of the final POA Document were delivered on 8 May 1991 to ESD/SR-1 and SAF/AQK.

2.5.2 Weekly Status Meetings (WSMs) and Bi-weekly Technical Interchange Meetings (BTIMs)

The WSMs documented the results of the work performed under SOW paragraph 3.1. Minutes were taken and produced for each of the WSMs and the BTIMs. These minutes were delivered to SAF/AQK as agreed to in the POA.

2.5.3 Preliminary Ada/C++ Comparative Analysis Report

This preliminary report documented work performed under SOW paragraphs 3.1.1 and 3.1.2. Copies were delivered as stated in the POA. The preliminary report was delivered on 21 May 1991 to ESD/SR-1 and SAF/AQK.

2.5.4 Final Ada/C++ Comparative Analysis Report

The Final Comparative Analysis Report documents the results of the work performed under SOW paragraphs 3.1.1 and 3.1.2. It is submitted in three hard copies to ESD/SR-1 and SAF/AQK on this date. CTA will respond to comments provided by the Government for updates to this report.

3.0 PRODUCTIVITY AND COST DATA

3.1 Overview

This section describes the results of acquiring the productivity and cost data for Ada and C++ projects. As anticipated, relatively few projects collect such data at the desired level of granularity. Where retained, the data definitions differ from project to project making precise comparisons difficult at best and invalid at worst. In terms of the SOW items a. through q., we were not successful in finding any project which could supply all of the items listed. Of the 80 some contacts made, only a fraction could offer any of the data desired. By far, only aggregate data is available with some observation of relative percentage of time spent in each of the classical waterfall phases of development activity.

The caveats are many, but that is not our purpose. The following sections present the data obtained first in summary form, and then in the raw detail. Section 3.3 summarizes the software productivity and cost databases encountered during the study. Section 3.4 offers commentary on the validity and precision of the detail data. The summary results are presented as a series of tables, each addressing some aspect of productivity and cost data. For these tables, simple steps have been taken to establish a consistent framework to aid in obtaining a profile of accumulated experience.

Section 3.3 describes performance and cost databases, as distinct from isolated projects. In many ways such data is of great value since care has been taken to ensure a consistent and detailed basis for acquiring and comparing such data. Both commercial and government databases have been identified; each with its own area of specialty and potential use. Comparisons among databases have difficulties similar to those encountered in making project to project comparisons.

3.2 Data Summaries

The following sets of tables summarize the productivity and cost data available. The first series of tables portray the better detail derived from both individual projects and collective detail derived from selected databases. The table in Section 3.2.2 summarizes the data from a dozen individual projects.

3.2.1 Summary Data

The set of tables offered below present data from individual project and selected databases. The topics presented were included where sufficient data was available to make more reasonable comparisons among several sources. Further, it was frequently possible to factor out variations due to factors such as reuse or variations in the life cycle phases. The lettered table identifiers correspond to the detailed data items of SOW Section 3.1.1, a-q. Note that only a few tables, i.e. topics, are offered. The sources are identified by number, as shown below: Note that the first four are individual projects, while the last three represent many projects.

1 AT&T
2 AT&T
3 STANFINS R
4 BOFORS
5 Martin Marietta, Denver
6 Reifer Consultants, Inc.
7 NASA/GSFC, Software Engineering Laboratory (SEL)

a. OVERALL EFFORT

This is the only representation of total size, cost , duration, and effort.

Source	$	Staff Months	Total Development Months	Size (KSLOC)	Language
1	$22 MEG	336 (5 Versions)	48	132	C++
2		95	10	125	C++
3	$24 MEG		24	2000	Ada
4	$22 MEG		38	1500	Ada
6	$7 MEG $6.5 MEG $5.5 MEG			100 (Ave.) 100 (Ave.) 100 (Ave.)	All Lang. Ada C++
7		126 72 45-54 60-108	18 18 15-18 15-18	29.2 16.3 12.7 15.2	Ada Ada Ada Ada

b. OVERALL PRODUCTIVITY

This table offers productivity rates for sizable projects in both languages. Most are from the business application domain. The "Developed SLOC/HR" column indicates that variations due to reuse or automatic code generation were factored out as far as possible.

Source	Size (KSLOC)	Application	Language	Developed SLOC/HR	Data Points
1	132	MIS	C++	1.2	1
2	125	MIS	C++	1.4	1
3	2000	MIS	Ada	1.5	1
4	1500	C2	Ada	1.5	1
5		MCCR	Ada	8	-
6	100 (Ave.)	MIS	Ada	1.31	153
		MIS	C++	1.17	23
		MIS	All Lang.	1.14	543
7	19 (Ave.)	Flight Dyanmics	Ada	1.02	5

d. DEFECT DATA

Relatively few projects or databases address the defect removal progression.

Source	Application	Language	Discovered through Integration (Errors/KSLOC)	Delivered (Errors/KSLOC)	Points
5	MCCR	Ada		3.5	
6	MIS	All Lang.	33	3	543
		Ada	24	1	153
		C++	31	3	23
7	Flight Dynamics	Ada		5.4	2

e. RATIO OF ESTIMATED TO ACTUAL COSTS

No data for C++ was available. Notably, reuse and code generation considerably reduced actual costs, while requirements growth and inaccurate estimates had detrimental effects.

Source	Language	Size (KSLOC)	Actual/Estimated Cost
	C++		No data.
3	Ada	2000	50% Decrease due to Auto Code Generation
4	Ada	1500	35% Decrease due to Extensive Reuse
5	Ada	-	60% Increase due to Requirements Growth 20% Increase due to Inacurate Estimate

g. LEVERAGED PRODUCTIVITY (REUSE/AUTO GENERATION)

Both Ada and C++ implementations can benefit from aspects of reuse. The examples cited show significant benefits. Interestingly, the data from the SEL suggests that leveraged productivity improves with experience with Ada. The same is probably true for C++ applications.

Source	Total Size (KSLOC)	Applications	Language	Reuse/Generation
1	132	MIS	C++	34% Direct, 19% Inheritance
2	125	MIS	C++	70% Reuse of Classes
3	2000	MIS	Ada	60% Auto Generated
4	1500	C2	Ada	65% Reused for Modifie
7	19.2 Ave.	Flight Dynamics	Ada	25% Initial, 35% Current, 33% Generic Packages

k. DEVELOPMENT EFFORT DISTRIBUTION

No distribution of effort data available for C++ projects reflects the distinct development environments for the two languages. C++ applications tend to be considerably less formally disciplined than DoD (Ada) applications. The shift of emphasis to the design phase should be comparable for both language implementations, since both offer packaging and design features not supported by other languages.

Source	Language	Size (KSLOC)	Req. Anal.	Design	Code	Test
	C++		(No Data)			
4	Ada	1500	(	83%	)	17%
6	Ada		(	50%	) 15%	35%
	FORTRAN		(	40%	) 20%	40%
7	Ada	19 (Ave.)	3-65%	27-36%	43-52%	13-18%

i. TOOL/TRAINING INVESTMENT

Reportedly, the transition from C to C++ is relatively smooth and straightforward. The examples cited indicate that the meaning of this statement is that many of the design features (inheritance, polymorphism) are often not used at all. In contrast, the move to Ada is deemed sufficiently different that significant training and pilot projects are employed to make the transition. While many Ada tools have emerged over the past decade, beyond the usual compiler /debugger level, support for C++ is just beginning to emerge.

Source	Language	Training Time	Product	Tool Investment
1	C++	Transition from C	OJT	None
2	C++	Transition from C	OJT (SABRE C++)	Commercial Tool
4	Ada	16 Months	Pilot Project (50 KSLOC)	Rational 1000 $690 K
5	Ada	(3 Wks. Min.)	Class Projects	Various (Rational, DEC)
7	Ada	6 Months Initial Project)	Electronic Mail System (6 KSLOC)	DEC Compilers
	Ada	4 Wks. (Subsequent Projects)	-	DEC Compilers

q. LEARNING/TRANSITION

The striking, if expected, trend here is the improved productivity that results from experience. Roughly the same extent of improvement was observed for both languages. It appears that this accrues largely from the design support aspects offered by both.

Source	Language	Productivity (SLOC/HR) Initial	3rd Project	Data Points
1, 2	C++	No Data		
5	Ada	.4	.8	-
6	C++	1.01	1.17	23
	Ada	.95	1.31	153
7	Ada	1.03	1.2	3

r. LANGUAGE POWER OF EXPRESSION

This item does not appear in the SOW, but is certainly pertinent to choice of implementation language. Power of expression has to do with the amount of work that can be accomplished with each language statement. While productivity is popularly measured with respect to source lines per hour, the real measure is amount of work accomplished per source line. Available evidence indicates that if the full power of C++ can be employed (packaging/inheritance/dynamic binding), then the power of language expression could be advantageous.

Language	Level	Source Statements Per Function Point
FORTRAN	3.0	105
COBOL	3.0	105
Ada	4.5	71
C++	11.0	29
4GL	16.0	20

3.2.2 Individual Project Data

The following table is organized to follow the topics identified in the SOW, Section 3.1.1. The rows of the table correspond to the paragraphs of Section 3.1.1, and each column represents the data available for each individual project. The information presented here is representative of the data encountered in making the contacts cited above. These examples demonstrate the lack of available detail. In an effort to level the data across projects, the following assumptions and conventions were adopted:

- One engineering month is equivalent to 160 engineering hours.
- Errors are counted as reported with no indication of type or seriousness
- Size estimates may be either estimates or counted
- Reuse estimates are taken as given with no attempt to define reuse
- The dollar value of any time amounts is not estimated

Evidently the detail available varied widely. No project had all of the desired detail, and often only commentary is offered where quantitative detail is desired. Without normalization, the indicated values likely do not represent the similar development environments and constraints. Collectively, this table should be viewed only as bounds upon the listed attributes.

3.3 Databases

3.3.1 Discussion

In the search for productivity and cost data, a modest number of related databases were encountered. This section provides an assessment of the value of these sources in a study such as this.

Most of the interesting databases are proprietary, but many are commercial in the sense that the data has been used to calibrate, validate, and substantiate software development cost and schedule prediction models. Use of the models, automated support tools, and default values and range of industry data values are being offered.

The number of distinct project "data points" in a productivity/cost database vary from tens of projects to thousands. In order to provide rational comparison of data collected from a variety of sources and circumstances, a normalization process typically is applied by the vendor/institution. Normalization attempts to factor-out differences among projects: complexity, real-time

Project	1	2	3
a. Effort	2,000,000 LOC	1,500,000 LOC	55,000 LOC
b. Productivity	3.7 LOC/Hr	3.28 LOC/Hr	4.30 LOC/Hr
c. Rework	-	-	-
d. Errors	-	-	-
e. Cost vs. Est.	0.50	-	0.35
f. Schedule vs. Est.	-	-	1.0
g. % Reuse	-	-	-
h. Size vs. Est.	-	-	306%
i. General Tools	-	-	-
j. Language Tools	-	-	$690,000
k. Direct Personnel Cost	-	-	-
Design	-	-	16 Eng-mo*
Code	-	-	5 Eng-mo
I & T	-	-	-
Implementation	-	-	-
l. Indirect Personnel Cost	-	-	-
m. Other Costs	-	-	-
n. Total Cost	$24,000,000	-	$760,000
o. Operational Benefits	-	-	reduced maint.
p. Lang/Total Costs	-	-	91/.49/-
q. Other	-	-	-

1. Ada, CSC: Finance Redesign, Rational case study information
2. Ada, BOFORS Electronics:FS2000 (1 ship of 5 only), Rational case study information
3. Ada, Philips Elektronikindustrier AB: UndC, 1988 Paris AFCEA Symposium

*Eng-mo = 1 engineer month

Project	4	5	6
a. Effort	5159 LOC	132,000 LOC	125,000LOC
b. Productivity	1.05 LOC/Hr.	2.4 LOC/Hr.	4.8 LOC/Hr.
c. Rework	-	-	-
d. Errors	12	-	-
e. Cost vs. Est.	-	-	-
f. Schedule vs. Est.	1.0	-	.8
g. % Reuse	-	53%	-
h. Size vs. Est.	87% (Turbo Pascal)	72% (C)	-
i. General Tools	-	-	-
j. Language Tools	-	-	-
k. Direct Personnel Cost	-	336 Eng-mo.	145 Eng-mo
Design	12.7 Eng-mo	-	-
Code	13.8 Eng-mo	-	-
I & T	2.6 Eng-mo	-	-
Implementation	-	-	-
l. Indirect Personnel Cost	-	-	-
m. Other Costs	-	-	-
n. Total Cost	-	-	-
o. Operational Benefits	-	Reduced maint.	Reduced maint.
p. Lang/Total Costs	-	-	-
q. Other	-	-	pre-processor

4. Martin Marietta Astronautics Group: Ada Initiative, 1990 International Society of Parametric Analysts Conference
5. C++, AT&T: CXR
6. C++, AT&T: BUFR

Project	7	8	9
a. Effort	94,000 LOC	9107 LOC	60,000 LOC
b. Productivity	3.15 LOC/Hr	-	-
c. Rework	6.5 hr.KLOC	-	-
d. Errors	10.4/KLOC	-	-
e. Cost vs. Est.	.94	.63	-
f. Schedule vs. Est.	-	-	-
g. % Reuse	-	24%	-
h. Size vs. Est.	-	-	-
i. General Tools	-	-	-
j. Language Tools	-	-	-
k. Direct Personnel Cost	-	-	-
Design	56 Eng-mo.	1.8 Eng-mo.	-
Code	117 Eng-mo.	1.1 Eng-mo.	-
I & T	13 Eng-mo.	0.9 Eng-mo.	-
Implementation	-	-	-
l. Indirect Personnel Cost	-	-	-
m. Other Costs	-	-	-
n. Total Cost	-	-	-
o. Operational Benefits	-	-	-
p. Lang/Total Costs	-	-	-
q. Other	-	-	-

7. Ada, (Reference not publicly available)
8. Ada, USAISSDC-W/CAO
9. Ada, Motorola: CID, Ada Strategies, July 1989

Project	10	11	12
a. Effort	200,000 LOC	100,000 LOC	98,000
b. Productivity	-	8.7 LOC/Hr.	4.8 LOC/Hr.
c. Rework	-	-	-
d. Errors	-	-	-
e. Cost vs. Est.	-	-	-
f. Schedule vs. Est.	.4	.1	-
g. % Reuse	-	-	-
h. Size vs. Est.	-	-	-
i. General Tools	-	-	-
j Language Tools	-	-	-
k. Direct Personnel Cost	-	-	-
Design	-	-	-
Code	-	-	-
I & T	-	-	-
Implementation	-	-	-
l. Indirect Personnel Cost	-	-	-
m. Other Costs	-	-	-
n. Total Cost	-	-	-
o. Operational Benefits	reduced maint.	-	-
p. Lang/Total Costs	-	-	-
q. Other	-	-	-

10. Ada, USMC: IRMC, AdaStrategies, May 1989
11. Ada, Genesis:
12. Ada/C/SQL/Window Maker/Data Repository, Wells Fargo: WFNIA, Ada Strategies, April 1991

implications, development environment, and the like. Application domain differences are so great that separate databases are kept by some institutions for each domain. These normalization processes are important to understanding the dominant cost and schedule drivers for the collected set of projects.

Very basic differences in the set of development activities are apparent among commercial and DoD projects. Some models identify a comprehensive set of activities that might be applied to any development project. Identification of whether or not each activity was actually employed is an important part of the normalization and understanding process. DoD projects tend to employ the majority of development activities, as dictated by standards such as DoD-STD-2167A. Commercial projects tend to skip or informally combine such activities. Evidently, each activity requires an associated effort and cost, and hence a potential difference in the basis of estimate. These differences must be accounted for in any attempt to make a valid comparison among project productivity and cost. Also critical is the underlying level of abstraction and the software development lifecycle assumed.

Some software development contractors have developed productivity and cost databases in order to understand and improve their own ability to predict development costs, and improve their ability to price contractual bids. Consequently, these databases are highly sensitive and generally unavailable in the public domain. In some instances, the underlying models can be made available sans the proprietary data.

Commercial organizations which have acquired extensive data have agreements with their sources guaranteeing anonymity. The advantage to the participating sources, of course, is gaining access to the collective range of experience.

Typically, the commercial sources offer sale or lease of their model, analysis tools, and appropriate training, in addition to access to their database. Most encourage tailoring model parameters to the specific experience of the contractor.

3.3.2 Assessment

The overwhelming advantage of using one of these commercially available databases, is the normalization process they have employed in placing all projects reviewed on the same basis. Apparently, for each database, statistically significant results can be reached and profitably employed.

Understanding each distinct normalization processes is critical. Any attempt to place all of these databases on the the same basis would require considerable effort, but might be worthwhile if a fine level of detail for a very large database was warranted. Even without detailed normalization, high level comparisons among databases can usefully be made with subjective interpretation of differences in the normalization processes.

One can conclude that each of these databases is probably more useful than an ad hoc collection of diverse, unnormalized, data points. Unexamined individual project datapoints can only be used for qualitative comparisons.

PRODUCT #1 - SOFTCOST-ADA

Source: Reifer Consultants, Inc.

Contact: Dr.Donald Reifer
25550 Hawthorne Blvd.
Suite 208
Torrance, CA 90505
213) 373-8728

Background:

Reifer Consultants, Inc. offers software management consulting, software management training, and results of research into metrics and cost models pertinent to software development and maintenance. Available support includes sizing tools (Function Points, SLOCs), cost/schedule estimating tools (Softcost-R, Softcost-Ada), and productivity and cost analysis tools to be applied to the database.

Database:

The RCI database contains some 543 projects which can be related to business systems. Of these, 153 are Ada projects and 23 are C++ projects. Care has been taken to ensure that all projects are considered on the same basis, i.e. normalized. Sensitivity analyses facilitate understanding of impact on overall costs in response to change in one or more of the cost drivers. By far, this source provided the best set of C++ productivity and cost data found. Tabular results pertinent to this study are offered elesewhere in this report.

Availability:

The database is available indirectly through lease of one or more of the RCI cost modeling tools. The RCI products are available from the contact given above. Ad hoc analyses using the RCI database can also be arranged with RCI

PRODUCT #2 - CHECKPOINT

Source: Software Productivity Research (SPR), Inc.

Contact: Mr.John Zimmerman/Mr.Capers Jones
SPR, Inc.
77 South Bedford Street
Burlington, MA 01803-5154
(617) 273-0140

Background:

SPR offers consulting services, applied software measurement tools, methodology training, and industry benchmark data in support of achieving software quality and productivity objectives. Checkpoint is a PC based tool which automates and combines the ability to measure, assess, and estimate all factors that influence software development. Some twenty-five software development activities, and up to 140 standard tasks, are identified which affect the cost and quality of software. Few projects employ every activity, but their inclusion or omission serve to normalize data from a variety of projects. The quality attribute is limited to defect detection and elimination

Database:

The database contains data from over 4000 projects, with 67% of the project data acquired since 1981. Over 50% of the projects describe system software developments, with an additional 25% MIS projects. SPR notes that more than 90% of US enterprises do not collect defect data at all, and therefore the quality data is relatively sparse. Most of the projects contain function point sizing as well as SLOC data. Some include "feature point" data in addition to function points which accomodates the high algorithmic content of non-MIS applications. Multiple implementation languages are represented, including Ada and C++. The relatively fine granularity of the normalized data provides an excellent basis for performing ad hoc studies.

Availability:

Checkpoint is a product available from SPR, Inc. at the above point of contact. As mentioned, consultation and training are also offered. An authorized ADP price schedule is available.

PRODUCT #3 - SOFTWARE ENGINEERING LABORATORY (SEL) DATABASE

Source: National Aeronautics and Space Administration/Goddard Space Flight Center

Contact: Mr. Frank McGarry
Systems Development Branch
Code 552
Goddard Space Flight Center
Greenbelt, MD 20771
(301) 286-6846

Background:

The SEL was established in 1976 with the goal of understanding software engineering technologies and practices when applied to the development of applications software. Identification and collection of suitable metrics is a critical objective. The activities, findings, and recommendations of the SEL are described in the Software Engineering Laboratory Series, a continuing series of recorded results.

Database:

The current version (April, 1991) of the SEL database contains 104 projects and 12 megabytes of data. The early projects were implemented mostly in Fortran, but the most recent projects describe Ada developments. One or two C++ projects have recently submitted their reporting forms to the database.

The database contains the following types of data for each of the projects:

- Resource Data-Time charges of developers and managers as well as computer time used.
- Error Data-Errors reported during development and testing.
- Product characteristics-Source size, number of components, component information, etc.
- Estimates history-Manager's periodic estimate of size, effort, schedules, etc.
- Growth history-Weekly history of the size of completed source code (SLOC).

- Change history-Weekly history of number of source code changes made.
- Project characteristics-Dates, sizes, staffing, reuse, etc.

The database is available as an exported Oracle file. Effective analysis would require importing to some database system with subsequent queries and reports to be generated.

The database is provided with all of the usual caveats regarding the need for understanding the environment, application domain characteristics, special project constraints, and the like.

Availability:

The SEL Data Base is available on magnetic tape at a modest cost from:

- Data & Analysis Center for Software (DACS)
- Rome Laboratories
- Griffiss Air Force Base
- Rome, NY 13441
- (315) 336-0937

PRODUCT #4 - PRODUCTIVITY DATABASE (PADS)

Source: Quantitative Software Management, Inc.

Contact: William Sweet
Quantitative Software Management, Inc.
2000 Corporation Ridge Suite 900
McLean VA 22102

Background:

Larry Putnam's organization offers several PC based tools for software cost and schedule estimation, project control, size planning, and a productivity database. The models are based upon the well known SLIM model (Raleigh curve) of software development life cycle introduced by Putnam. This model has several attractive features, including a mathematical basis for the widely observed cube root relationship between overall effort and schedule. A productivity index, based simply upon past projects duration, total effort, and size, can serve to predict and organization's capability to perform on new developments.

Database:

Much of the database supports the use of the high-level productivity index and the related peak staff build-up rates. Accordingly, the granularity of the data applies to the complete development life-cycle, although the models have successfully been applied to the maintenance phase as well.

Availability:

The models and supporting tools are offered commercially, as is the productivity database. Special services and specific analyses are quoted contractually on the basis of the effort required.

PRODUCT #5 - CEIS

Source: Computer Economics Inc.

Contact: Anita Gigliello
Computer Economics, Inc.
4560 Admiralty Way, Suite 109
Marina del Rey, CA 90292
(213) 827-7300

Background:

CEI offers two products related to software cost and schedule estimation. System-4 is similar in nature to the COCOMO cost estimation model, requiring twenty-six cost driver inputs or use of a default set. The model also accomodates variations in new, existing, deleted, modified, and purchased lines of code.
CEIS, the second product, provides an historical database of 40 reference projects for all types of software development. Eight of the reference projects were implemented in Ada; none were implemented in C++. The intended use of this database is to make pairwise comparisons between a future project and existing projects in a novel approach for making sizing estimates.

Database:

The current database consists of 40 projects characterized at rather a high level of abstraction. The available variables include attributes comparable to those in the COCOMO model. The system is designed to accomodate an expanding number of projects implemented in a variety of languages. Accordingly, the database is expected to grow with use.

Availability:

The CEIS product and associated database is available from CEI at the address listed above. As with most of the products, one leases or buys the tool; the database is a part of the product.

PRODUCT #6 - SOFTWARE ARCHITECTURE, SIZING AND ESTIMATING TOOL

Source: Martin Marietta Corporation

Contact: Dr.Aaron Silver/William Cheadle
Martin Marietta Corporation
P.O. Box 179
Denver, CO 80201
(303) 971-6730/40

Background:

Long term contracts with the Navy and Air Force has enabled this Martin Marietta group to build a sizeable software productivity and cost database. As with the other database sources, the main products are the parametric models used for software management, estimating, scheduling, calibration, risk assessment, maintenance, and sizing. The database supports the application and development of these models.

Database:

Over 500 projects, completed over the past decade and reflecting over 45 million source lines of code, provides a database representative of a broad range of DoD applications. Less than 20 percent of the data represents Martin Marietta projects. More recently, data from many Ada projects has been accumulated, but there is very little C++ data.

Availability:

The services of this Martin Marietta group are available via contract. The models and supporting database are not offered on a commercial basis.

3.4 Validity and Precision Considerations

The data, except for the RCI dataset and perhaps the other databases, is so sparse as to be inconclusive. Clearly, the major advantage of either language is the more precise design definition facilities present in both languages. Statistically significant results would require considerably more individual project information than has been identified. The current thrust of TQM/SEI Software Process Improvement likely will have considerable effect upon the measurement of the software development process. Carefully gathered data should have considerable impact upon the way software is ultimately developed in this country. Effective measurement is the key to understanding and improvement. As suggested earlier, comparison of data from two distinct normalized databases is suspect because of the widely varying bases for normalization.

3.5 Source Data

The available series of reports and pertinent data has been accumulated by others over the past few years. Some of the initiatives of this investigation has led to identification of additional information pertinent to this study. The summary data offered above provides useful perspective, however the raw data itself may be of continuing interest and is appended to this report.

4.0 ADA/C++ COMPARATIVE ANALYSIS

4.1 Overview

This section offers an analysis and interpretation of the productivity and cost data collected during this study. While quotable data sources were relatively sparse, especially for C++ projects, bounds on the differences between C++ and Ada developments came into focus. A major difference in the basic cost/productivity arena is the more formal development environment of DoD projects using Ada, and the commercial projects using C++. The additional activities invoked on DoD projects add to basic costs and lower overall, productivity indices. The differences in levels of standardization and maturity will have continuing impact on the portability and maintenance of C++ implementations. Finally, the reported advantages of using either language is the benefits deriving from its use in the design process as well as the implementation process. The fundamental application objects and their interrelationships are evident at the design phase. It is these factors, which aid understanding, that are so important to maintenance and continuing evolution of the system. The following paragraphs identify and discuss some of the major issues contrasting these two languages.

4.2 Major Issues

4.2.1 Rate of Source Line Production

The average number of hours required to produce an average single programming language statement is the conventional measure of software productivity. Imperfect for many reasons, it remains widely accepted because it is (ultimately) easy to obtain and can be used to predict the effort required to implement similar products. Existing software effort and schedule models modify the resulting estimates by factors which attempt to account for differences in complexity, development environment, staff experience, and the like. Since these models include the effects of all activities performed throughout development, it is clear that an average rate will depend heavily upon the nature and number of activities performed as well as the basic capability of the staff itself. Standardization of the process, such as compliance with DoD-STD-2167A documentation requirements, tends to validate project by project comparisons. Differences in the process, such as inclusion of formal design inspections, tend to invalidate comparisons. A consistently applied, well defined development process (at a given maturity level) offers the promise of better precision in the nominal rate of production for a particular organization.

Other important influences on comparative source line production rates include:

- Definition of a source line itself. There are several different, but widely used definitions. It makes considerable difference in project comparisons.
- Definition of the life cycle phases included, such as accommodation of system requirements definition.
- Inclusion of development support services such as software quality assurance, configuration management, data management, and the like.
- Formality of the process: number of change control boards, frequency and scope of reviews, distinct number of test beds, etc.

Given all of these caveats, Table b. offers representative source line production rates observed for Ada and C++. Data from the same source for both languages may offer some assurance that a consistent basis was employed. The apparent consistency of the rates shown should be viewed with skepticism for the reasons cited above.

4.2.2 Language Level

The number of source lines written by the programmer cannot be directly related to the achieved functionality or work performed by the software. A major influence is the language's power of expression, or ability to deal with higher level abstractions. One extreme is the use of so-called fourth generation languages (4GLs), a proven method of leveraging productivity by describing only what is desired rather than the step by step process required to obtain it. Other examples include the famous "one-liners" (and obscure) programs in the APL language, where just a few characters of "code" could elicit very complex computations and results.

Both Ada and C++ support object oriented design methods (although many say that neither is completely satisfactory). Object oriented design improves productivity by raising the level of abstraction in dealing with software components through packaging mechanisms. These facilities also offer the promise of software component reuse rather than repeated reconstruction of needed functionality via a new sequence of source lines of code. Reuse has an enormous effect on apparent productivity.

A striking, if atypical, example of power of expression is Booch's reported reimplementation of his primitive data structure components in C++. The new version, which employed the full

power of C++ constructs, required only one-fifth the source lines required by the original Ada implementation.

Software Productivity Research, Inc. has ordered the expressive power of many programming languages by examining the average number of assembler instructions required to perform a single statement in that language. They have also related the number of source language statements required to implement an average "function point", a measure of externally observable work particularly descriptive of MIS applications. Table r. presents these relative values for Fortran, Ada, COBOL, C++, and a 4GL.

4.2.3 Reusable Components

The extent of exploiting reusable components would seem to depend upon the existence of readily available component sets. Examples of these sets are available for both Ada and C++. The May/June issue of the ACM's Ada Letters cites 22 potential sources of Ada components. The C++ literature contains a growing body of Class Library advertisements. Experience in exploiting these components seems to be mixed, as yet. On the other hand, several projects boast significant benefits from reuse of project specific components. This experience has been demonstrated for both Ada and C++ projects of significant size. Table g. provides statistics for a few notable examples. There are several remarkable examples showing high productivity leverage via reuse or code generation for both Ada and C++.

4.2.4 Maintenance Phenomena

Effective maintenance depends largely upon the size, complexity, and understanding of the software. All three aspects are simplified when we can deal more abstractly with segments of code larger than individual source lines. Thus, manipulating packages, rather than code, in either Ada or C++, should improve the maintenance process. Again, product enhancement should offer opportunities for reuse of existing components in the system. Several examples of the effectiveness of abstraction in the maintenance phase deal with experience of not having to affect package interfaces.

On the other hand, full use of the expressive power of C++ is required in order to benefit from the inheritance feature. In a comparative study of C++ and Ada maintenance, Wild (CADRE) found that making even minor changes in a class frequently had unwanted and difficult to find inherited effects in other areas of the software.

4.2.5 Maturity

Maturity of the language impacts many qualitative aspects of language choice, and has significant impact on cost and quality of software products.

C++ is relatively new, and was not a language designed to meet a fixed set of requirements. Consequently, additional "desirable" features are regularly brought forth for consideration in the embryonic standardization process. Most of these refinements relate to the inheritance facilities and neglect other features such as task synchronization among independent processes. Version 3.0 will follow quickly on the heels of the recently released version 2.0. Continuing change creates confusion among practitioners, and does little to establish a stable body knowledge, experience, and reusable components in the application of the language. Further there is no acceptance test suite to ensure that all compilers will produce consistent and portable code, although the original version maintained consistency by simply distributing the identical C preprocessor.

Changes in the language make life more difficult for the CASE vendor, since his product must track the language features. It appears that The Saber C++ product is becoming the de facto standard environment.

Finally, cultural lag and the so-called "third project" phenomenon shows that any organization requires significant time in order to assimilate a new language and development process. Data from GSFC's SEL demonstrates the shift in use of Ada language features as experience and understanding are gained. RCI's Ada project data shows that productivity increases over the span of performing on three projects while converging to a stable value beyond that. Similar experience can be expected with respect to introduction of C++.

5.0 CONCLUSIONS

As expected, few organizations gather productivity, cost, and quality data with the granularity defined by the SOW. At best, some measure of overall cost and productivity was available. At worst, only anecdotal information could be obtained. While valuable in conveying and understanding of their development experience, the lack of quantitative data was not helpful to the stated objectives of this effort.

An important common distinction between C++ and Ada development projects was the formality of the development environment itself. Typically, the Ada developments were in accordance with military standards; and incorporated formal reviews, additional documentation, and additional engineering support activities such as formal QA and CM. Most of the C++ projects are commercial, and do not incorporate such activities as extensively. Additionally, the C++ developers are usually intimately involved with the users, resulting in considerably less requirements engineering effort. Consequently, the C++ developments are inherently less costly and the productivity rates favorably skewed toward C++.

Based upon the stated and observed experiences, much of the benefit of using C++ and Ada is the ability to design at a higher level of abstraction; i.e., using packaging concepts and an object-oriented approach. The choice of implementation language appears to be not as important as the improved design approach. Such an approach generally leads to high leverage productivity improvements through techniques such as automatic code generation, inheritance of properties of existing objects, and more extensive reuse of previously developed encapsulated objects.

Relative maturity of Ada and C++ has many important implications. It may be years before a stable C++ language specification can be established. New "desirable" features are continually being considered for the latest standard release. Vendors are likely to offer their own "enhanced" version of C++ compilers and CASE tools, thus complicating the portability and general reuse issue.

The C++ language is new. Although it can be used immediately by C programmers developing C code, using C++ for design (inheritance, polymorphism) apparently requires a learning investment and methodology development similar to learning software engineering with Ada (packages, tasks, and generics).

The maintenance issue is clouded. The proper use of either language facilitates understanding of large systems because of the abstraction or packaging concept. Interfaces among objects tend to remain fixed, even though detailed internal implementation may be altered. Similarly, possibilities of component reuse are enhanced for both. However,there is evidence that the C++ inheritance property, although a powerful construction tool, complicates the maintenance process by obscuring the propagation effects of a local change in an object. Spaghetti classes develop as easily as spaghetti code. This may mean simply that better tools are required to cope with "engineering" inheritance.

Class Libraries for C++ are being offered commercially for reuse, similar to the component libraries for Ada. For the most part, project unique reusable components appear to be the more valuable. Examples of highly leveraged reused project components exist for both languages. The C++ language provides easy access to extensive existing system software implemented in C. Such systems include graphics support, communication protocols, operating systems, and database management. Effective Ada bindings to such systems are emerging, perhaps making this a moot point for business applications as well as MCCR systems.

Finally, all of the original arguments for establishing a single programming language for military applications remain. Common training, tools, understanding, and standards simplify the acquisition, support, and maintenance problems. After maturing for a decade, the benefits of Ada have been proven for all classes of application. There is no demonstrable overwhelming advantage of one language over the other. Why introduce a new complication into the community?

APPENDIX D. Final Report (TRW Substudy)

This appendix contains the following reports from the TRW substudy: a lifecycle cost analysis of Ada and C++ and a case study example.

TRW Inc.
Space &
Defense Sector

One Space Park
Redondo Beach, CA 90278

Title

Ada and C++
A Lifecycle Cost Analysis

DATE June 1, 1991 Revision: 1

CDRL No. 002

Contract No. MDA970-89-C-0019

PREPARED FOR
SAF/AQKS

TABLE OF CONTENTS

1. Introduction

Ada is a stable, mature language supporting modern software engineering approaches. Compilers, tools, and integrated programming support environments for Ada are available covering a wide range of computer architectures. Ada has been used successfully to develop, deliver, and maintain multi-million-line-of-code systems ranging from MIS to embedded real-time applications.

C++ is relatively new language that supports object-oriented programming and modern software engineering approaches. Although its definition has not yet stabilized, compilers and tools are now available for many computer architectures. C++ has been used to successfully develop and maintain multi-hundred-thousand line systems, and is currently being used to develop systems larger than a million lines.

The purpose of this study is to take an objective, independent look at the current capabilities and maturity levels of both languages from the perspective of lifecycle costs.

> For a typical DoD management information system application (i.e., large to very large, highly reliable, 10+ year lifecycle, maintained by government personnel) which language would achieve the necessary mix of software quality (primarily function, performance, reliability and maintainability) for the lowest lifecycle cost with the highest probability of success (i.e., minimum risk).

The intent of this study is to make an engineering and business assessment of the two programming languages, identifying their strengths and weaknesses, and recommending a set of actions to maximize the DoD's future cost effectiveness and return on software technology investment.

1.1 Approach

Our approach for this tradeoff analysis (Figure 1) was to define a set of maximally independent criteria, judge each language with respect to those criteria and then translate those judgements into cost impacts in order to emphasize the importance of each criteria from a lifecycle cost perspective. In performing the tradeoff, we have included both the MIS and C^3 domain as well as Ada, C++ and C languages. The C^{378} domain and C language, while not primary subjects

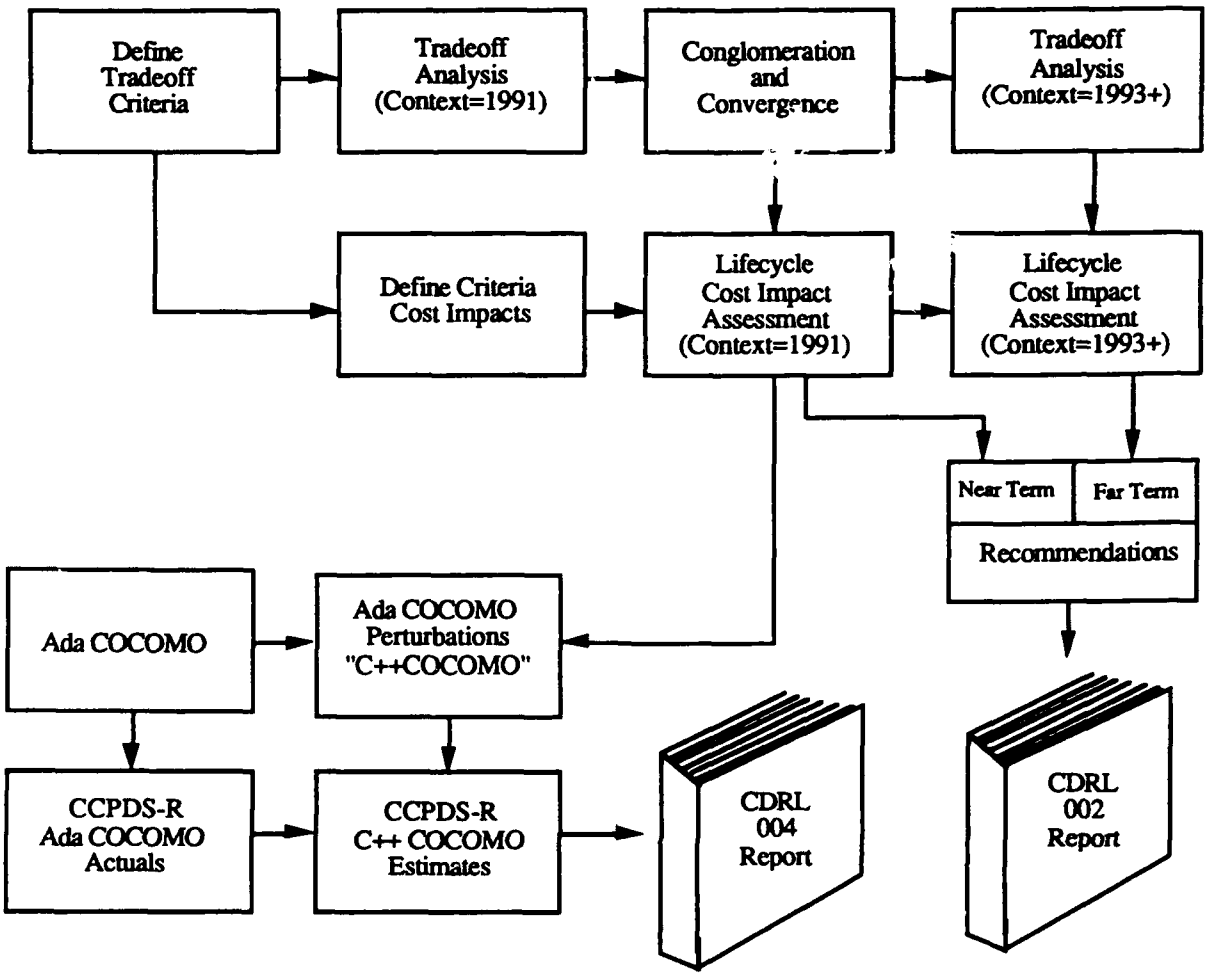

Figure 1: Tradeoff Analysis Approach

of the tradeoff, provide: 1) a large relevant experience base, and 2) a "control" perspective for objective analysis. The results of this analysis were translated into perturbations to the Ada COCOMO cost model so that a real world specific project application could be compared.

The following tasks pertain to the production of CDRL 002.

Define Tradeoff Criteria Identify the criteria for which each language will be evaluated. These criteria should consider all aspects of technical, pragmatic and human perspectives impacted by programming language choice. These criteria should be as independent of one another as possible and cover all significant aspects of cost impact.

Tradeoff Analysis, 1991 The tradeoff matrices will be evaluated by software engineers and computer scientists who are familiar with both languages, large DoD software development, and modern computer science in theory and practice. The tradeoffs will include an assessment of *importance* of each criteria to the MIS domain (with the C^3 domain evaluated also for comparison) and the *ranking* of C++ and Ada for each criterion (with the C language evaluated also for comparison). These tradeoffs will be done in the context of 1991 technology.

Tradeoff Analysis, 1993+ The tradeoffs will be re-evaluated based on speculation of technology maturity in the post 1993 timeframe.

Conglomeration and Convergence The rankings will be combined and averaged into a group ranking which represents the consensus of evaluators. Criteria which have large standard deviations amongst the evaluators will be reassessed with explanations for the high and low rankings in order to reconcile the divergence.

Define Cost Impact Each criterion will be traced to the Ada COCOMO cost drivers to reflect its relative importance in affecting lifecycle cost. *Weights* will be determined for each criterion based on the range of cost impacts (i.e., the sensitivity of lifecycle cost to the tradeoff critera). These weights will serve to emphasize the more important cost influences.

Lifecycle Cost Impact Assessment, 1991 A lifecycle cost assessment for all three languages (C, C++, and Ada) in each application domain (MIS and C^3) will be determined by taking the sum of *(importance * ranking * weight)* for each criterion.

Lifecycle Cost Impact Assessment, 1993+ The lifecycle cost impact assessment will be made for the 1993+ rankings to provide an extrapolation to the future.

Recommendations Finally, the various tradeoffs and analyses above will be summarized in a set of both near term and far term recommendations to the DoD with respect to their current commitment to Ada, future commitment to Ada, and the ramifications of current and future use of C++.

1.2 References

[1] Boehm, B. W., "Software Engineering Economics", Prentice Hall, 1991.

[2] Boehm, B. W., Royce, W. E., "TRW IOC Ada COCOMO: Definition and Refinements", *Proceedings of the 4th COCOMO Users Group*, Pittsburgh, November 1988.

[3] Booch, G., "Object Oriented Design with Applications", Benjamin/Cummings Publishing Company, Inc., 1990.

[4] Royce, W. E., "TRW's Ada Process Model for incremental Development of Large Software Systems", *Proceedings of the 12th International Conference on Software Engineering*, Nice, France, March 26-30, 1990.

[5] ANSI/Mil-Std-1815A-1983, Reference Manual For The Ada Programming Language, United States Department of Defense, American National Standards Institute Inc., 1983.

[6] UNIX System V AT&T C++ Language System, Release 2.0 Product Reference Manual.

[7] Ada 9X Project Report, Draft Ada 9X requirements Document, August 1990, Office of the Under Secretary of Defense for Acquisition, Washington, DC.

[8] Wild, F., "A Comparison of Experience With The Maintenance of Object-Oriented Systems: Ada vs. C++", *Tri-Ada '90 Proceedings*, Baltimore, December 1990.

[9] Humphrey, W. S., Sweet, W. L., CMU/SEI-87-TR-23, ESD/TR-87-186, "A Method for Assessing the Software Engineering Capability of Contractors", Software Engineering Institute, September 1987.

[10] Pohl, I., "C++ for C Programmers", Benjamin/Cummings Publishing Company, Inc., 1989.

2. Background

This section discusses various topics which are prerequisites to interpreting and understanding the tradeoff analysis.

2.1 Ada Summary

Ada is a stable, mature language supporting modern software engineering approaches. Compilers, tools, and integrated programming support environments for Ada are available covering a wide range of computer architectures. Ada has been used successfully to develop, deliver, and maintain multi-million-line-of-code systems ranging from MIS to embedded real-time applications.

2.1.1 Ada History

By the early 1970's, the trend of rising software costs, late deliveries, unreliable software and cost overruns led the DoD to begin the search for a more efficient and reliable means for developing software. In January 1975, the Director of Defense Research and Engineering formed a joint-service High Order Language Working Group consisting of DoD agencies and a liaison from the UK, West Germany and France. The charter was to identify requirements, evaluate existing languages, and recommend the adoption or implementation of a minimal set of DoD high-order languages.

Over the next few years, several generations of language requirements documents were developed and reviewed. The result was the 1977 Ironman requirements document. Two independent economic analyses indicated that substantial savings could be obtained by the adoption of a common high-order language.

In 1977 an RFP was issued internationally to solicit language designs based on the Ironman requirements. Of the 17 responses, DARPA selected 4 contractors to continue a 6 month development period. In 1978, the designs were stripped of identifying markings and evaluated worldwide by 400 volunteers participating in 80 review teams. In 1978, two of the designs were selected for further refinement. The following year, language designs were completed and reviewed. By May 1979, the winner was selected and Ada became the official name of the DoD's common high-order language.

Ada was released in July 1980. The first ACM Symposium on Ada was held in December 12, 1980 and on the same day MIL-STD 1815 was established as the approved DoD standard for Ada. The Ada language reference manual was approved as an ANSI standard in February 1983.

Once the language definition stabilized, tools vendors began developing compilers, tools, and eventually integrated programming support environments for Ada. Today these tools and environments are fairly mature and are available for a wide range of computer architectures. Standardization of Ada promotes portability of Ada applications and allows the development of integrated and compatible tools. However, because of the smaller size of the Ada market relative to languages such as COBOL and C, commercial investments in integrating Ada with COTS products are limited.

Ada has satisfied the goals of the DoD in being a highly reliable and maintainable general purpose language. It has been used in a diversity of application domains, such as real- time, MIS, and mission-critical systems. A substantial number of very large applications, greater than 1 million SLOC, have been successfully delivered and maintained. The DoD backing of Ada has lead to adoption of Ada by international defense communities. Ada is now being adopted by the United States government for large-scale applications development due to its success in implementing complex defense applications.

2.1.2 Ada Overview

Ada, as defined by ANSI/MIL-STD-1815A(1983), supports the creation of user-defined types, where implementation is separate from specification. These types enable the encapsulation of the type's data structure with the subprograms defined for the type. Packages can be parameterized by types, subprograms, or objects to provide very general data structures and programs, and a high level of code reuse. Subtyping permits construction of types which restrict the value range of their super types. Derivation declares new types which use the data structure and subprograms of another type.

Strong typing prevents type mismatches and is achieved at compilation time by static type checking. Unchecked type conversion can be achieved by explicit use of the Unchecked_Conversion function. Safe use of types is enhanced by run-time checking of array boundaries and type ranges, and by providing a predefined Boolean type.

The structure and management of large programs is simplified by well-defined compilation units and compilation dependencies. Packages can be used to group related types and subprograms. The specification of a package provides a separate compilation unit from the body or implementation. This isolates units that use the package from implementation details and changes. Ada exploits this separation with well-defined rules for determining the compilation order of units that minimize the extent of recompilation due to changes.

Handling of errors and exceptional conditions during program execution is supported by the use of exceptions. This standard mechanism works with user-defined and predefined exceptions.

Concurrent execution is provided by tasks. Tasks can execute both asynchronously and synchronously. Tasks may be partitioned across multiple processors or they may share a single processor. Scheduling of concurrent tasks on a single processor can be determined by assigning each task a priority.

Ada provides a variety of mechanisms for interfacing with the external world. Standard packages are defined for input and output and support both text and binary data. Program interrupts are handled by tasks. The predefined *pragma interface* provides a mechanism to interface Ada with other programming languages.

2.1.3 Ada 9X Summary

The revision of ANSI/MIL-STD-1815A, the Ada Programming Language, is being conducted by the Ada 9X project office located at the Air Force Armament Laboratory at Eglin AFB, Florida. The Ada 9X project is sponsored by the Ada Joint Program Office in the Office of the Under Secretary of Defense for Acquisition. The project was initiated in October 1988 with participation by numerous government, contractor, and international language experts. The Ada 9X Project manager, Ms. Christine Anderson, provided the following list of Ada 9X enhancements to Ada 83 which should benefit MIS applications:

Decimal Arithmetic Explicit support for objects of decimal-scaled types (e.g., dollars and cents) will be provided along with relevant arithmetic and I/O support.

Expanded Character Sets Explicit support for international character sets (8 bit or possibly 16 bit) will simplify many MIS applications which are constrained by the limitation of Ada's current 7 bit ASCII standard. Furthermore, explicit support for IBM's EBCDIC character set will permit more natural interfaces to the large base of existing MIS applications.

Improvements to I/O Portable append-mode output, and added control over record size, space allocation and other file format information ("Form String") will permit Ada programs to more elegantly access existing binary data files without bulk format conversions.

Multilingual Interfaces Ada 83 includes a one way mechanism ("pragma interface") for calling foreign language programs. Ada 9X will provide a bidirectional mechanism (i.e., Ada programs can call and be called) so that it will be easier to integrate multilingual applications. This support will ease the complexity of fitting Ada into existing programs and fitting existing programs into Ada.

Expanded Representation Specifications Expanded record layout control and unsigned integer support will make it easier to integrate Ada types with existing binary data formats from COTS DBMS's or other non-Ada objects.

Object Oriented Programming Single inheritance and polymorphism of Ada derived types will be added in Ada 9X to expand Ada's support for object oriented design.

Miscellaneous Software Engineering Improvements There are many other language simplifications, relaxed constraints and minor language improvements which reflect lessons learned in numerous years of Ada usage. These improvements should improve software engineering with Ada, understandability and ease compiler burden.

2.2 C++ Summary

C++ is a relatively new language that supports object-oriented programming and modern software engineering approaches. Although its definition has not yet stabilized, compilers and tools are now available for many computer architectures. C++ has been used to successfully develop and maintain multi-hundred-thousand line systems, and is currently being used to develop systems larger than a million lines. These applications include MIS and embedded systems. They are being developed primarily by commercial companies with existing C/UNIX applications. C++ provides

language features that make it particularly useful for developing applications such as graphics user interfaces and database interfaces. Since C++ is an extension to C, it is relatively easy to write C++ software that reuses existing C software or that uses COTS software packages with C interfaces.

The primary design goals of C++ are to provide object oriented programming capabilities (encapsulation, abstraction, polymorphism, and inheritance) "...without compromising the advantages of C." Such advantages include:

(a) runtime performance
(b) memory utilization
(c) compilation times
(d) development in existing (traditional C) environments.

As C++ evolves, however, such advantages may be subject to compromise. Desirable features, such as parameterized types and exception handling, are being considered for inclusion in future versions of C++. Hence some of the original goals may change over time.

2.2.1 C++ History

The C++ programming language was developed at AT&T Bell Laboratories in the early 1980's by Bjarne Stroustrup. The development of C++ was motivated by a desire to make the design of good programs easier. C was chosen as the base language for C++ because it is versatile, highly portable, low-level enough to be useful as a systems programming language and fits into the UNIX programming environment.

In 1980, function argument type checking and conversion, and classes were added to C; the resulting language was called "C with Classes". During the early 80's, C with Classes was redesigned and extended. The major additions were the ability to redefine functions in derived classes, and the ability to overload the basic operators of the language. The new language, C++, was released by AT&T in late 1985. Within six months there were commercial ports of C++ available on over 24 systems, ranging from PCs to large mainframes. By 1988 the first native compilers had been produced for the PC and workstation markets.

Version 2.0 of the C++ language was released in 1990. Additions were made to C++ to improve support for large-scale library building. These include multiple inheritance, abstract classes, and linkage to other languages.

"The Annotated C++ Reference Manual" forms the base for ANSI standardization of C++ and specifies version 2.1 of the AT\&T CFRONT translator. Until standardization is complete, the AT&T CFRONT translator serves as a de facto standard. Most C++ compiler vendors support CFRONT and update their compilers in accordance with new versions of it.

AT&T's commitment to C++ has led to quick acceptance of the language in the commercial sector. C++ is being used to implement CAD/CAM, object-oriented database and graphics applications, and to re-engineer a significant number of C programs. A small number of very large systems, greater than 1 million SLOC, have been successfully undertaken, however these applications are not mature enough to demonstrate the maintainability of C++ code. The significant market share of C and Unix represents an important C++ advantage in selecting an object-oriented language in the 90's.

2.2.2 C++ Overview

C++ is a superset of the C programming language and extends C by supporting data abstraction, abstract classes, inheritance, polymorphism and strong typing.

The key concept in C++ is the notion of classes. Classes allow the user to define new types which can be used in the same way as the language's predefined types. Classes separate the implementation of the type from the specification, forcing objects to be used through a specific interface and isolating users of the type from change. Inheritance provides a mechanism for deriving new classes by adding features to an existing base class. An inheritance hierarchy can be constructed, where derived classes inherit features from a base class. A base class can be used as a common interface to its derived classes. Inheritance promotes code reuse by allowing derived classes to use code defined in base classes.

Polymorphism is closely related to the notion of inheritance. It allows objects of a derived class to be used in any context where an object of its base class is expected. Virtual functions allow functions first defined in a base class to be redefined in derived classes. Dynamic binding ensures that the correct version of a function is called at runtime, when an object of a derived class is being used in a function which expects the base class.

Polymorphism and inheritance combine to provide reusability and maintainability. Inheritance provides code reuse and allows existing modules to be specialized for new applications. Polymorphism ensures that code that uses one class can still work on derived classes introduced later in the lifecycle of the system. New system requirements and changes to fix errors can be handled effectively by using these features.

Strong typing ensures that values of different types cannot be mixed and allows type errors to be detected at compile time. Explicit type casting provides a mechanism for bypassing the constraints enforced by static type checking.

Classes provide a mechanism for decomposing large systems into modules. Classes can be tested separately and bound through well defined interfaces. C++ has a powerful preprocessor which can be used to define conditional compilation of code.

C++ has a variety of mechanisms for interfacing with the external world. C runtime libraries can be invoked from C++ programs, and all implementations of C++ define linkage to C. This feature allows C++ extensions to be written to existing C applications easily. Linkage to other languages may be defined but is not supported directly by the language. C++ also provides a standard I/O library.

Despite the relative immaturity of the language, acceptance of C++ in the C/UNIX community has ensured that a large number of vendors are producing, or plan to produce, products which integrate directly with C++. Examples include object-oriented databases, CASE tools, and C++ libraries which support graphic user interfaces. It is probable that the availability of tools and products for C++ will explode in the 90's, as happened in the 80's for UNIX and, to a lesser degree, Ada.

2.2.3 C++ Version 3.0 Summary

The ANSI base document defines two new features for the C++ language, a template mechanism supporting parameterized types, and exception handling. The addition of these features to the C++ language will increase its support for maintainability, reusability and reliability. Templates are expected in the 3.0 Version of C++, and exceptions are expected in a later 3.n release of C++.

Templates and Parameterized Types Parameterized types allow the construction of extensible, maintainable, and reusable software components. A single module with a uniform interface can be defined and used to construct similar modules for any number of different types. The classic example is a general list data structure which can be used to create lists for integers, strings, and any user-defined types. The list is implemented once, and then instantiated with different types to construct the new list data structures. The code is reused in each instantiation. A variety of software components can be constructed in this way and used to build reusable software libraries. Code is high quality and less error prone as each component is implemented only once.

Templates in C++ are analogous to generics in Ada. They provide parameterized types. There are minor differences in syntax but the two mechanisms are inherently the same. In C++ templates can be combined with inheritance, thus enhancing the power of the language. Both features promote reusability, extensibility, and maintainability, and therefore allow powerful software libraries to be constructed.

Exception Handling Exception handling provides a mechanism to deal with runtime errors in an orderly fashion. When a runtime error occurs, an exception is raised. Once an exception is raised in a routine, control is passed back to the code that invoked the routine. The invoking code can then take appropriate action to deal with the exception, or, depending on the severity of the error, can choose to perform any necessary housekeeping and exit the program. Exception handling allows the construction of reliable, fault-tolerant systems.

C++ will use the same model for exception handling as Ada. However, C++ will not have any predefined exceptions like Ada has for runtime errors such as array bounds checking and arithmetic overflow and underflow. C++ will allow information about the error that caused the exception to be passed to the invoking routine, thus allowing more powerful exception handling than is currently possible for Ada.

2.3 C++/Ada Similarities and Contrasts

Ada is a mature language which has met its development goal of providing a language for large-scale development. It has demonstrated maintainability and reliability. Ada is safer but less flexible than C++. The additions and refinements provided in version 9X will evolve the Ada language and make it more useful for developing certain classes of applications, including graphic user interfaces.

C++ is a newer, evolving language, that is already the most widely used object-oriented language in the commercial arena. Since it has a C and Unix base, there will be quick penetration of C++ in commercial applications. C++ provides powerful object- oriented programming features that make it particularly suitable to applications such as graphic user interfaces. C++ is highly flexible and therefore less safe than Ada. The emerging C++ standards will help to increase the portability and maintainability of C++.

The comparison of the similarities and contrasts between Ada and C++ will be based on current versions of the languages. For Ada, the language is defined by ANSI/MIL-STD- 1815A-1983 For C++, the language is defined by version 2.1 of the AT\&T CFRONT translator which is a subset of the language described by the Annotated C++ Reference Manual (ARM) by Ellis \& Stroustrup. The ARM version of the language is not supported yet by commercially available compilers.

Table I identifies some important language features and their relative support in the two languages. The common features have been discussed previously, the following paragraphs discusses the features where one language has an advantage and the features that are only available in one of the languages.

Feature	Ada Only	Ada +	Both =	C++ +	C++ Only
Parameterized Types	X				
Safe Types	X				
Error Handling	X				
Concurrency	X				
External Interrupts	X				
Compilation Management		X			
Strong Typing			X		
Modularity			X		
External I/O			X		
Extensible Typing			X		
Overloading			X		
Multilingual Support				X	
Polymorphism					X
Inheritance					X
Subprogram Variables					X
Conditional Compilation					X

Table I: Ada and C++ Support for Key Language Features

2.3.1 Features Where Ada Has An Advantage

Parameterized Types A parameterized type mechanism is useful for building strongly typed reusable component libraries. Ada provides this support through generics. This feature is not available commercially in C++. Templates have been accepted into the ANSI working documents for C++. Some users of the present version of the C++ language have built their own template preprocessors, the remaining users provide macros.

Safe Types Ada provides run time checks for array subscript variables and ranges. The C++ array mechanism does not provide bounds checking. C++ provides flexible dynamic memory allocation which must be used carefully to prevent problems.

Error Handling A reliable standard mechanism for handling errors is essential for building reliable and maintainable systems. Error handling is provided in Ada with user defined exceptions and five predefined exceptions.

Concurrency Multi-threaded software is essential for the efficient implementation of large systems. Ada provides support for concurrency with tasks.

External Interrupts Many programs depend on the ability to reliably receive and handle interrupts from the external environment. Ada provides a standard mechanism for handling interrupts as task entry points.

Compilation Management Efficient management of compilation dependencies within very large software systems can save large amounts of computer and human resources. Good compilation dependency information also simplifies the creation of software tools that rely on analyzing code structures. These include configuration management tools, test generators and code analysis tools. Compilation dependencies are well defined in Ada. The definition of compilation management is not as well defined in C++. For example, the C++ ARM does not specify where a translator will look for the implementation of template functions.

2.3.2 Features Where C++ has an Advantage

Interface to Other Languages Interfacing well with other languages is an important attribute of any programming language. Many large systems use more than one programming language to take advantage of existing or COTS software. C runtime libraries can be invoked directly from C++ programs and all implementations of C++ define linkage to C. This provides the capability to use existing C software and COTS software with C interfaces when developing C++ programs. Ada defines an optional pragma interface for interfacing to other languages. Most compilers provide little support for this pragma.

Inheritance Inheritance is an extension to the type system that provides the object-oriented designer with the ability to accurately model sub-type hierarchies in the application domain. C++ supports both single and multiple inheritance. This feature is not available in Ada. The C++ inheritance facility is more powerful than the derived type mechanism in Ada. Single inheritance is expected to appear in the Ada 9X language.

Polymorphism Languages Languages that support polymorphism can directly model different kinds of objects using a common interface. Polymorphism is used extensively in graphic user interfaces. C++ supports polymorphism through its inheritance mechanism. Inheritance and polymorphism is expected in Ada 9X; however, it will only be single inheritance which is less powerful than the multiple inheritance which is supported in C++.

Subprogram Variables C++ has pointers to functions and pointers to class members. Pointers to subprograms are expected in the Ada 9X language.

Conditional Compilation C++ supports conditional compilation, via the preprocessor mechanism. TBD why this is good.

2.4 C^3/MIS Similarities and Contrasts

A C^3 system is a type of data processing system that is used to support strategic or tactical decision makers in the conduct of real time operations. A Management Information System (MIS) is a type of data processing that is used to the maintain financial and administrative information resources of an enterprise.

The development methods for these types of systems have traditionally been treated as separate design domains, with each domain having its own approach to requirements analysis, system design, and software development. Recent advances in information processing technology have dramatically affected both domains. These advances include the availability of low cost processing power and the emergence of industry standards in such diverse areas as operating system, communications, programming languages, databases, and graphical user interfaces.

New technology applicable to both domains has brought the two domains closer together in terms of the design tools and techniques used in each domain. Moreover, the rapid rate of change in the hardware environment has created more demand for portable, non-proprietary software products to protect software investments against obsolesence due to hardware changes. This section compares the two domains in the context of this new environment for the purpose of identifying the key HOL attributes that affect software life cycle cost.

2.4.1 C^3/MIS Similarities

At a high level of abstraction, the two domains are very similar. The primary function of both C^3 systems and MIS is to provide decision makers with access to information resources. The specific types of information handled in different instances of each type of system may vary widely, but many of the basic processing functions required (acquire, manage, distribute, and display information) are common. Instances of each type of system have the following common capabilities and characteristics:

Interactive Users As decision support systems, both domains must provide interactive access to processed information. In the past, the two domains have used radically different interactive user interface designs. C^3 systems typically feature graphical presentation of processed information on automatically updating high resolution tiled displays and audio/visual alarms that are triggered by realtime events. They have traditionally been implemented using special purpose consoles containing graphic processors and multiple CRT displays. MIS systems have generally used low resolution "dumb terminal" devices as the primary interactive user interface device. However, the use of modern workstation technology and windowing user interfaces has recently become common in both domains. User interface standards such as X-Windows and Motif have sufficient flexibility to satisfy the specific user interface requirements of each of the domains using the same foundation of COTS display software products.

Distributed Architecture Although both domains are inherently distributed, they have traditionally used different methods to handle their communication processing requirements. In the C^3 system domain, remote information sources transmit messages to command center processing facilities via a wide variety of special purpose military and commercial carrier communication systems. These messages are subsequently processed by the command center which ultimately generates and transmits processed results to remote users. The C^3 domain uses a wide variety of message sets and communication protocols which typically require the custom software development. In the MIS domain, a centralized data processing facility typically serves a geographically distributed community by providing remote, transaction oriented access to databases and batch processing resources. The communication services required to support this type of operation are typically provided by proprietary commercial hardware and software products. The current trend in both domains is to replace large scale, proprietary data processing facilities with networks of smaller processors using non-proprietary implementations of industry standard operating systems and the open systems communication protocols. Software architectures that support easy distribution of processing load over multiple processors and hide the underlying network from applications will work well in both domains in future developments.

Database Management Both domains contain database management as a foundation functional capability. Currently fielded systems in both domains typically use either a "flat file" system based on a vendor provided, proprietary file access method. In addition, C^3 systems often have responsiveness requirements that can only be satisfied by augmenting their database management capability with memory resident data managers to handle the time critical portion of their database. The availability of fast, cheap memory and processors will enable commercial database management systems to achieve performance characteristics that will make them usable even in time critical C^3 system applications. New systems in either domain will tend to use commercial databases with SQL based interfaces to applications for the portability to protect software development effort.

Continuous Operations Both domains must support continuous operations and therefore use some mixed form of software and hardware redundancy to achieve the required high availability. In the C^3 systems domain, multiprocessors and distributed hardware architectures are often used to ensure that a command center is not vulnerable to a single point failure. The overall system architectures for typical C^3 systems usually include physically separate command centers that are capable of mutual backup in the event of

catastrophic failures. Rapid reconfiguration and restoration of operations in the event of a system failure is mandatory in the C^3 systems domain. The restoration of operations timeliness requirements tend to be less severe in the MIS domain. The strategy in the MIS domain has typically been to protect the integrity of the database using checkpoints and transaction journalling. Centralized MIS systems tend to restore operations from checkpoints following repair instead of switching to a backup system. Both domains commonly use fault tolerant processing components to augment single point availability.

Growth and Flexibility Both domains require system architectures that provide the growth and flexibility to accommodate evolving requirements. In the C^3 systems domain, requirements evolve because new external sensor and weapons systems change continually in response to new technology and changing threats. The MIS domain is about to enter a period of substantial change wherein existing proprietary systems will be re-engineered to take advantage of the new open systems environment. This process must be evolutionary because of the sheer size and complexity of the existing MIS infrastructure. Both domains will therefore require system architectures that support scalability as processing requirements increase over time and high leverage software development techniques evolve to control new software acquisition costs.

2.4.2 C^3/MIS Differences

The MIS and C^3 systems domains have many common characteristics, there are also some differences that impact the lifecycle cost implications of Ada and C++ in each of the domains. Considering the differences between the two domains will provide some insight into the relative value of various language attributes in each of the domains.

Response Time Characteristics C^3 systems typically have message processing and display response time requirements that are on the order of one second. The inability of the system to rapidly provide a decision maker with timely processed results triggered by an incoming message can have mission critical consequences. MIS systems have much less stringent performance requirements. Performance is important in the MIS domain because it affects the end user's productivity, but it is not mission critical. Language features which support designing systems with realtime performance requirements are more important in the C^3 system domain than in the MIS domain.

Data Volume The databases that must be managed by a typical MIS application are typically much larger than the databases that are managed by C^3 systems. Current MIS systems often have huge disk farms and magnetic tape based archive systems.

Algorithm Complexity C^3 systems often have complex, computationally intensive scientific processing requirements, whereas MIS applications normally involve database access processing (sorting and searching), statistics and decimal arithmetic.

Applicability Of Other COTS Products There is more need for end user programmability in the MIS domain than in the C^3 systems domain. Database oriented special purpose applications such as 4GL languages, forms editors, and report writers are commonly used in the MIS domain for application development. Other tools for office automation applications such as spreadsheets and word processoring programs are commonly available on the modern MIS user's desktop computers. These types of products are not as common in the C^3 systems domain.

Graphics Utilities Both the C^3 system and MIS domains will be using the same types of graphics based windowing systems for their user interfaces in the future. However, their use of this technology will be different. In general, the high resolution graphical display of information is a more useful technique for aiding the decision maker in C^3 systems than in MIS systems. MIS will continue to use low resolution displays such as menus, tables, and forms even though they will be implemented on a graphics based system.

2.5 Evolutionary Development

For of this study, the tradeoffs will be evaluated assuming that a modern development process at SEI Level 3 or better (see reference [9] for the definition of SEI levels) is being employed. This assumption is important as it forms the basis of the language tradeoff context as well as the cost estimation model inherent in Ada COCOMO. The foundation of Ada COCOMO is an underlying development approach called TRW's Ada Process Model. While the Ada Process Model was developed to exploit the features of Ada, most of its techniques and process improvements are equally applicable to other languages. In the discussion that follows, we will refer to a generic, language independent version of this approach as an "evolutionary process model". This section provides a summary of the important aspects of this process relevant to a C++/Ada language tradeoff. A more detailed description can be found in reference [4].

The process defined here is *virtually* independent of language. However, real world experience has identified certain language characteristics which support or hinder successful execution of the process. Support for partial implementations, abstraction, rapid prototyping, disciplined configuration control, expressiveness and readability are subtle language features which can be exploited to provide an improved development approach.

Incremental Evolutionary Development Modern software development is both *incremental* (partitioning the overall development into a set of smaller more manageable increments) and *evolutionary* (evolving an increment from an abstract representation into a requirements compliant implementation through a series of systematic refinements).

Builds are selected subsets of software capability which implement a project specific risk management plan. These increments represent a cross section of components which provide demonstrable threads of capability. Integration of builds is mechanized by constructing Major Milestone (SSR, PDR, CDR) demonstrations of capabilities which span multiple builds.

Conventional software PDRs employ standards which result in tremendous breadth of review where only a minimal amount is really important, or understood by the large diverse audience. For example, reviewing all requirements in equal detail at a PDR is inefficient and unproductive. Not all requirements are created equal, some are critical to design evolution, some are don't cares. The effectiveness of design review is improved in this process model by allocating the technical breadth review to smaller scale design walkthroughs and focusing the major milestone reviews on demonstration of the important design issues where uncovering a design flaw has a large return on investment.

Design Walkthroughs review sets of partially implemented components within a build to permit evolutionary insight into the build's structure, operation and performance as an integrated set. Integration of components within builds is mechanized by constructing small scale demonstrations composed of capabilities which span multiple components.

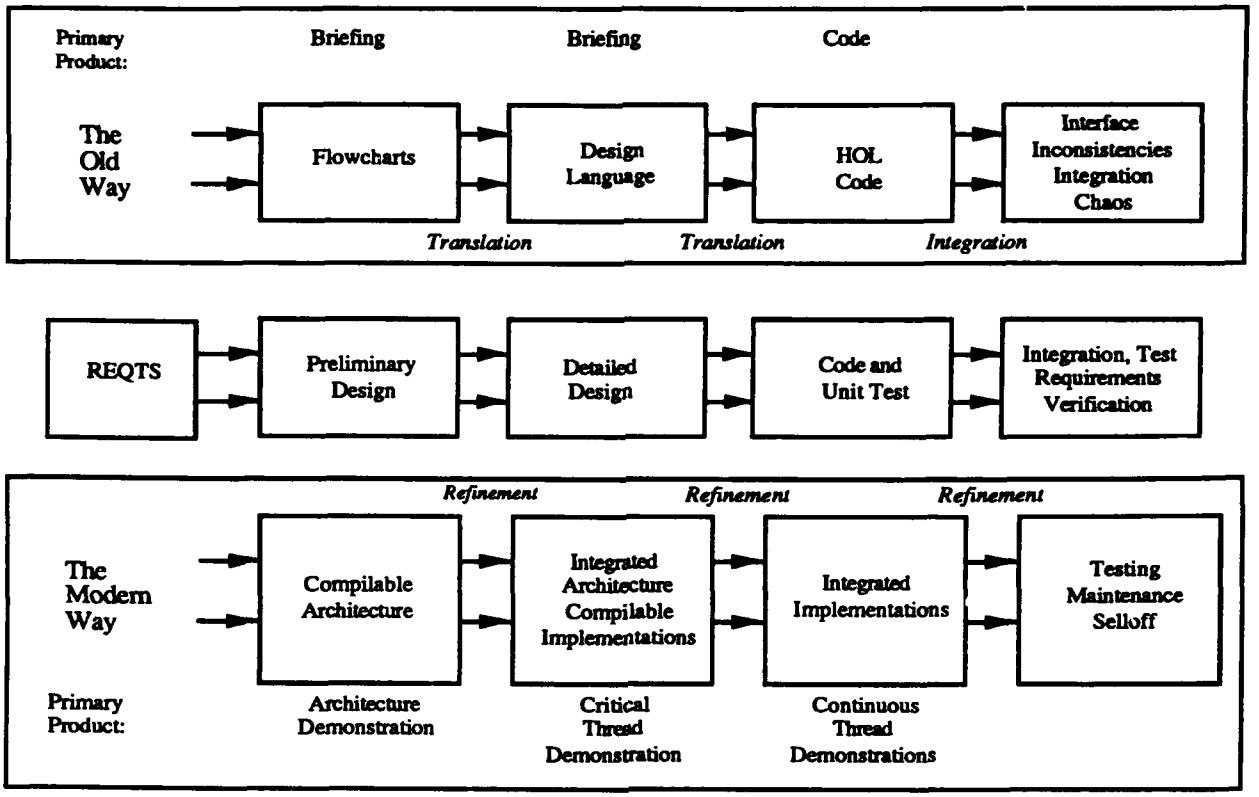

Figure 2: Conventional Approach vs. Modern Techniques

Risk Management Planning the content and schedule for each of the builds is perhaps the first and foremost risk management task. The efficiency of the software development depends on the build plan's initial quality *and the ability for the project to react to changes in build content and schedule as the development progresses.* The last point is important because of the need to adjust build content and schedule as more accurate assessments of complexity, risk, personnel, and value engineering are achieved.

Homogeneous Lifecycle Language The use of a compilable Design Language (which is the same syntax and semantics as the target language) is one of the primary facets of our process which provides uniformity of representation format and insight via software development progress metrics. The terms "design language" and "coding language" are virtually interchangeable with respect to our usage standards; the standards which apply to our final coded products are the same as those that apply to earlier design representations.

Top level design means coding the structural or critical components (main programs, executives, types, classes and objects, etc.). Lower level design means coding the

procedural or non-critical components. This approach best supports our technique of *evolving* designs into implementations without translating between two sets of standards or using two sets of specialists, one for designing and one for coding.

Metrics One of the by-products of our definition of design language is the uniform representation of the design with a complete estimate of the work accomplished (Source lines of code) and the work pending (source lines of design language) embedded in the evolving source files in a compilable format. Although the coded source lines are not necessarily complete (further design evolution may cause change), they do represent an accurate assessment of work accomplished. Given this life cycle standard format, the complete set of design files can be processed at any point in time to gain insight into development progress.

The metrics collection process is performed periodically for detailed management insight into development progress, code growth, and other indicators of potential problems. Monthly or quarterly metrics can be collected by build, and by CSCI so that high level trends and individual contributions can be assessed.

Demonstration Based Design Evaluation Software developments under the current Military Standards focus on documentation as intermediate products. This is useful and necessary, but by itself it is inadequate for large systems. Fundamental in our process is forcing design review to be more tangible via visibly demonstrated capabilities. These demonstrations serve two key objectives:

(a) The generation of the demonstration provides tangible feedback on integratability, flexibility, performance, interface semantics and identification of design and requirements unknowns. It satisfies the software designer/developer by providing first hand knowledge of the impact of individual design decisions and their usage/interpretation by others. *The integration of the demonstration is the real design review.* This activity has proven to provide the highest return on investment by uncovering design deficiencies early.

(b) The finished demonstration provides the monitors of the development activity (users, managers, customers and developers) tangible insight into functionality, performance, and development progress. One sees an executing implementation of important and relevant capability subsets.

For major milestone demonstrations, a demonstration plan is developed which identifies the capabilities planned to be demonstrated, how the capabilities will be observed, and explicit pass/fail criteria. Pass/fail criteria should be defined as "thresholds" for taking risk management actions, not requirements tests. The pass/fail criteria should trigger an action based on exceeding a certain threshold of concern as negotiated by the responsible engineering authorities for both contractor and customer.

With early demonstrations (whether on the target hardware, or host environment), significantly more accurate assessments of performance issues can be obtained and resolved *early, where appropriate redesign is tenable.* Given the typical design review attitude that a design is "innocent until proven guilty", it was quite easy to assert that the design was adequate. This was primarily due to the lack of a tangible design representation from which true design flaws were unambiguously obvious. Under this process model, design review demonstrations force the design to be evaluated as "guilty until proven innocent" and are far more efficient at identifying and resolving design flaws.

Total Quality Management In an Evolutionary Process Model there are two key advantages for applying TQM. The first is the common representation format throughout the lifecycle permitting consistent software metrics across the software development work force. Although these metrics don't all pertain to quality (many pertain to progress), they do permit a uniform communications vehicle for achieving the desired quality in an efficient manner.

Secondly, the demonstrations serve to provide a common goal for the software developers. This "integrated product" is a reflection of the complete design at various phases in the life cycle for which all personnel have ownership. Rather than individually evaluating components which are owned by individuals, the demonstrations provide a mechanism for reviewing the team's product. This team ownership of the demonstrations is an important motivation for instilling a TQM attitude.

3. Language Tradeoff Criteria

The sections that follow summarize the tradeoff perspectives which were considered in this study. The chosen criteria were defined as independently as possible while stressing the most important aspects of software engineering of MIS systems in the context of DoD applications.

3.1 Technical Criteria

Reliable S/W Engineering Software errors can be categorized into two types, *Heisen bugs and Bohr bugs*. Heisen bugs are errors which are coincidental with a certain probabilistic occurrence of a given situation and/or sequence of events. This category of errors (typically unrepeatable), represents the class of errors for which it is impossible to provide complete testing coverage. Bohr bugs are errors which always result when the software is stimulated in the same way regardless of timing and other coincidental conditions. This category of errors (typically repeatable), represents the class of errors for which it is possible (albeit, likely expensive) to provide complete test coverage.

Single CPU, single program, non-realtime systems typically contain only Bohr bugs. Distributed systems, realtime systems and multiple, interoperating programs executing on a time shared operating system are vulnerable to Heisen bugs. Due to the difficulty of isolating and resolving Heisen bugs, it is important that the programming language support safe designs. Note that Heisen bugs are almost always considered design flaws whereas Bohr bugs are typically coding flaws. The distinction here is important since Heisen bugs typically involve more complicated and broad design breakage than Bohr bugs.

This criteria therefore, evaluates language support, both compile time and runtime, for elimination and isolation of both types of bugs.

Maintainable S/W Engineering Software engineering for maintainability (or flexibility, changeability) is critical to reducing development costs as well as lifecycle maintenance and upgrade costs. One of the by-products of a modern incremental development approach is the early maintenance activity necessary for initial increments. The migration of software maintenance into the development phase where it is in the contractor's best interest to develop a maintainable design is perhaps one of the evolutionary process model's greatest strengths.

The principal advantage of software is its flexibility. It is this quality of quick reprogrammability that allows a software system to change its functionality and performance to satisfy new - perhaps unforeseen - requirements. Similarly, the ability to extend system life beyond its original goals is largely due to the design flexibility of the software embedded within the system being maintained.

The central principles of software engineering are all aimed at dampening down software's unbridled flexibility in the name of achieving product quality. Almost all software people, in varying degrees, are annoyed with this quality-oriented "overhead" that detracts from software's inherent ease of change. (The slow acceptance of Ada in some domains is may be a reflection of this resentment. Ada, more than any other language, deliberately introduces support to, and enforcement of, software engineering into its syntax and semantics. Not every software person agrees that this is a worthy endeavor).

Reusable S/W Engineering Reusable software rarely happens by coincidence. It requires serious design investment, language support and knowledge about the domain of target reusage. Reuse experience has emphasized the importance of encapsulation, generic programming, object orientation, performance tunability and error handling as important language support features which enhance reusable software engineering.

Reusable software is distinguished from portable software engineering in that reusable software constitutes design for different applications while portable software constitutes design for different execution platforms.

Realtime S/W Engineering Realtime software engineering requires explicit language support for resource management, fault isolation and recovery, concurrent execution, hardware device control and synchronous/asynchronous event handling. In essence, language support for real time programming provides the ability to deal with abstractions of real physical objects and real resources including CPU, memory and time without having to program directly in the target machines native (lower level) language.

Portable S/W Engineering The portability of software is a function of the trustworthiness of a language in its behavioral implementation across different platforms. While most portability problems are attributed to compile time issues, runtime performance differences and runtime library behavior are equally, if not more serious, since they tend to result in more substantial design breakage. The important language features which support portability include standardization, encapsulation, simplicity, and runtime library complexity.

Runtime Performance Runtime performance, while often enhanced by ever increasing hardware performance, is nevertheless still a paramount software engineering concern. As we continually stress flexibility and maintainability, we must introduce new layers of performance overhead. This is further complicated by the continual growth in customer requirements for accuracy, depth, breadth, and timeliness of data processing. Therefore, the execution speed, minimization of runtime overhead, and efficiency of interfaces (whether Operating System, COTS product, or external systems) are still vitally important.

Compiletime Performance The more a language does in eliminating errors and supporting higher level abstractions, the more burden on the underlying compiler. While this is generally considered a positive tradeoff (i.e., computer response time for quality and function), there is a point where slow compiler response time impedes productivity more than it improves it. The complexity of a language, the compilation rules, the levels of abstraction supported and the level of error elimination are important attributes.

Multilingual Support Tremendous effort has gone into developing COTS products and previous applications programs which are not necessarily Ada or C++. The facilities that a language provides in integrating foreign language objects is important to reducing lifecycle cost and avoiding custom development. Examples of multilingual support include Ada's pragma interface and C++'s upward compatibility with C.

OOD/Abstraction Support There are various programming paradigms for achieving the qualities of reliability, maintainability, reusability and portability. The techniques of object oriented design (reference [3]) and data abstraction are important to achieving many of these qualities. The language support for data typing, object classes, inheritance and encapsulation are important. Furthermore, support for "megaprogramming" is another perspective of OOD/abstraction which is geared at improving productivity and quality through the generation of even higher level languages from the language primitives of a 3rd generation language. Megaprogramming can take several forms including:

(a) generic programming
(b) "software chipsets" which are tunable to application needs
(c) Automated source code generation from high level representations
(d) 4th generation languages
(e) reuse libraries

3.2 Pragmatic Criteria

Program Support Environment (PSE) Quality PSE quality includes lifecycle support for programming, documentation, testing, configuration management and quality assurance. Since our underlying process model emphasizes a homogeneous lifecycle language, the PSE support and integrity with respect to language definition are vital to each lifecycle phase. The overall PSE criteria includes language and tool support for:

(a) syntax directed editting
(b) integration of editor/compiler/linker/debugger/CM system
(c) depth and breadth of debugging
(d) on-line data dictionary for object browsing, traversal and definition
(e) graphical design support
(f) documentation automation

An important aspect of this criteria is the volatility and maturity of the language PSE. The extent of trustworthiness that is inherent in a PSE is an important driver of lifecycle cost.

Readability Software readability is as much a function of project/programmer discipline as it is a function of language support. Nevertheless, there are many important language features which promote readability. Since programs are read many times during their lifecycle (as opposed to written only a few times), it is important that they promote understandable abstractions and traceability, and minimize ambiguity, interpretation and guessing. Managers, customers, designers, programmers, testers, and other interested observers need to communicate consistently through readable source code.

Writeability Too much verbosity can impede programmer productivity and introduce new sources of error. Since one of the primary activities of our whole lifecycle is coding, the easier it is to program, the more productive an individual programmer can be.

Large Scale S/W Engineering Language Language support for software engineering in the large is an important criteria for the class of systems targeted by this study. Software engineering in large teams (e.g., 50 people+) necessitates more discipline, control and adaptabilty. The features of the language, as well as the PSE, can go a long way towards enforcing discipline, permitting control and enhance adaptability.

COTS S/W Integration The extent of commercial off-the-shelf product support for direct integration with a language is important to minimize development risks and leverage existing products.

Precedent Experience Relevant past experience by the assigned personnel with a language and its PSE, whether good or bad, is important to any project. It provides a basis for costing, risk management and planning.

3.3 Human Criteria

Popularity Personal preference is important in a human oriented activity like software engineering. The characteristics of a language that make it "popular" include ease of learning, ease of use, availability on popular platforms (such as PCs), availability and support in educational institutions, open literature support, and association with popular or unpopular organizations or institutions.

Existing Skill Base The current momentum of the engineering population with respect to a given language is a criteria which drives personnel availability, and required training investment.

Acceptance This criteria represents the degree of ownership and motivation that designers/programmers/testers have with respect to a language. While this is hard to quantify, it includes such things as perceived design freedom, programming freedom, NIH (e.g., commercial companies ignoring Ada or defense contractors ignoring C++), bureaucracy and future career growth.

3.4 Cost Impact Weighting

The original version of the COnstructive COst MOdel (COCOMO) (reference [1]) was calibrated to 56 software development projects and validated on 7 subsequent projects. Three software technology thrusts in the 1980s motivated the development of a revised version of COCOMO: the use of the Ada programming language, the use of incremental development, and the use of an Ada Process Model to capitalize on the strengths of Ada to improve the efficiency of software development. This revised COCOMO model, Ada COCOMO, is defined in reference [2]. The cost impact ranges defined in Ada COCOMO were used as the basis of defining the various weights of each tradeoff criteria.

Table II provides a list of the relevant Ada COCOMO cost drivers which could be impacted by the tradeoff criteria. These cost drivers are each assigned a cost impact weight based on the range of impact that the cost driver can cause in a project's software development costs.

The rationale for the various cost impact weights is provided below:

ACT and ESLOC Reducing the amount of developed software (equivalent source lines of code-ESLOC) or software maintenance volume (annual change traffic-ACT) are paramount to lifecycle cost reduction. Reuse, use of COTS, abstraction and higher order languages are focused on exactly these cost drivers. The overall impact of reducing the volume of software to be developed or modified can be very broad. We will assume that a typical target maximum reduction is a factor of two, namely that the amount of developed software or amount of maintenance changes are cut in half. Since the Ada COCOMO equation has a typical exponent of 1.2 applied to the volume of ESLOC, this would translate into a net cost reduction range of 2.3 ($2^{1.2}$=2.3). Since the effect of ACT in reducing maintenance costs is linear, its cost impact weight is assigned a value of 2.0.

PMEX, PDRT, RISK, and RVOL Each of these *process* parameters can adjust the Ada COCOMO exponent by a range of .05. For a typical 100K ESLOC project, this results in a range of impact of 1.3 for each of these process attributes. We will assume that each of these has an equal range of contribution.

RELY, DATA, CPLX, RUSE, TIME, STOR All of these *product* parameters use the full range of impact defined in the Ada COCOMO model.

ID	Cost Driver	Cost Impact Weight
ESLOC	Equivalent Source Lines of Code	2.30
ACT	Annual Change Traffic	2.00
PMEX	Process Model Experience	1.30
PDRT	PDR Thoroughness	1.30
RISK	Risks Resolved by PDR	1.30
RVOL	Requirements Volatility	1.30
RELY	Required Reliability	1.65
DATA	Data Base Size	1.23
CPLX	Product Complexity	1.96
RUSE	Required Reusability	1.50
TIME	Execution Time Constraint	1.66
STOR	Execution Storage Constraint	1.56
VMVH	Virtual Machine Volatility, Host	1.27
VMVT	Virtual Machine Volatility, Target	1.25
TURN	Turnaround Time	1.46
ACAP	Architect Capability	2.57
PCAP	Programmer Capability	1.62
AEXP	Applications Experience	1.57
VEXP	Virtual Machine Experience	1.34
LEXP	Language Experience	1.47
MODP	Modern Programming Practices	1.59
TOOL	Program Support Environment	2.00
SCED	Project Schedule Constraint	1.23
SECU	Project Security Level	1.10

Table II: Cost Impact Weights

VMVH, VMVT, TURN, MODP, TOOL, SECU, SCED All of these *project* parameters use the full range of impact defined in the Ada COCOMO model.

ACAP, PCAP, AEXP, VEXP, LEXP All of these *personnel* parameters use the full range of impact defined in the Ada COCOMO model.

In the next section, the tradeoff criteria have each been assigned a weight based on their relevant cost impacts. These weights provide an objective measurement of the importance of a various cost driver and likewise the importance of the various language tradeoff criteria. For

example, a tradeoff criteria which impacts ESLOC, CPLX, and ACT is far more (cost) important than a tradeoff which only affects VMVH and VMVT because the potential cost leverage of ESLOC, CPLX and ACT (2.3*1.96 2.0 = 9.02) is far greater than VMVH and VMVT (1.27*1.25 = 1.59).

4. Ada vs C++

4.1 Tradeoff Results

The previous sections have defined the tradeoff criteria and the cost impact weights that serve to objectively discriminate the importance of each criteria in terms of lifecycle cost impact. The next step is one of assimilating expert judgement on the part of a diverse set of individuals in order to arrive at a relatively unbiased group consensus. In this analysis, we have employed the following participants: Brett Bachman, Peter Blankenship, Grady Booch, Mark Gerhardt, Charles Grauling, Don Reifer, Walker Royce and Winston Royce. This team represents a broad range of expertise and experience in MIS, C^3, Ada, C, and C++ applications, software engineering technology, software cost estimation, and management. It should also be noted that this group of evaluators has considerably more Ada experience (both positive and negative) than they do C++ experience.

Table III provides the average of each of the participants opinions on the relative importance to MIS and C^3 domains, and the relative rankings of each language. The resulting MIS and C^3 domain scores by language are computed by taking the product of the importance, ranking and weight for each language-domain pair.

4.2 Tradeoff Analysis

This section describes some of the rationale for the rankings represented in the table as group concensus. Table IV identifies the standard deviations on the various rankings to reflect the general level of agreement among the respondents.

Reliable S/W Engineering Ada is, in general, a more reliable language than C++. While C++ does provide more checking than C, it lacks many of the features found in Ada.

For example, Ada supports safe array indexing through both compile and runtime (via exception handling) checks. While it is possible to write safe array indexing in C++, it is not part of the language. The checks can be made, but C++ does not enforce it. Ada supports safe array indexing by default.

Type	Criteria	Importance[1] C³	Importance[1] MIS	Ranking[2] C	Ranking[2] C++	Ranking[2] Ada	Cost Impact[3] Impacts	Cost Impact[3] Weight
TECHNICAL	Reliable S/W Engineering	5.0	3.8	1.5	3.2	4.5	ACT, ESLOC RISK, RELY	9.9
	Maintainable S/W Engineering	4.1	4.8	1.6	3.2	4.4	RVOL, ACT CPLX, ESLOC	11.7
	Reusable S/W Engineering	3.8	4.0	2.0	3.8	4.1	RUSE, ESLOC	6.9
	Realtime S/W Engineering	4.5	2.3	2.3	2.8	4.1	CPLX, TIME	4.9
	Portable S/W Engineering	3.3	4.1	2.9	2.9	3.6	CPLX	2.5
	Runtime Performance	4.3	2.5	4.5	3.6	3.0	RISK TIME, STOR	6.6
	Compile-Time Performance	1.9	2.2	4.6	3.1	2.3	TURN	1.5
	Multilingual Support	2.3	3.5	2.4	2.4	3.1	ESLOC, RISK	3.0
	OOD/Abstraction Support	3.5	3.5	1.5	4.6	3.9	ESLOC, ACT PDRT, CPLX	11.7
PRAGMATIC	Program Support Environment	4.3	4.1	2.9	2.1	4.1	VMVH, VMVT TOOL, ESLOC PDRT	9.5
	Readability	4.3	4.0	1.8	2.9	4.4	ESLOC ACT, MODP	7.3
	Writeability	2.8	3.1	4.4	3.5	3.4	LEXP, ESLOC	3.4
	Large Scale S/W Engineering	4.3	3.8	1.4	3.3	4.9	ACT, RISK TOOL, PDRT RVOL	8.8
	COTS S/W Integration	3.4	4.4	4.5	3.6	2.8	LEXP ESLOC, RISK	6.9
	Precedent Experience	4.1	3.6	4.4	1.5	3.6	LEXP, VEXP AEXP, PMEX	10.2
HUMAN	Popularity	2.4	2.9	4.6	4.0	2.8	LEXP, VEXP	2.0
	Existing Skill Base	3.8	3.4	4.4	1.8	3.0	LEXP, VEXP	2.0
	Acceptance	2.9	2.8	4.4	3.3	2.5	ACAP, PCAP	4.2
			MIS SCORE—>	1098	1324	1631		
			C³ SCORE —>	1165	1401	1738		

[1] 1 = Marginal Value, 5 = Critical Value
[2] 0 = No Support, 5 = Excellent Support or Language Enforced
[3] Ada COCOMO Cost Drivers and Relative Cost Weight

Table III: Group Consensus

Type	Criteria	Importance		Ranking		
		C^3	MIS	C	C++	Ada
TECHNICAL	Reliable S/W Engineering	0	.4	.7	.3	.5
	Maintainable S/W Engineering	.9	.4	.9	.6	.5
	Reusable S/W Engineering	.8	.9	1.0	.7	.6
	Realtime S/W Engineering	.5	.8	.7	.4	.3
	Portable S/W Engineering	1.0	.7	.6	.8	.5
	Runtime Performance	.4	.5	.7	.7	0
	Compile-Time Performance	.7	.7	.7	.9	.7
	Multilingual Support	.4	1.2	1.7	1.7	1.1
	OOD/Abstraction Support	.9	1.1	.9	.7	.3
PRAGMATIC	Program Support Environment	.7	1.1	1.1	.6	.2
	Readability	.7	.7	1.0	.4	.5
	Writeability	.7	.6	.7	.5	.7
	Large Scale S/W Engineering	.7	1.0	.5	.4	.3
	COTS S/W Integration	.5	.7	.5	1.1	.8
	Precedent Experience	.6	.7	.7	.7	.7
HUMAN	Popularity	.7	.3	.5	.7	.4
	Existing Skill Base	.7	.9	.5	.7	0
	Acceptance	.9	1.0	.5	.7	.5

Table IV: Standard Deviations

Maintainable S/W Engineering With respect to maintainability support, Ada was deemed superior to C++. The most obvious reasons include Ada's library management, increased reliability, existing support environments, and increased readability. In effect Ada supports disciplined software development to a greater extent than C++. One subtle advantage of Ada is the language support (and PSE support) for fault containment and isolation. Although inheritance and dynamic binding do provide powerful design features, they can complicate maintenance by disguising the source of some errors.

Reusable S/W Engineering Ada supports reusability somewhat better than C++. Resuability in Ada is accomplished through proper design, standardization, encapsulation, and generics (C++ supports all of these features with the exception of generics, which may be supported in the future).

C++, on the other hand, provides another form of reuse through inheritance and dynamic binding, concepts not supported by Ada (Ada 9X will support inheritance). Inheritance enables a programmer to "reuse" an existing class (somewhat similar to an Ada package) and enhance it as necessary for the problem at hand. In effect most of a class may be reused, with alterations/enhancements being done in the derived class.

Dynamic binding allows a program to determine at runtime what function will be invoked. Since the decision of which function to call is done at runtime, existing code does not have to be modified, or re-compiled. Dynamic binding is often used together with inheritance to provide for reuse in C++.

Realtime S/W Engineering Ada provides several features that are designed for realtime systems, such as tasking, exceptions, representation specifications, and interrupt handling that make it a more appropriate language for realtime systems.

Portable S/W Engineering Since Ada supports many operating system features, designers/programmers can design/program to this interface instead of a specific operating system. This implies that more of a program written in Ada can be moved from one operating system to another without change than can one written in C or C++. Furthermore, the government enforced language standardization ensures that the core language definition is interpreted functionally in the same way on every different platform to the maximum extent possible.

It should be noted that Ada was designed to be used on "real-time embedded systems", hence Ada supports the writing of programs that are mostly operating system independent. This is especially the case in the C^3 environment.

Runtime Performance Since Ada abstracts operating system capabilities into the Ada runtime system (thus increasing portability), and is generally a richer language, Ada programs tend to be a bit larger and slower than programs written in C++ or C. It should be noted that due to C++'s dynamic binding, among other features, C++ is generally slower than C.

Compiletime Performance While compiletime performance of some languages may try programmer patience, it was not considered to be detrimental to any language's use, or the decision of which language to design and build a system in. Hence, this category scored the lowest in terms of importance for both C^3 and MIS domains.

Regarding the relative scores, C was fastest, followed by C++, and finally Ada. In theory, Ada should be slower since Ada compilers try to detect more errors (such as array index out of bounds) at compiletime than other languages.

Multilingual Support The group's assessment was that Ada supports multilingual programs better than either C++, or C. The Ada language supports, via pragma interface, the calling of other languages. This capability permits Ada programs to call subprograms written in languages other than Ada, from Ada. Programmers must, of course, be cognizant of how a given implementation of Ada and the foreign language pass parameters, represent objects (such as records and arrays), and use dynamic memory. Hence, this feature is not entirely portable. Furthermore, this capability is not supported well by most compilers today.

C++ can interface (can call) other language routines, through the "extern" declaration, in a way similar to Ada. In addition, C++ accommodates the calling of C++ routines from other languages, something which Ada does not currently support in a portable way (but will in the 9X version). As with Ada, C++ programmers must be aware of how parameters are passed, how objects (data structures) are represented, etc.

The actual comparison between Ada and C++ in this category resulted in the most disagreement between the evaluators, consequently it had the highest standard deviation. The reasons for this diversity included the importance of upward compatibility with C and existing weaknesses in some Ada compiler implementations. In retrospect, the panel should also have evaluated which language integrates best with FORTRAN (for C^3) and COBOL (for MIS).

OOD/Abstraction Support C++ provides better support for object oriented design and programming. Regarding Ada's support for OOD (which includes Object Orient Programming in our definition) it is felt that Ada provides many of the important features of an Object Oriented Programming (OOP) language. These include encapsulation, support for abstract data types, etc. Ada, however, only partially supports polymorphism and does not support inheritance - key components of OOP. (Ada 9X will support single inheritance and polymorphism).

The features of OOP Ada lacks, C++ supports. These include polymorphism, inheritance (single and multiple), and dynamic binding. In addition, C++ deals primarily with objects, while Ada is procedurally oriented. For example, consider an object called shape. To draw this shape:

C++:	Shape.Draw();
Ada:	Draw(Shape);

When parameters to the draw routine are added:

C++:	Shape.Draw(10,10,100,100);
Ada:	Draw(Shape, 10, 10, 100, 100);

In the C++ example, the message or command "draw" (with parameters) is being sent to the object shape. Whereas in the Ada example, the object shape (with parameters) is being sent to the procedure draw. The primary difference is that in C++ the basic element is an object, which includes both data structures and operations (functions) that act on those structures. In Ada the basic element is data structures, procedures and functions, but these components are not grouped together to form one entity that responds to commands (messages).

Program Support Environment The support environments for Ada are much richer than those available for C++, or even C. Some of the Ada Programming Support Environments (APSEs) provide extensive configuration management, editing, and browsing capabilities. In fact, programming large systems in Ada without such facilities was one of the primary reasons that early Ada projects were less than successful.

The need for support environments, especially for languages that are object based is well founded. Smalltalk, one of the earliest object oriented languages, includes a browser as part of the language. For large projects, such browsers are not just beneficial, they are necessary.

Readability Readability is one area where Ada far exceeds C and C++. Ada was designed to be a readable language, in favor of a writable one. C and C++, on the other hand, can be very obscure, as both languages include features to enconomise on keystrokes. A statement like:

```
matches_found += occurrence++;
```

definitely saves typing, however it is also difficult to read, especially for nonprogrammers and programmers who are new to C or C++. The above statement would be written in Ada as follows:

```
matches_found:= matches_found + occurrence;

occurrence := occurrence + 1;
```

An even more obscure C++ example (taken from reference [10] page 118):

```
return (i > j ? i : j);
```

which in Ada would be:

```
if i > j then
return i else
return j;
        endif;
```

In both examples the C++ statements could be written similar to the Ada statements, but this is not required nor enforced by the language.

Writability The flip side of readability is writability. Languages are usually good at one or the other, but not both. In general Ada and C++ provide the same level of writability.

Large Scale S/W Engineering Ada is the best language for large scale software engineering. Ada provides features such as compilation management (component consistency and obsolecence is detected and enforced by Ada compilers) static binding (some errors are detected at compiletime instead of runtime, thus debug/compilation/build frequency is reduced), exception handling, generics, and PSEs. All of these features facilitate construction of large projects and enhanced reusability.

Some of these features, such as exception handling and generics, may be part of the C++ language in the future. However, static binding and compilation management are not part of C++, making large scale project development more resource intensive, primarily in terms of project personnel.

COTS S/W Integration The results of this category are a consequence of the fact that many commercial products are written in C. C++ does conserve upward compatibility with C to enhance this form of integration. Virtually all existing COTS products provide a C compatible interface, only a few support Ada today.

Precedent Experience There is more programming experience in C than in Ada or C++. As Ada and C++ continue to mature, and quality compilers become common-place on smaller machines, the experience base for both should improve.

Popularity Despite the support for improved software engineering provided by both Ada and C++, C remains the most popular language around. Its size, together with almost universal availability makes it the first choice of most commercial projects.

In the near future, both C++ and Ada should increase in popularity compared to C. Both Ada and C++ are more complicated to learn and program in and their decreased flexibility will alienate some programmers despite the potential productivity gains.

Existing Skill Base The existing skill base is greatest for C. This is a consequence of the language's age, availability, and popularity. As time goes on, the number of projects, and hence people, programming in Ada and C++ should increase, providing a greater skill base from which to draw from.

Acceptance The acceptance of C was higher than that of Ada or C++. There are many factors that affect this category. Chief among them is ease of use and availability of compilers. Start up costs for projects written in C are much lower than those using Ada and a bit lower than those using C++.

4.3 1991 vs 1993+

This section provides an assessment of Ada and C++ technologies over the next few years to speculate on how this tradeoff analysis would change in the post 1993 timeframe. In this analysis, we have assumed that both Ada 9X and C++ Version 3 are available from tool and environment vendors. Table V summarizes the changes from the previous tradeoffs of Section 4.1 done in today's timeframe. Table VI provides the absolute assessments of the two languages in the context of this future timeframe.

Type	Criteria	1993 Change[1] C++	1993 Change[1] Ada	Rationale
T E C H N I C A L	Reliable S/W Engineering	=	=	No Significant Change
	Maintainable S/W Engineering	+	=	C++ 3.0: Exceptions, Templates
	Reusable S/W Engineering	+	=	C++ 3.0: Templates
	Realtime S/W Engineering	=	=	No Significant Change
	Portable S/W Engineering	+	+	C++ Standardization, Ada 9X
	Runtime Performance	=	+	Compiler-Runtime-H/W Maturation
	Compile-Time Performance	+	+	Compiler-Runtime-H/W Maturation
	Multilingual Support	+	+	C++ Popularity, Ada 9X
	OOD/Abstraction Support	=	+	Ada 9X Improvements
P R A G M A T I C	Program Support Environment	++	+	C++ Maturation-Commercial Popularity
	Readability	=	=	No Significant Change
	Writeability	=	=	No Significant Change
	Large Scale S/W Engineering	+	=	C++ 3.0: Templates, PSE Support
	COTS S/W Integration	+	+	C++ Popularity, Ada 9X
	Precedent Experience	++	+	Completion of Ongoing Projects
H U M A N	Popularity	+	=	Commercial Breadth
	Existing Skill Base	++	+	DoD and Commercial Commitment
	Acceptance	+	+	Completion of Ongoing Projects

[1] +:Improved, =:No Significant Change, ++:Much Better

Table V: 1993+ Differences

Type	Criteria	Importance[1] C[3]	Importance[1] MIS	Ranking[2] C	Ranking[2] C++	Ranking[2] Ada	Cost Impact[3] Impacts	Cost Impact[3] Weight
TECHNICAL	Reliable S/W Engineering	5.0	3.8	1.5	3.3	4.6	ACT,ESLOC RISK, RELY	9 9
	Maintainable S/W Engineering	4.1	4.8	1.6	3.6	4.5	RVOL, ACT CPLX, ESLOC	11.7
	Reusable S/W Engineering	3.8	4.0	2.0	4.5	4.4	RUSE, ESLOC	6.9
	Realtime S/W Engineering	4.5	2.3	2.3	3.0	4.3	CPLX, TIME	4.9
	Portable S/W Engineering	3.3	4.1	2.9	3.8	4.2	CPLX	2.5
	Runtime Performance	4.3	2.5	4.5	3.8	3.4	RISK TIME, STOR	6.6
	Compile-Time Performance	1.9	2.2	4.6	3.6	2.8	TURN	1.5
	Multilingual Support	2.3	3.5	2.4	2.9	3.7	ESLOC, RISK	3.0
	OOD/Abstraction Support	3.5	3.5	1.5	4.8	4.6	ESLOC, ACT PDRT, CPLX	11.7
PRAGMATIC	Program Support Environment	4.3	4.1	2.9	3.6	4.4	VMVH, VMVT TOOL, ESLOC PDRT	9.5
	Readability	4.3	4.0	1.8	3.1	4.4	ESLOC ACT, MODP	7.3
	Writeability	2.8	3.1	4.4	3.6	3.6	LEXP, ESLOC	3.4
	Large Scale S/W Engineering	4.3	3.8	1.4	3.8	4.9	ACT, RISK TOOL, PDRT RVOL	8.8
	COTS S/W Integration	3.4	4.4	4.5	4.3	3.6	LEXP ESLOC, RISK	6.9
	Precedent Experience	4.1	3.6	4.4	3.3	4.1	LEXP, VEXP AEXP, PMEX	10.2
HUMAN	Popularity	2.4	2.9	4.6	4.4	2.9	LEXP, VEXP	2.0
	Existing Skill Base	3.8	3.4	4.4	3.4	3.5	LEXP, VEXP	2.0
	Acceptance	2.9	2.8	4.4	4.0	3.1	ACAP, PCAP	4.2
			MIS SCORE—>	1098	1567	1768		
			C SCORE —>	1165	1642	1876		

1 1 = Marginal Value, 5 = Critical Value
2 0 = No Support, 5 = Excellent Support or Language Enforced
3 Ada COCOMO Cost Drivers and Relative Cost Weight

Table VI: 1993+ Group Speculation

5. Recommendations

Ada is an outgrowth of a remarkable vision that was first ennunciated perhaps fifteen years ago. Today, it is a vision realized. The struggle to transform the vision into reality (via very useful products), has been pursued with remarkable intellectual vigor. And it was far from being the most popular language initiative of the computer science community. Ada's principle raison d'etre is that the DoD world needed a language in which the software engineering paradigm was supported by, and in some instances, enforced within the semantics of the language. This reason to be has been very nearly achieved. It also has a lot to do with why Ada won the numbers battle of this study, and rather decisively at that.

Let us alter the vision slightly. The invention of new languages will continue unabated. But for the sake of argument, let us postulate that, partially because of its success and partially because the DoD likes to reuse a good thing, Ada semantics may somewhat unintentionally freeze up. The supporting intellectual vigor of new Ada technology development will likely disappear. Clearly, to protect against this eventuality, Ada might benefit from some friendly intellectual competition. C++ seems to be the strongest competitor today.

Ada's semantics can be characterized as providing strong support for project management functions; somewhat lesser support is provided to advanced computer science attributes. C++, on the other hand, provides little project management support in its selected semantics, but it is designed for stronger support to the computer science attributes underlying object-oriented design.

Object-oriented design, as better supported by C++, is a development productivity enhancer but its productivity improvement is unquantified as yet. This is the principal weakness in our evaluation and rankings. If object-oriented design is discovered to be a surprisingly powerful increaser of productivity, or promoter of reusability, or injector of added quality, then our collective judgements may be biased too much towards Ada. Of course the next Ada 9X team could quickly incorporate newer, better semantics - but that is not the point. A different language than Ada (C++ in this case) has pioneered better object oriented design semantics, partially because a language based on a different semantic foundation was deliberately used.

Our detailed recommendations that follow reflect this view - Namely, that Ada will benefit from continued competition. They should help the DoD to continue promoting advances in software engineering while hedging their bets on both the future of Ada and C++.

5.1 Near Term (1991-1993)

(a) Stand firm on the Ada mandate for all development intensive MIS projects or conversion intensive MIS projects where the converted system's previous language is not C.

(b) Grant waivers to the Ada mandate for C++ conversions of existing C based systems or COTS integration intensive systems where development and conversion are less than one third of the development costs.

(c) Stand firm on the Ada mandate for all C^3 projects.

(d) Deny all waivers for C, COBOL, JOVIAL, CMS-2, FORTRAN and other 3rd generation languages.

(e) Encourage Ada 9X to expand its support for C++/Ada multilingual development.

(f) Encourage AT&T to expand its support for C++/Ada multilingual development.

(g) Request AT&T to comment and respond to the results of this study.

(h) Request Ada 9X project office to comment and respond to the results of this study.

(i) Request DARPA ISTO to comment and respond to the results of this study.

(j) Invest in both C++ and Ada environment R&D, metrics collection, and language independent CASE tools.

(k) Incentivize commercial vendors to provide COTS interfaces for both Ada and C++.

(l) Encourage AT&T to support a language control mechanism similar to Ada to ensure better standardization; encourage (or fund) AT&T to introduce into its semantics further support for large scale software engineering.

5.2 Far Term (1993-1995)

(a) Institutionalize a joint language control facility and clearinghouse of implementations.

(b) Deregulate software development standards (i.e., 2167, 2168) to promote accelerated improvements of DoD software engineering practices by defense contractors and to open up to commercial software developers.

(c) Prepare for advances in megaprogramming where software design environments become language independent and development/test/maintenance environments are interoperable and equivalent for both C++ and Ada.

(d) In 1995, Review the measured productivity of Ada and C++; If within 20% of each other, permit C++ to be the second approved language for use on DoD projects.

TRW Inc.
Space &
Defense Sector

One Space Park
Redondo Beach, CA 90278

Title

Case Study: Ada and C++ Cost Comparison For CDPDS-R

DATE June 1, 1991 Revision: 1

CDRL No. 004

Contract No. MDA970-89-C-0019

PREPARED FOR
SAF/AQKS

1. Introduction

This document meets the CDRL 004 requirements specified for contract MDA-970-89-C-0019, Task Area 05 "Modeling and Simulation" Subtask 05-07/00 "Ada and C++ Programming Languages".

The primary purpose of CDRL 004 is to present a case study lifecycle analysis of the CCPDS-R project using the C++ COCOMO model derived from the tradeoff results of CDRL 002. As prerequisites to this analysis, both a summary of Ada COCOMO and the derivation of C++ COCOMO will be presented.

1.1 Approach

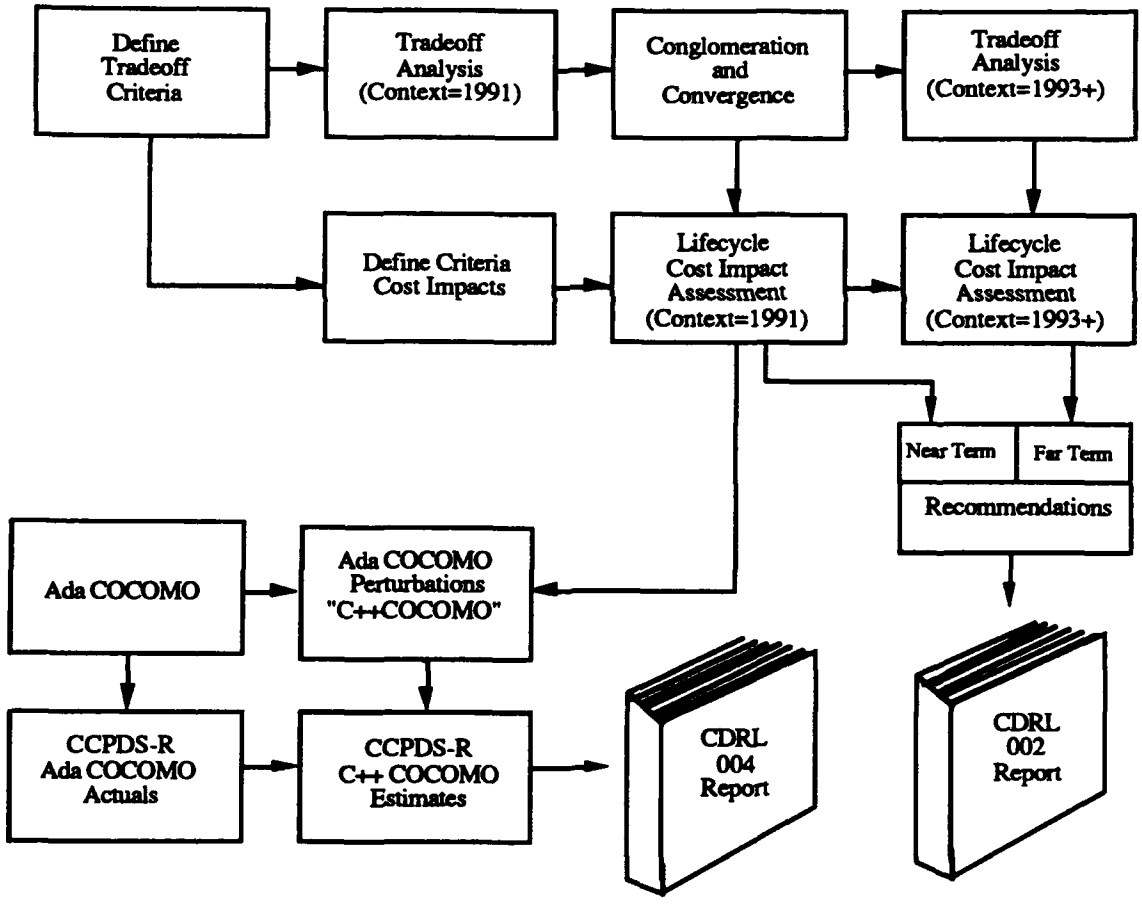

Figure 1: Tradeoff Analysis Approach

CDRL 002 identified the lifecycle cost implications of Ada and C++ in the context of MIS applications. The approach for this report is to summarize Ada COCOMO as a baseline cost estimation model and to calibrate that model to the actuals of the CCPDS-R Early Common Capability (ECC). "Calibration", in this sense, means that the various input parameters of Ada COCOMO will be adjusted to match the actual effort expended and to best reflect the product, project and people characteristics of ECC. The results of CDRL 002 will be used to define perturbations to the Ada COCOMO model resulting in a "C++ COCOMO". The ECC parameters will then be used to estimate the resulting cost differential which would be expected had CCPDS-R been implemented in C++. This cost estimate will use all of the same calibrations for parameters which are language independent (such as personnel ratings, product attributes, etc.), and adjust those parameters which would be language dependent (tool maturity, complexity, experience, SLOC, etc.). The result should be a very accurate description of the ECC actuals using Ada and an expert speculation on the lifecycle cost differences had it been done in C++.

The following tasks pertain to the production of CDRL 004.

Ada COCOMO The Ada COCOMO model will be used as the basis for the cost impact weights and as a starting point for creating a C++ COCOMO.

Ada COCOMO Perturbations, "C++ COCOMO" The tradeoffs and analysis will be converted into a set of perturbations to the Ada COCOMO model in order to best reflect a cost estimation technique tuned to the current state of C++. These perturbations will include modifications to the *underlying model* (representing absolute cost estimation differences between the two languages) and modifications to the *settings* (representing time relative cost estimation differences resulting from the differences in maturity between the two languages implementations and support environments).

CCPDS-R Ada COCOMO Actuals The CCPDS-R project cost actuals will be presented using Ada COCOMO as a controlled perspective.

CCPDS-R C++ COCOMO Estimates The CCPDS-R project will then be estimated using C++ COCOMO using all the same settings except those impacted by current language maturity levels. The result will be a real world project context of the lifecycle cost differences between C++ and Ada.

1.2 References

[1] Boehm, B. W., "Software Engineering Economics", Prentice Hall, 1981.

[2] Boehm, B. W., Royce, W. E., "TRW IOC Ada COCOMO: Definition and Refinements", *Proceedings of the 4th COCOMO Users Group*, Pittsburgh, November 1988.

[3] Royce, W. E., "TRW's Ada Process Model For Incremental Development of Large Software Systems", *Proceedings of the 12th International Conference on Software Engineering*, Nice, France, March 26-30 1990.

[4] Royce, W. E., "Pragmatic Quality Metrics for Evolutionary Development Models", *TRI-Ada Proceedings*, Baltimore, December 1990.

[5] Wild, F., "A Comparison of Experience With The Maintenance of Object-Oriented Systems: Ada vs. C++", *Tri-Ada '90 Proceedings*, Baltimore, December 1990.

2. Ada COCOMO Summary

The original version of the COnstructive COst MOdel (COCOMO) was calibrated to 56 software development projects and validated on 7 subsequent projects. Three software technology thrusts in the 1980s motivated the development of a revised version of COCOMO: the use of the Ada programming language, the use of incremental development, and the use of an Ada process model to capitalize on the strengths of Ada to improve the efficiency of software development. This revised COCOMO model, Ada COCOMO, is defined in reference [2]. This section will provide a brief summary of the differences between standard COCOMO and Ada COCOMO as a prerequisite to the description of a C++ version of COCOMO. Readers without a working knowledge of the COCOMO model should consult the more detailed references [1] and [2].

Ada COCOMO has three main categories of differences from standard COCOMO:

General Improvements to COCOMO, which can be incorporated as improvements to standard COCOMO as well. These comprise a wider range of ratings and effects due to software tools and turnaround time; the splitting of virtual machine volatility effects into host and target machine effects; the elimination of added costs due to schedule stretchout; the addition of cost drivers to cover effects of security classified projects and development for software reusability; and the addition of a model for incremental development.

Ada-specific effects, including reduced multiplier penalties for higher levels of reliability and complexity; a wider range of ratings and multipliers for programming language experience; and a set of Ada oriented instruction counting rules, including the effects of software reuse in Ada.

Effects of using the Ada Process Model, which can largely be adapted to projects using other programming languages. Their use on non-Ada projects would require some experimental tailoring of standard COCOMO to accommodate the resulting cost and schedule effects. These effects include the revised exponential scaling equations for nominal development effort, development schedule, and nominal maintenance effort; the extended range of modern programming practices effects; the revised ranges of analyst capability and programmer capability effects; and the revised phase distributions of effort and schedule.

The remainder of standard COCOMO remains the same as it was: the overall functional form, most of the effort multipliers, the software adaptation equations, the activity distribution tables, and the use of annual change traffic for software maintenance.

2.1 Ada Development Costs

Ada COCOMO models the effect of an Ada Process Model in terms of a reduction in the equation exponent of 1.20. The new effort equation takes the form:

$$MM = 2.8*EAF*(KESLOC)^{1.04+\Sigma}$$

where:

MM estimates the complete software development cost in 152 hour manmonths. The scope of the effort estimated includes the management, engineering, production, test, administration and selloff of an integrated software product. The specific efforts which are not included in the estimate include, computer facility support, secretarial support, requirements analysis and specification, higher levels of project/corporate management and matrix management support (training, performance appraisals, hiring, human relations, etc.).

Σ is the exponent which relates the projects process maturity and risk exposure.

EAF is the effort adjustment factor which characterizes the complexity and nature of the development task through a set of 18 Development Effort Multipliers (DEMs). These parameter allow the estimate to be tailored to specific project, product and personnel attributes.

KESLOC is an estimate in units of thousands of Equivalent Source Lines of Code. This value depicts the size of the product to be developed.

2.1.1 Ada Effort Adjustment Factor EAF

Table I identifies all of the Ada COCOMO development effort multipliers with some brief commentary on the source of any changes from standard COCOMO.

DEM	1 Very Low	2 Low	3 Nom	4 High	5 Very High	6 Extra High	Description
RELY	.75	.88	.96	1.07	1.24		Contracted Effects
DATA		.94	1.00	1.08	1.16		Unchanged
TIME			1.00	1.11	1.30	1.66	Unchanged
STOR			1.00	1.06	1.21	1.56	Unchanged
VMVH		.92	1.00	1.09	1.17		Separated host from target
VMVT		.93	1.00	1.07	1.16		Separated target from host
TURN	.79	.87	1.00	1.07	1.15		Added very low, otherwise Unchanged
ACAP	1.57	1.29	1.00	.80	.61		Expanded Effects
PCAP	1.30	1.12	1.00	.89	.80		Contracted Effects
AEXP	1.29	1.13	1.00	.91	.82		Unchanged
VEXP	1.21	1.10	1.00	.90			Unchanged
LEXP	1.26	1.14	1.04	.95	.86		Across the board increase
MODP	1.24	1.10	.98	.86	.78		Across the board decrease
TOOL	1.24	1.10	1.00	.91	.83	.73	Added XX-High
SCED	1.23	1.08	1.00				Deleted Schedule protraction penalty
SECU			1.00	1.10			Added, independent of language
RUSE		1.00	1.10	1.30	1.50		Added, to account for extra design effort
CPLX	.73	.85	.97	1.08	1.22	1.43	Contracted Effects

Table I: Ada COCOMO Development Effort Multipliers

Brief descriptions of the Ada COCOMO modifications to the development effort multipliers are provided below:

RELY The nominal, high and very high ratings were reduced for Ada COCOMO to account for Ada language features (strong typing, tasking, exceptions, elaboration, configuration consistency, runtime library resource management and packages) which support the elimination of many classes of software errors, limits their side effects, or makes them easier to isolate.

VMVH,VMVT was derived from the original COCOMO DEM VIRT. With the movement to a host/target development environment, Ada COCOMO split VIRT into two DEMs.

The combined productivity range for VMVH and VMVT is slightly higher than VIRT to reflect the somewhat deeper interaction between an Ada runtime and the underlying virtual machine (operating system, COTS, DBMS).

TURN effects were unchanged except for an additional very low rating to reflect the current state of advanced interactive workstation response times.

ACAP effects are significantly expanded to account for the importance of the structural architecture of an Ada program and the increased emphasis on design. This is especially important in distributed systems where structural design integrity is paramount.

PCAP effects are contracted in a complementary way to the expansion of the ACAP effects. The reduced programmer emphasis again reflects the increased focus on design and the belief that a "good" design can be built by average programmers but a "bad" design can't be built by superstar programmers.

LEXP has been expanded to account for the richness and breadth in the Ada language. This expansion reflects the fact that an experienced Ada developer can capitalize on the broader solution space in Ada to simplify many program functions, while an inexperienced Ada developer is more likely to choose an implementation with undesireable side effects (performance, maintainability, or reliability problems).

MODP effects are slightly better across the board since Ada encourages or enforces some of the practices which had to be manually enforced in conventional languages (configuration consistency, structured programming).

TOOL effects remained unchanged except for the addition of two higher level ratings. The extra high rating corresponds roughly to the capabilities of the Rational environment. The XX high rating reflects the future existence of a fully integrated object oriented design environment including graphical design, automated documentation and automated metrics support.

SCED effects have eliminated the negative impact of a "too long" schedule since incremental development has not been observed to incur a cost penalty.

SECU has been added independent of Ada based on the discussion in Chapter 28 of reference [1].

RUSE has been added to address the effects of developing software for reuse. This potential for additional effort reflects the need for more complicated design, more elaborate documentation, and more extensive testing for a broader usage.

CPLX effects have been reduced from conventional COCOMO to reflect Ada's increased abstraction (packages, advanced types, tasks, exceptions, generics) which supports better complexity management and control. This permits programming constructs which were previously complex in conventional languages to be implemented in a much more straightforward manner.

2.1.2 Ada Exponent Σ

The parameter Σ measures the project's estimated degree of compliance with the Ada Process Model and the potential risk exposure in terms of four parameters:

(a) The team's previous experience in applying an evolutionary process model

(b) The design thoroughness at PDR (i.e., the quality and stability of the software architecture skeleton (SAS)

(c) The level of remaining risk exposure at PDR

(d) The degree of requirements stability at PDR

If a project is fully compliant with the Ada Process Model and has resolved all significant risks, then Σ will be .00, and the diseconomy of scale exponent will be 1.04. If a project exhibits the typical hasty PDR symptoms, the Σ will be 0.16, and the exponent will be 1.20, the same as for a traditional COCOMO embedded mode project.

Table II identifies the rating scale used to determine a project's Σ.

Wi Weights	0.0	.01	.02	.03	.04	.05
PMEX Experience with Ada Proces Model	Successful on > 1 Project	Successful on 1 Project	General Familiarity	Some Familiarity	Little Familiarity	No Familiarity
PDRT Design Thoroughness At PDR (SAS Maturity)	Fully 100%	Mostly 90%	Generally 75%	Often 60%	Some 40%	Little 20%
RISK Risks Eliminated At PDR	Fully 100%	Mostly 90%	Generally 75%	Often 60%	Some 40%	Little 20%
RVOL Requirements Volatility	No changes	Small Noncritical Changes	Frequent Noncritical Changes	Occasional Moderate Changes	Frequent Moderate Changes	Many Large Changes
Exponent:			$1.04+\sum_{i=1}^{4} W_i$			

Table II: Ada COCOMO Exponent Settings

2.1.3 Ada ESLOC Definition

Ada COCOMO defines a source line explicitly and provides mechanisms for incorporating reuse and fourth generation languages (4GLs).

Ada/COCOMO [2], defines SLOC for Ada programs as:

> Within an Ada specification part, each carriage return counts as one SLOC. Within Ada bodies each semi-colon counts as one SLOC. Generic instantiations count one line for each generic parameter (spec or body).

This definition requires some standards for specification layout to clarify the meaning of carriage return:

(a) each parameter of a subprogram declaration be listed on a separate line

(b) for custom enumeration types (e.g., system state, socket names, etc.) and record types each enumeration or field should be listed on a separate line.

(c) for predefined enumeration types (e.g., keyboard keys, compass directions), enumerations should be listed on as few lines as possible without loss of readability.

(d) Initialization of composite objects (e.g., records or arrays) should be listed with one component per line.

The definition above treats declarative (specification) design much more sensitively than it does executable (body) design. It also does not recognize the declarative part of a body as the same importance as a specification part. Although these and other debates can surface with respect to the "optimum" definition of a SLOC, the optimum *absolute* definition is far less important than a consistent *relative* definition.

Besides the published differences between COCOMO and Ada COCOMO, TRW (and specifically the CCPDS-R project) defined some clarifications for more accurate estimation of Source Lines of Code. This was done to eliminate some of the confusion in accommodating different forms of reuse which were cost drivers in CCPDS-R. The result is an extension of the COCOMO technique for incorporating reuse (see reference [1] page 133 for a definition of this technique), called Equivalent Source Lines of Code (ESLOC).

In essence, ESLOC converts the normal COCOMO measure of Delivered Source Instructions into a normalized measure which is comparable on an "equivalent effort per line" basis. The need for this new measure arises in the budget allocation and productivity analysis activities for mixtures of newly developed, reused, and automatically produced source code. For example, a 10,000 SLOC component which is automatically produced from some higher level representation format (which may require 1000 "higher level source lines") should not be allocated the same effort as a 10,000 SLOC newly developed component. This ESLOC technique provides a normalized measure of relative effort is obtained. In the above example, component 1 would have

an ESLOC=3500 whereas component 2 would have an ESLOC=10,000. This conversion technique is described below.

There are four categories of software source including new, reused, automated, and 4GL. Table III identifies the criteria for translating SLOC in each of these representations into ESLOC.

SLOC Format	Design Max=40%	Implement Max =20%	Test Max =40%	ESLOC	Rationale Summary
COTS/NDI	0%	0%	0%	0%	Commercial Off-The-Shelf Non-Developmental Item No Effort Required
New	40%	20%	40%	100%	100% of the Design Effort 100% of the Implementation Effort 100% of the Test Effort
Reused	20%	5%	30%	55%	50% of the Design effort 25% of the Implementation Effort 75% of the Test Effort
Automated	0%	0%	40%	40%	0% of the Design Effort 0% of the Implementation Effort 100% of the Test Effort
4GL	30%	10%	10%	50%	75% of the Design Effort 50% of the Implementation Effort 25% of the Test Effort

Table III: SLOC to ESLOC Conversion Factors

COTS/NDI software does not result in any contribution to the ESLOC. COTS software and NDI software contributions to effort are inherent in the categories below through the volume of interfacing software which must be developed and through some of the development effort multipliers which characterize the maturity and robustness of the underlying virtual machine. NDI type software includes "reused software" where the component is reused with no modification.

New software must be developed from scratch. It requires complete design, implementation and test effort and has an ESLOC multipler of 100% (i.e., a 1 for 1 conversion).

Reused components represent code that was previously developed for a different application but is applicable to the project at hand with modification. While there are many diverse ways to assess the relative cost of reuse, and each instance is better handled on a case by case basis, our technique provides a simple rule of thumb as a default. TRW experience has demonstrated that most instances of reuse typically require about 50% of the design effort (reuse analysis and partial redesign where necessary), 25% of the implementation effort, and 75% of the test effort for ensuring proper implementation in the new application context.

Automated code usually requires a separate source format to be defined (a 4GL higher level language format or tables of objects, attributes, and relationships) for input into a tool (a higher level language compiler) which then automatically produces the normal (3GL) source code SLOC. The higher level 4GL source code is costed separately, however, automated source code does become part of the end product, is customly developed for an application, and requires testing in an integrated context. Automated source code requires 0% design (usually it has a pre-existing design or template), 0% implementation (the coding is automated) and 100% testing. Note that the design and coding efforts for automated SLOC are derived from the corresponding 4GL. Also, if the tool which automates the source code production must be developed, its SLOC should be included in the ESLOC.

4GL can take many forms such as representations for input to automated source code production tools, higher level language CASE tools, DBMS programming (SQL), etc. These higher level abstraction formats require 75% of the design effort (less detailed design), 50% of the implementation effort (higher level syntax and semantics), and 25% of the test effort (the primary focus on testing is on the output of the 4GL compiler (i.e., the automated source code defined above) since these formats need only be tested to accurately represent the designers specifications (i.e., "getting the 4GL to compile").

As an example of how ESLOC relate to SLOC, Table IV identifies the three CCPDS-R subsystems and their relative values. Most of the ECC subsystem was developed from scratch (there was automated SLOC); the impact of reuse and automation in the latter two subsystems is much greater.

Subsystem	SLOC	ESLOC
Early Common Capability (ECC)	294	248
PDS Subsystem	225	100
SAC Subsystem	360	160

Table IV: Examples of SLOC/ESCLOC for CCPDS-R Subsystems

2.2 Ada Maintenance Cost Estimation

COCOMO and Ada COCOMO estimate annual software maintenance in terms of a quantity called ACT (Annual Change Traffic), which is defined as the fraction of the product's source code volume which undergoes change during a typical year, either through enhancement or modification. See reference reference [1] page 129 for a complete treatment of COCOMO maintenance estimation.

The Ada COCOMO equation for estimating annual maintenance effort is:

$$MM_{maint} = ACT*EAF_M*MM_{Dev}$$

where ACT is the Annual Change Traffic, EAF_M is the maintenance adjustment to the development EAF and MM_{Dev} is the magnitude of the development effort. Note that MM_{Maint} is in terms of maintenance effort per year. The EAF_M can be used to adjust for changes in personnel or project attributes during the maintenance phase. While most of the DEMs have the same effects during maintenance as they do during development, two (MODP, RELY) of the DEMs have redefined effects.

RELY The result of developing reliable software results in two competing effects during maintenance. First, the maintenance effort associated with fixes and enhancements increases in complexity (as in development) as the required reliability increases. Second, the higher the reliability required, the less latent errors and documentation inaccuracies are present during maintenance thereby reducing the required effort for fixes. The net effect of these phenomena is a very different set RELY effects during maintenance.

MODP The effect of using modern programming practices (structured, programming, object-oriented programming, design walkthroughs, software scaffolding, anticipatory documentation, integrated environments) during development and maintenance improves overall maintainability and makes it easier to maintain large products as efficiently as small products. The net effect is an expanded impact (i.e., positive impacts get better, negative impacts get worse) of the MODP DEM.

Since the domain of this study (DoD MIS applications) would almost always have a high MODP and High RELY, these differences would not impact the language comparison significantly. The primary drivers of the maintenance effort are the ACT and the development effort. Therefore, whatever conclusions are drawn with regard to Ada/C++ development cost differences would also apply to lifecycle maintenance cost differences.

The other important characteristic is the ACT. While Ada COCOMO does not modify any of the techniques for estimating annual maintenance (other than those inherent in the development cost estimate) there was much speculation that the net effect of using Ada would be a significant reduction in the ACT due to a reduction in both the number of latent software errors and the the extent of breakage resulting from desired fixes and enhancements. CCPDS-R ECC data has demonstrated these outcomes with a reduction in ACT of approximately a factor of 2 (i.e., ECC experienced an ACT=.04, the average COCOMO project has an ACT=.08).

3. C++ Perturbations to Ada COCOMO

This section will define the perturbations to Ada COCOMO derived from the tradeoff analysis. These differences are generalizations of the tradeoff results in the context of todays language technology (i.e., Ada 83 and C++ version 2.0). This analysis needs to be updated when the new language implementations (Ada 9X and C++ version 3.0) become available. A specific project would likely have some counterexamples of the arguments that follow (just as with COCOMO and Ada COCOMO), however, an average project should experience the following differences:

3.1 C++ Development Costs

There are no differences to the basic form of the development cost estimation equation for C++.

$$MM = 2.8 * EAF * (KESLOC)^{1.06+\Sigma}$$

where:

MM estimates the complete software development cost in 152 hour manmonths. The scope of the effort estimated includes the management, engineering, production, test, administration and selloff of an integrated software product. The specific efforts which are not included in the estimate include, computer facility support, secretarial support, requirements analysis and specification, higher levels of project/corporate management and matrix management support (training, performance appraisals, hiring, human relations, etc.).

Σ is the exponent which relates the projects process maturity and risk exposure. The constant in the exponent has been increased slightly to accommodate the tradeoff results for programming in a large scale and precedent experience.

EAF is the effort adjustment factor which characterizes the complexity and nature of the development task through a set of 18 Development Effort Multipliers. These DEMs allow the estimate to be tailored to specific project, product and personnel attributes. While the general meaning of each DEM has been conserved in C++ COCOMO, the underlying effects and settings are somewhat different.

KESLOC is an estimate in units of thousands of Equivalent Source Lines of Code. ESLOC have a similar definition in C++ as in Ada, emphasizing the declarative parts (carriage returns counts) over the executable parts (terminating semi-colons).

3.1.1 C++ Effort Adjustment Factor (EAF)

Table V summarizes the difference between Ada COCOMO development effort multipliers and C++ COCOMO multipliers. There are two types of differences described in the table. Differences in "effects" correspond to an underlying change in the model due to a true language difference. For example, to achieve a very high reliability using Ada results in a 24% cost penalty, whereas in C++, it results in a 40% cost penalty compared to achieving a nominal reliability. Differences in "settings" reflect the fact that achieving a certain quality has the same cost impact in either language but the current state of the practice, maturity of the language or its support software is such that a project undertaken today would use different settings depending on the language. When both languages are equally mature (say 5 years from now), these settings should converge.

DEM	1 Very Low	2 Low	3 Nom	4 High	5 Very High	6 Extra High	Description
RELY	.75	.88	1.00	1.15	1.4		Reverted Effects to Conventional COCOMO
DATA		.94	1.00	1.08	1.16		Unchanged Effects, Similar Settings
TIME			1.00	1.11	1.30	1.66	Unchanged Effects, Typically Lower Settings
STOR			1.00	1.06	1.21	1.56	Unchanged Effects, Similar Settings
VMVH		.92	1.00	1.09	1.17		Unchanged Effects, Typically Higher Settings
TURN	.79	.87	1.00	1.07	1.15		Unchanged Effects, Typically Lower Settings
ACAP	1.57	1.29	1.00	.80	.61		Unchanged Effects, Similar Settings
AEXP	1.29	1.13	1.00	.91	.82		Unchanged Effects, Similar Settings
PCAP	1.30	1.12	1.00	.89	.80		Unchanged Effects, Similar Settings
VEXP	1.21	1.10	1.00	.90			Unchanged Effects, Similar Settings
LEXP	1.26	1.14	1.04	.95	.86		Unchanged Effects, Typically Lower Settings
MODP	1.24	1.10	1.00	.91	.82		Reverted Effects to Conventional COCOMO
TOOL	1.24	1.10	1.00	.91	.83	.73	Unchanged Effects, Typically Lower Settings
SCED	1.23	1.08	1.00				Unchanged Effects, Similar Settings
VMVT		.93	1.00	1.07	1.16		Unchanged Effects, Typically Higher Settings
SECU			1.00	1.10			Unchanged Effects, Similar Settings
RUSE		1.00	1.21	1.43	1.65		Expanded Somewhat (10%)-Less C++ Reuse Support
CPLX	.75	.85	.93	1.03	1.16	1.36	Contracted Somewhat More (5%) From Ada COCOMO to Account for Better OOD Support

Table V: C++ COCOMO Development Effort Multipliers

The C++ COCOMO modifications to the development effort multipliers are discussed below:

RELY The nominal, high and very high ratings were reverted back to conventional COCOMO since C++ provides only minimal language support (strong typing, elaboration) compared to Ada for supporting reliable programming.

VMVH,VMVT These effects are left unchanged, however, given the current immaturity of existing C++ platforms, less stringent standardization (compared to Ada's Compiler validation, and Mil-Std rigidity) and immaturity of support environments, near term settings should reflect greater volatility.

TURN effects are unchanged. However, C++ settings should reflect somewhat better response times perceived by users since C++ compilers are currently more efficient than Ada.

ACAP Unchanged effects and settings, language independent.

PCAP Unchanged effects and settings, language independent.

LEXP Unchanged effects. However, given the smaller available C++ skill base (at least in general DoD contractors), settings should reflect an average language experience level less than is available for Ada.

MODP Effects were reverted back to conventional COCOMO primarily because there is no enforced configuration control mechanisms inherent in C++. Furthermore, since C++ supports conventional C programming, there is very little enforcement of better practices.

TOOL Unchanged effects, however, typical settings would reflect a less capable and mature development environment for C++. This is especially important to exploit the advantages of C++ advanced OOD support, where the need for object class browsers, debugger support for dynamic binding and inheritance resolution is paramount.

SCED Unchanged effects and settings, language independent.

SECU Unchanged effects and settings, language independent.

RUSE The effects of reuse for C++ were expanded slightly from the Ada COCOMO effects to account for less language support in designing a component to be reused. Without support for generics and rigid standardization these effects were deemed to be approximately 10% worse in terms of cost required to engineer the same level of reuse into a component.

CPLX effects have been reduced from Ada COCOMO to reflect C++ OOD support which permits even better abstraction supporting better complexity management and control. This permits programming constructs which were previously complex (even in Ada) to be implemented in a more straightforward manner.

3.1.2 C++ Exponent Σ

The Evolutionary Process Model effects are primarily focused at relieving the "software diseconomy of scale", namely, reducing the exponential effects of increased development scale. While the process techniques and overall paradigm are mostly independent of Ada and C++, there are significant differences in the inherent language support for programming in the large. The original Ada COCOMO definition embeds this support in the exponent's constant (i.e., 1.04). While the IOC version of Ada COCOMO still maintains that this constant is greater than 1 (indicating that, even with Ada, there is still an inherent diseconomy of scale with software development), there was substantial consideration of incorporating the possibility for an economy of scale situation with Ada. This was considered a little too revolutionary for the time and hence put off for a later update following an historical precedent.

Nevertheless, one substantial difference between Ada and C++ is the amount of energy a project needs to invest to achieve software engineering discipline and maintain configuration control without loss of efficiency. This difference gets exponentially worse as the size of the job increases, consequently, the C++ COCOMO will employ a slightly higher constant in the equation exponent (1.06). This *"disadvantage"* could be overcome somewhat through proper exploitation of OOD techniques, but this would be handled through the standard linear effects inherent in the development effort multipliers.

3.1.3 C++ ESLOC Changes

ESLOC volume is more a function of the architectural solution, designed-in reuse, and design abstractions than it is a function of language. However, there are some general language differences which would be manifested in the quantity of ESLOC. Table VI identifies some expected differences in language support for efficient (i.e., fewer ESLOC needed) solutions in various functional domains. In the table, *verbose* implies that the language generally requires more ESLOC than *neutral*, which requires more than *concise* (i.e., these are simple relative indicators for the purpose of general comparison). Note that the smallest ESLOC solution is not optimal if it compromises other qualities such as readability, maintainability, or performance. Therefore, this table provides relative trends per unit quality, not absolute guidance.

Functional Area	Ada	C++	Comments
Declarative	Verbose	Verbose	Declarative Design Emphasis in both
Executive	Concise	Verbose	Tasks, Exceptions, Runtime Library
Structural	Concise	Verbose	Tasks, generics, exceptions are advantages
Data Management	Neutral	Concise	C++ OOD advantages
User Interface	Verbose	Concise	C++ OOD advantages
Data Processing	Concise	Concise	Similar Support
Communications	Concise	Neutral	Tasks, Rep Specs advantages
Tools/Environment	Neutral	Concise	C++ OOD advantages

Table VI: C++ ESLOC Differences From Ada

3.2 C++ Maintenance Cost Estimation

There are no basic differences between the Ada COCOMO estimate of maintenance and the C++ estimate of maintenance except those inherent in estimating the development costs (the maintenance costs are computed as a function of the development costs and the annual change traffic). We would expect that, at least in the near term, the ACT for a C++ project would be at least 25% more than an Ada project based on the results of our tradeoff analysis (C++ received a ranking of 3.2, compared to Ada's ranking of 4.3). This increase reflects the increased number of latent software errors as well as the increased complexity of maintaining (i.e., isolating problems, analyzing resolutions, incorporating enhancements) a C++ baseline. Reference [5] discusses a (fascinating, real life) maintainability comparison of Ada and C++ in fair detail.

4. CCPDS-R Case Study

A working model of C++ COCOMO was created with PCOC, a cost estimation tool which implements both the COCOMO model and derived COCOMO models (such as Ada COCOMO and C++ COCOMO).

The complete cost estimates for the CCPDS-R actuals (using Ada COCOMO) and the CCPDS-R estimate (using C++ COCOMO) which are described in the next sections were computed and analyzed with PCOC.

The C++ and Ada versions of COCOMO, as implemented with PCOC are provided as an appendix to this study (in 5.25 in floppy disk format) along with the respective cost analyses of CCPDS-R. The Ada actuals are resident in the file: ECC_CPP.

4.1 Cost Analysis For Ada (Actuals)

CCPDS-R has been documented by a complete set of lifecycle metrics. Table VII identifies the relevant COCOMO parameters which characterize the CCPDS-R Early Common Capability subsystem. These parameters have been set to reflect the actual cost and productivity data experienced. Note that many of the parameters (such as LEXP) reflect the level of personnel experience available *at the time of contract award* (four years ago). These actuals reflect those (and not today's) settings of maturity and experience levels.

The items in the table are further clarified below:

CSCIs This analysis grouped the CCPDS-R software into three distinct CSCIs: NAS (Network Architecture Services, the resuable software building blocks which formed the foundation of the architecture), FRM (Framework, the software architectural infrastructure and top level object interfaces), and APPs (Applications, the mission specific software which interfaced with external systems, interfaced with the user, and provided mission specific data management and data processing algorithms). While CCPDS-R has a somewhat different set of CSCIs (for project specific reasons), this set of three CSCIs permits a simpler, more understandable cost estimate and description.

DEM	CSCI NAS	CSCI FRM	CSCI APPs	Description
RELY	5	5	4.91	The values in this table identify the settings for each parameter:
DATA	3	3	3.36	
TIME	4.89	3.27	4.55	
STOR	3	3	3	1=Very Low
VMVH	3	3.20	2.47	2=Low
TURN	2	2	2	3=Nominal
ACAP	4.38	3.33	3.09	4=High
AEXP	2.11	1.72	2.46	5=Very High
PCAP	3.75	3.07	2.93	
VEXP	2.40	2	2.47	Some settings are decimal values
LEXP	2.48	2.37	1.87	to reflect a weight averaged
MODP	4	4	4	(by ESLOC) mixture of settings
TOOL	6	5	5	at a lower (CSC) level.
SCED	3	3	3	
VMVT	3	3	3.28	
SECU	4	4	4	
RUSE	4.72	3.19	3.50	
CPLX	4.91	4	4.04	
CSCI EAF	1.24	1.29	1.5	The total impact is the product of each settings relative impact (Table I)
ESLOC	14.9K	67.4K	166.1K	Equivalent Source Lines of code
EAF		1.43		$\frac{(1.24*14.9)+(1.29*67.4)+(1.5*166.1)}{248.4}$
EXP		1.11		PMEX=.03, PDRT=.01, RISK=.01, RVOL=.02
ACT		.04		

Table VII: Ada COCOMO Development Effort Multipliers: CCPDS-R ECC

DEMs All of the development effort multipliers (DEMs) reflect settings based on CCPDS-R actuals and the Ada COCOMO definitions of the relative settings. Many of the settings reflect a mixture of lower level settings. For example an LEXP rating of 1.5 represents the fact that about half (in terms of source lines of code) of the applications (APPS) components utilized people with an LEXP of 1 and the other half utilized people with an LEXP of 2. For more detail on the lower level ratings, the ECC_ADA.dat file should be browsed using PCOC.

ESLOC The total delivered source lines of code is 295K. The reduction to 248K ESLOC resulted from some reuse (about 15K) and significant instances of automated source code

generation (about 120K). While the gross savings of this automation appears large, it must be offset some by the development of the 4GL input files (about 37K) and the development of the source code production tools (about 24K) for a net savings of 47K ESLOC.

EAF The total effort adjustment factor for ECC is 1.43. This value reflects a macro-level view of numerous micro-level inputs. In general ECC can be characterized as a very demanding product (high reliability, stringent performance, limited reusability, and complex application with numerous interdependent goals), built by a capable team (talented, but inexperienced in the language, application and environment), under excellent project conditions (excellent tools, disciplined practices, adequate schedule, adequate resources, and stable environment).

EXP ECC operated with a process/risk assessment exponent of 1.11. Compared to the standard COCOMO exponent of 1.2, this reduction reflects a significant improvement. This can be primarily attributed to proper preparation prior to contract award in defining a new risk management oriented process (the Ada Process Model), planning the development effort thoroughly, and committing the right resources and personnel to conduct the architectural design phase successfully. Maintaining the management discipline and follow-through necessary to ensure successful acceptance of the approach at all project and customer levels as the project progressed was equally as important.

PMEX was set to .03 since ECC was a first generation Ada Process Model project. There was considerable volatility of the techniques, standards and practices as the project evolved through the first year. PDRT was set to .01 reflecting a very thorough design at PDR (NAS and FRM components were 90+% complete, other applications were at approximately 50%). RISK was set to .01 due to the overwhelming success of the demonstrations in resolving most of the major risks (NAS overhead, realtime database distribution performance, most display response time and reliable reconfigurability) and in at least quantifying the risk exposure of remaining risks (external interface protocol performance, some display response times). Finally, RVOL was set to .02 reflecting the continuous refinement of minor requirements, especially in the user displays, mission algorithms and external interface protocol. While these areas did change frequently, there were no major differences which caused design breakage, most were small scale implementation changes. This would normally be perceived as a large risk, the ability to absorb these changes

generally resulted in increased quality far in excess of the delta cost and solidified an excellent customer/contractor/user rapport.

Manmonths/Productivity/Quality ECC actuals were 2027 total man months (of which 201 were for requirements analysis and specification). This results in a productivity of 136 ESLOC/MM (COCOMO productivities exclude the effort for requirements) or 162 SLOC/MM. This productivity represents an achievement of approximately double the productivity of conventional TRW project experience with conventional languages (FORTRAN, JOVIAL, C). Furthermore, the delivered quality (in terms of reliability and maintainability) demonstrated approximately a twofold increase. While these improvements cannot be attributed totally to Ada, it is clear that the language, its supporting environment and its support for the NAS architectural approach and evolutionary design were significant contributors. The improvement in TRW's software development process, the commitment of the people *(customer, contractor and user)* and the thorough early planning and commitment were also important.

ACT- Maintenance TRW defined an explicit set of maintainability metrics to evaluate the quality of CCPDS-R evolving increments. The derivation of these metrics and the complete analysis of ECC are described in reference [4]. Table VIII presents the results of these metrics for the ECC subsystem. The ACT for ECC was assessed at a level of .04. This measurement reflects the breakage and repair experienced during the last six months of development (2%) when the primary focus was on formal test with stable baselines. Post-FQT experience has been even better than this.

The descriptions below summarize the definitions of these metrics and their results for ECC.

Rework Proportions The R_E value identifies the percentage of effort spent in rework compared to the total effort. In essence, it probably provides the best indicator of productivity. The activities included in these efforts should only include the technical requirements, software engineering, design, development, and functional test. Higher level system engineering, management, configuration control, verification testing and higher level system testing should be excluded since these activities tend to be more a function of the company, customer or project attributes independent of quality. The goal here is to normalize the widely varying bureaucratic activities out of the metrics. R_S provides a value for comparing with similar projects, future increments, or future projects as well as other in progress

analyses. Basically, it defines the proportion of the product which had to be reworked in its lifecycle.

Metric	Definition	CCPDS-R Value
Rework Proportions	$R_E = \frac{\text{Effort}_{\text{Rework}}}{\text{Effort}_{\text{Total}}}$	11%
	$R_S = \frac{\text{SLOC}_{\text{Reworked}}}{\text{SLOC}_{\text{Configured}}}$	24%
Modularity	$Q_{mod} = \frac{\text{TotalBreakage}}{\text{No. of SCOs}}$	$54 \frac{\text{SLOC}}{\text{SCO}}$
Changeability	$Q_C = \frac{\text{TotalEffort}}{\text{No. of SCOs}}$	$17 \frac{\text{Hrs}}{\text{SCO}}$
Maintainability	$Q_M = \frac{R_E}{R_S} = \frac{\text{Productivity}_{\text{Development}}}{\text{Productivity}_{\text{Change}}}$	.45

SCO: Software Change Order - Discrete Configuration Baseline Change

Table VIII: End-Product Quality Metrics Definition

Modularity (Q_{mod}): The average extent of breakage. This identifies the need to quantify *extent of breakage* (we will use volume of SLOC damaged) and number of instances of rework (Number of SCOs). This value characterizes the extent of damage expected for the average SCO. A value of 54 SLOC implies that the average SCO only affected the equivalent of one program unit. Since most of the trivial errors get caught in standalone test and demonstration activities, this value indicates the average impact for the non-trivial errors which creep into a configuration baseline.

Changeability (Q_C): The average complexity of breakage. This identifies the need to quantify *complexity of breakage* (we will use effort required to resolve) and number of instances of rework (Number of SCOs). This value provides some insight into the ease with which the products can be changed. While a low number of changes is generally a good indicator of a quality process, the magnitude of effort per change is sometimes even more important. As a project average, 17 hours suggests that change is fairly simple. **When change is simple, a project is likely to increase the amount of change thereby increasing the inherent quality.**

Maintainability (Q_M): Theoretically the maintainability of a product is related to the productivity with which the maintenance team can operate. Productivities however, are so difficult to compare between projects that this definition was intuitively unsatisfying. If we normalize the productivity of rework to the productivity of development, we end up with a value which is independent of productivity but yet a reflection of the complexity to change a product in relation to the complexity to develop it. This normalizes out the project productivity differences and provides a relatively comparable metric. Maintainability then, will be defined as the ratio of rework productivity and development productivity. Intuitively, this value identifies a product which can be changed three times as efficiently ($Q_M = .33$) as it was developed as having a better maintainability than a product that can be changed twice as efficiently ($Q_M = .5$) as it was developed, independent of the absolute maintenance productivity realized. The statistics needed to compute these values are the total development effort, total SLOC, total rework effort and total reworked SLOC.

This value identifies the relative cost of maintaining the product with respect to its development cost on a per line of code basis. For example, if R_E=R_S, one could conclude that the cost of modification is equivalent to the cost of development from scratch (not highly maintainable). A value of Q_M much less than 1 would tend to indicate a very maintainable product, at least with respect to development cost. Since we would intuitively expect maintenance costs of a product to be proportional to its development cost, this ratio provides a fair normalization for comparison between different projects. Since the numerator of Q_M is in terms of effort and its denominator is in terms of SLOC, it is a ratio of productivities (i.e., effort per SLOC). Some simple mathematical rearrangement will show that Q_M is equivalent to:

$$QM = \frac{\text{Productivity}_{\text{Development}}}{\text{Productivity}_{\text{Maintenance}}}$$

This value along with the change traffic experienced during the last phase of the life cycle could be used to predict the maintenance productivity expected from the current development productivity being experienced. The overall change traffic during development should not be used to predict operational maintenance since it is overly biased by immature product changes. The FQT phase change traffic (likely a lower value than the complete development lifecycle traffic), is a more accurate measure. A value of .45 seems like a good maintainability rating, but further project data would permit a better basis for assessment.

This value requires some caveats in its usage. First, this maintenance productivity was derived from small scale maintenance actions (fixes and enhancements) as opposed to large scale upgrades where system engineering and broad redesign may be necessitated. Secondly, the data is derived from the development lifecycle, therefore, it should be treated as more of an upper bound in planning the expectations during the maintenance phase of a product where the existence of defects should be less than that experienced during development. The personnel performing the maintenance actions however, were knowledgeable developers which may bias the maintainability compared to the expertise of the maintenance team. The message here, is that this data, like any productivity data, must be used carefully by people cognizant with its derivation to ensure proper usage.

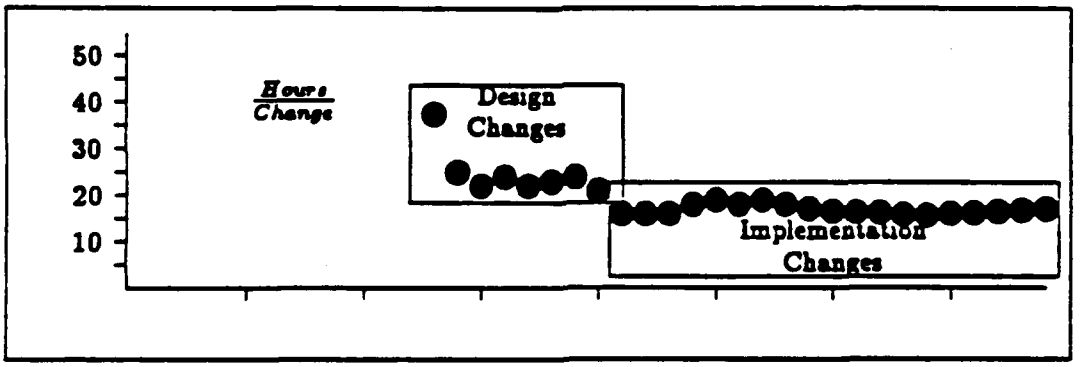

Figure 2: Changeability Evolution

Figure 2 identifies how the changeability (Q_C) evolved over the project schedule to date. While conventional experience is that changes get more expensive with time, CCPDS-R demonstrates that the cost per change improves (at least stablizes) with time. This is consistent with the goals of an evolutionary development approach [12] and the promises of an object oriented architecture [13] where the early investment in the foundation components and high risk components pays off in the remainder of the life cycle with increased ease of change.

Activity Partition The actual breakdown of labor by major activity is defined in Table IX. This breakdown provides an interesting perspective of both the ramifications of an evolutionary process model and the effects of Ada on the relative activity partition.

Activity	MM
Software management and Planning	97
Software Requirements	201
Software Engineering	215
Architecture Design	(108)
Metrics	(20)
Tools	(41)
Demos	(46)
Software Development	934
Design	
Implementation	
Standalone Test	
Documentation	
Software Testing	346
Turnover Integration	(110)
Formal String Testing	(163)
FQT	(73)
Software Configuration Management	124
Software Quality Assurance	85
Other (Tech Support, etc.)	25
Total	2027

Table IX: ECC Activity Partition

Some of the important conclusions that can be drawn from the above data are:

(a) While the data shows an explicit requirements analysis activity, in the evolutionary model followed by CCPDS-R, requirements analysis is performed almost continuously in all activities.

(b) Even though the data shows an explicit metrics activity, metrics effort was the foundation of the software management activities and was also supported by middle level management in all areas.

(c) The level of integration activity is perhaps the biggest difference between this data and that experienced in conventional projects. The demonstration activity listed in software engineering represents the initial design integration activities performed in preparation for the three major design reviews. In the opinion of CCPDS-R management, this activity had the largest return on investment in managing the critical software risks.

(d) The subordinate activities to software development are not broken out since the differentiation between these activities iunder the Ada Process Model employed is not well defined. As a generality, these activities experienced approximately a 40% design, 20% implementation and 40% test breakout.

4.2 Cost Analysis For C++ (Estimates)

This section describes a cost estimate for the ECC subsystem based on the use of the C++ COCOMO model presented earlier. This estimate uses the Ada COCOMO description of the actual costs as a baseline to ensure maximal comparability. All language independent parameters have maintained their calibrated values in this estimate, only perturbations which result from language differences, support environment differences and the C++ state of the practice have been made. It should be noted that to remain entirely comparable, parameters which are time dependent (such as the language experience available at the time of project start, i.e., LEXP) have been set to reflect the state of that parameter in the June 1987 timeframe. In general, we have probably ranked C++ more from the perspective of today's maturity (which results in a positive C++ bias) since there are limited historical results.

Table X identifies the relevant parameterization for the C++ estimate of the CCPDS-R ECC subsystem development costs. The items in the table which represent differences from the Ada actuals are further clarified below:

DEM	CSCI NAS	CSCI FRM	CSCI APPs	Description
RELY	5	5	4.91	The values in this table identify the settings for each parameter:
DATA	3	3	3.36	
TIME	4.60	3.10	3.82	
STOR	3	3	3	1=Very Low
VMVH	4	3.60	2.93	2=Low
TURN	1.50	1.50	1.50	3=Nominal
ACAP	4.38	3.33	3.09	4=High
AEXP	2.11	1.72	2.46	5=Very High
PCAP	3.75	3.07	2.93	
VEXP	2.40	2	2.47	Some settings are decimal values
LEXP	1.65	1.37	1.02	to reflect a weight averaged
MODP	4	4	4	(by ESLOC) mixture of settings
TOOL	4	4	4	at a lower (CSC) level.
SCED	3	3	3	
VMVT	3.50	3.50	3.48	
SECU	4	4	4	
RUSE	4.72	3.19	3.50	
CPLX	4.91	4	4.05	
CSCI EAF	2.14	1.82	1.93	The total impact is the product of each settings relative impact (Table I)
ESLOC	19.5K	67.4K	142.3K	Equivalent Source Lines of code
EAF	1.92			$\frac{(2.14*19.5)+(1.82*67.4)+(1.93*142.3)}{229.2}$
EXP	1.13			PMEX=.03, PDRT=.01, RISK=.01, RVOL=.02
ACT	.05			

Table X: C++ COCOMO Development Effort Multipliers: CCPDS-R ECC

ESLOC In general, the ECC software design was assumed to be conserved in this analysis and the ESLOC were deemed to be equivalent in most functions. The following perturbations were made to account for the differences between Ada and C++ in terms of required ESLOC:

(a) NAS source lines were increased to reflect the difficulty in implementing the Ada NAS design in C++. The CCPDS-R software architecture is based on the NAS object oriented architectural approach, and the NAS design relies heavily on Ada features which are not available in C++ (namely generics, pre-emptive tasking and exception handling). To provide the best comparison, we assumed that the NAS design could be conserved (albeit, a very risky development) with an increase in ESLOC and an increased effort adjustment factor.

(b) Given our assumption that the NAS design could be conserved, there were no changes in the framework SLOC.

(c) The applications components which implement the display design and the database design were substantially reduced to account for a more object oriented C++ implementation which could be achieved with ESLOC.

The net change for ESLOC was a reduction of approximately 18K ESLOC for the C++ implementation. This result is quite biased by the conservation of NAS assumption. Without this assumption the ESLOC would have risen by 30-60K.

DEMs As prescribed in Table V the C++ DEMs were adjusted in the following way:

RELY Unchanged from Ada settings.

VMVH,VMVT Slightly reduced (.5) most settings to reflect increased volatility in C++ platforms compared to Ada.

TIME Reduced most settings one full setting to reflect increased performance of C++ implementations compared to Ada.

TURN Slightly reduced (.5) most settings to reflect increased responsiveness of C++ environments compared to Ada.

ACAP Unchanged from Ada settings.

PCAP Unchanged from Ada settings.

LEXP Reduced most settings moderately (1 full setting) to reflect the lack of a C++ skill base available at the time of award within TRW (this reflects a global situation in defense contractors, not just TRW).

MODP Unchanged from Ada settings.

TOOL Reduced most settings by 1 reflecting the lack of environment support across the project for C++ compared to the VAX Ada environment (and supporting TRW Ada development tools). Reduced NAS by 2 full settings reflecting the loss of the capabilities available from the Rational R1000 environment which was used for NAS development.

SCED Unchanged from Ada settings.

SECU Unchanged from Ada settings.

RUSE Unchanged from Ada settings.

CPLX Unchanged from Ada settings.

EAF The increases in the EAF are primarily a result of the reduced maturity, experience, and supporting environment for C++ as well as the increased effort required to achieve reliablility (RELY) and software engineering discipline (MODP).

EXP Except for the underlying modification to the base exponent (i.e., the 1.06 constant instead of Ada COCOMO's 1.04), all other process and risk assessments were left identical to the settings in the ECC actuals. This is probably somewhat positively biased towards C++ since the NAS implementation in C++ would be inherently higher risk. Nevertheless, this would have been determined prior to contract award (as it was for Ada) and is not justifiable without further engineering/prototying.

Manmonths/Productivity The ECC estimate using C++ COCOMO is 2773 manmonths, or 37% more than that required in Ada. The resulting productivity is 92 ESLOC/MM or 50% less than that experienced with Ada (more effort, less lines of code). Even though the C++ implementation was estimated with fewer ESLOC, the language differences in maturity, environment support, reliability and support for large scale development result in a substantially higher cost.

Maintenance-ACT Since COCOMO maintenance estimates are computed as a function of the product of the ACT (25% higher for C++) and the development effort (37% higher for C++), it would estimate the maintenance costs to be (1.25 * 1.37 =1.71) 71% higher for the C++ version of ECC. In rough terms, this would correspond to a difference of 12 full

time maintenance engineers required for the C++ version to only 7 required for the Ada version.

4.3 CCPDS-R 1994: Ada vs C++

This section provides a speculative view of the CCPDS-R ECC subsystem lifecycle costs if it were to be performed in 1994 when both C++ and Ada have more similar support tools and experience bases. This cost analysis simply applies the conclusions of CDRL 002 to modify the development effort multipliers as would be expected during that timeframe. The result of this exercise provides a reflection of the absolute cost differences predicted by the two derived COCOMO models while normalizing the effects of relative maturity. Table XI identifies the modifications *(in italics)* for Ada input parameters that were used in this analysis and input into the Ada model. Table XII identifies the modifications *(in italics)* for C++ parameters that were used in this analysis and input into the C++ model. These two cost analyses can be examined with the PCOC cost model. They are provided on the diskette under the file names: ECCADA94.DAT and ECCCPP94.DAT.

The following paragraphs describe the relative differences from the two previous sets of inputs:

VMVH Modified all settings to reflect low volatility.

VMVT Modified all settings to reflect nominal volatility.

LEXP Modified all settings to reflect nominal language experience.

PMEX Reduced PMEX by .01 to reflect added experience with a modern evolutionary process model.

TOOL Modified all settings to reflect an extra high level of program support environment.

TURN Modified all settings to reflect increased responsiveness of both development environments on advanced personal workstations.

The above speculation assumes a rapid maturation of the C++ technology base. Under these assumptions, the results of the two cost models reflect a significant convergence of the development cost differential (Ada requires 1312 manmonths compared to C++ 1469 manmonths, i.e., about 10% less) but still a fairly significant advantage in Ada maintenance costs (about 30%

less). Both ACTs were left the same (Ada 25% better than C++) for the purposes of maintenance cost comparison since the results of the tradeoff reflect a longstanding advantage in Ada maintenance.

	CSCI			
DEM	NAS	FRM	APPs	Description
RELY	5	5	4.91	The values in this table identify the settings for each parameter:
DATA	3	3	3.36	
TIME	4.89	3.27	4.55	
STOR	3	3	3	1=Very Low
VMVH	*(2)*	*(2)*	*(2)*	2=Low
TURN	*(1)*	*(1)*	*(1)*	3=Nominal
ACAP	4.38	3.33	3.09	4=High
AEXP	2.11	1.72	2.46	5=Very High
PCAP	3.75	3.07	2.93	
VEXP	2.40	2	2.47	Some settings are decimal values
LEXP	*(3)*	*(3)*	*(3)*	to reflect a weight averaged
MODP	4	4	4	(by ESLOC) mixture of settings
TOOL	6	*(6)*	*(6)*	at a lower (CSC) level.
SCED	3	3	3	
VMVT	*(3)*	*(3)*	*(3)*	
SECU	4	4	4	
RUSE	4.72	3.19	3.50	
CPLX	4.91	4	4.04	
CSCI EAF	*(.98)*	*(.88)*	*(1.02)*	The total impact is the product of each settings relative impact (Table I)
ESLOC	14.9K	67.4K	166.1K	Equivalent Source Lines of code
EAF	*(.98)*			$\frac{(.98*14.9)+(.88*67.4)+(1.02*166.1)}{248.4}$
EXP	*(1.10)*			PMEX=.02, PDRT=.01, RISK=.01, RVOL=.02
ACT	.04			

Table XI: Ada COCOMO DEMs for CCPDS-R ECC: 1994

Would all projects experience this Ada advantage? Not necessarily. Clearly, these results show that a development intensive project (such as CCPDS-R) would experience reduced lifecycle costs in Ada. Our tradeoffs would suggest that a development intensive MIS application would also. But there are project domains where C++ would be advantageous. Specifically, projects

which are dominated by user interface software and database management functions may be implementable in far fewer SLOC so that a C++ implementation would be advantageous in the 1994 timeframe. Similarly, an application which is dominated by COTS integration or C conversion would also result in a reduced C++ development cost. One important note: conversions without appropriate redesign are usually just as unmaintainable as the original product (again, maintainability is more a function of design than it is one of language). Furthermore, COTS integration projects have historically been underestimated, so careful risk management is prudent.

	CSCI			
DEM	NAS	FRM	APPs	Description
RELY	5	5	4.91	The values in this table identify the settings for each parameter:
DATA	3	3	3.36	
TIME	4.60	3.10	3.82	
STOR	3	3	3	1=Very Low
VMVH	*(2)*	*(2)*	*(2)*	2=Low
TURN	*(1)*	*(1)*	*(1)*	3=Nominal
ACAP	4.38	3.33	3.09	4=High
AEXP	2.11	1.72	2.46	5=Very High
PCAP	3.75	3.07	2.93	
VEXP	2.40	2	2.47	Some settings are decimal values
LEXP	*(3)*	*(3)*	*(3)*	to reflect a weight averaged
MODP	4	4	4	(by ESLOC) mixture of settings
TOOL	*(6)*	*(6)*	*(6)*	at a lower (CSC) level.
SCED	3	3	3	
VMVT	*(3)*	*(3)*	*(3)*	
SECU	4	4	4	
RUSE	4.72	3.19	3.50	
CPLX	4.94	4	4.05	
CSCI EAF	*(1.17)*	*(1.0)*	*(1.09)*	The total impact is the product of each settings relative impact (Table I)
ESLOC	19.5K	67.4K	142.3K	Equivalent Source Lines of code
EAF	*(1.07)*			$\frac{(1.17*19.5)+(1.0*67.4)+(1.09*142.3)}{229.2}$
EXP	*(1.12)*			PMEX=.02, PDRT=.01, RISK=.01, RVOL=.02
ACT	.05			

Table XII: C++ COCOMO DEMs for CCPDS-R ECC: 1944

5. Conclusions

In general, our analysis and evaluation results in the following conclusion:

Both Ada and C++ represent improved vehicles for software engineering of higher quality products. Currently, C++ is approximately 3 years behind Ada in its maturity, tool support and precedent experience. In the near term, this fact alone provides development cost advantages for Ada on the order of 35% and maintenance cost advantages for Ada on the order of 70%. In the far term (1993+), this Ada advantage would erode to approximately a 10% in development costs and 30% in maintenance costs for a typical development intensive system.

The general form of our derived Ada and C++ cost models are very similar in that they primarily reflect advances in the development process and modern programming practices, approaches which are generally independent of language. There are subtle underlying differences between the two models which favor Ada (more inherent reliability, language enforced configuration management) and some that favor C++ (generally better abstraction support and complexity control, generally better compile time and runtime efficiency).

The primary strengths of Ada are in its support for realtime domains and large scale program development. Its primary weaknesses are its compiletime and runtime efficiency. The primary strengths of C++ are its support for better object oriented design, COTS integration and its compiletime and runtime efficiency. Its main weaknesses are its support for reliability and large scale program development. In general, Ada's weaknesses are solved by the ever-increasing hardware performance and compiler technology advancement. C++ weaknesses must be solved by advances in its support environment, a solution which will take a few years to resolve.

The conclusions drawn in this document are consistent with other sources of C++ and Ada comparisons. Specifically, Reifer Consultants provided us with some statistics from their historical data base. This data (acquired verbally through D. Reifer) is provided in Table XIII.

Criteria	Ada	C++
Number of Projects	150+	23
Average Size	200,000	100,000
Average Productivity	1.15*X	X
Average Error Rate (I&T)	X	1.38*X
Average Error Rate (FQT)	X	3*X

Table XIII: Soft Cost Database Statistics

In the table, we have avoided the use of absolute values for productivity and error rates since their definition in SoftCost is different than that of COCOMO and those presented within this report. For example, SoftCost has a different definition of a SLOC, uses a 160 Hour man month (COCOMO uses 152), includes requirements analysis in its scope, but excludes formal CM and QA (COCOMO excludes requirements, but include CM and QA). The lack of standards across the various software cost models and DoD contractors makes comparisons of productivity and cost data very error prone, consequently we present only relative differences, which are accurate for the purposes of this discussion.

The fact that the Ada projects achieved 15% higher productivity with increased quality and double the average size is consistent with our analysis. Extrapolating these numbers to comparable size projects would result in an expected productivity improvement of about 35%. It is important to note that in comparing this data, the C++ projects are dominated by commercial applications (only 3 are DoD) and the Ada projects are 75% DoD. The reduced requirements (of documentation, bureaucracy, reliability and maintainability) are also important productivity drivers. In spite of these positive biases in the C++ data, SoftCost data reflects our conclusion that Ada should be the near term language of choice and that C++ still needs significant maturing before it is a low risk solution for a large DoD application.

APPENDIX E. Excerpts (Naval Postgraduate School (NPS) Ada Policy Issue Study)

This appendix contains the following excerpts of the Naval Postgraduate School Ada Policy Issues Study Report (Emery and McCaffrey, 1991): sections I through VII and appendices A3 and A4. The full Report contains four additional appendices and an extensive bibliography consisting not only of the customary citations but also of copies of many referenced works in their entirety.

ADA AND MANAGEMENT INFORMATION SYSTEMS:

Policy Issues Concerning Programming Language Options for the Department of Defense

Department of Administrative Sciences
Naval Postgraduate School
Monterey, California

June 1991

ADA AND MANAGEMENT INFORMATION SYSTEMS:

Policy Issues Concerning Programming Language Options for the Department of Defense

Prepared for:

Director of Defense Information
Office of the Assistant Secretary of Defense
Command, Control, Communications and Intelligence
Washington, DC 20301-3040

James C. Emery
Martin J. McCaffrey
Co-Investigators

Tarek Abdel-Hamid
Tung X. Bui
Magdi Kamel
Kishore Sengupta
Dani Zweig
Faculty Contributors

Naval Postgraduate School
Department of Administrative Sciences
Information Technology Management
Monterey, California 93943-5000

ACKNOWLEDGEMENTS

We wish to thank our faculty colleagues in the Administrative Sciences Department at the Naval Postgraduate School who contributed so heavily to this report. Despite short notice and a hurried project schedule, they were able to devote substantial time to the interviewing and research that went into the preparation of the report. Each faculty member contributed to the overall report, but also focused on an important technical aspect of the study. The faculty and their primary area of investigation are as follows:

Prof. Tarek Abdel-Hamid	Cost-effectiveness issues
Prof. Tung X. Bui	AI/DSS and end-user computing
Prof. Magdi Kamel	Database management systems
Prof. Kishore Sengupta	STANFINS-R case study
Prof. Dani Zweig	Code reuse

LCDR Jeff Schweiger provided first-rate technical and administrative support for the project; he was especially helpful in compiling the bibliography that accompanies this report. Essential secretarial assistance was given by Rose Mendoza and Eva Long. Prof. David Whipple, Chairman of the Administrative Sciences Department, was a source of valuable support and encouragement. To Jan Evans, Support Staff Supervisor of the Department, fell the unenviable burden of dealing with the myriad administrative and fiscal tasks encountered during the course of the project.

We are very grateful for essential assistance given to us by numerous persons in the Office of the Secretary of Defense and other DoD services and agencies, industry representatives, and members of academic institutions. Without exception, they gave us their enthusiastic support and time. Several of them were generous enough to attend a review meeting at the Naval Postgraduate School. Whatever faults the final report may have - for which we and our participating faculty colleagues take full responsibility, of course - the results have been immeasurably improved by their help.

Prof. James C. Emery
Prof. Martin J. McCaffrey
Co-Principal Investigators

TABLE OF CONTENTS

*This Table of Contents represent the contents of the full Study Report. Excerpts of the Study Report are contained herein.

TABLE OF CONTENTS (CONT'D)

TABLE OF CONTENTS (CONT'D)

I. MANAGEMENT SUMMARY

Management information systems (MIS) are gaining an increasingly important role in the activities of the Department of Defense (DoD). As a result, the effective implementation of these systems has become an important issue within DoD. Success in developing an MIS requires getting a lot of things right: sound information systems planning, a supporting infrastructure that facilitates the integration of an application with other parts of the system, and a powerful software development environment.

The choice or programming language for MIS applications is an important aspect of any effort aimed at improving the software development process. The programming language Ada has earned a solid record within DoD as a valuable contributor to advancing software engineering practices. Congress has now mandated the use of Ada, "where cost effective," for all new applications, including MIS. In order to administer this mandate, DoD must establish policies and guidelines for defining the circumstances under which a programming language other than Ada may be used.

This report is intended to provide objective background information for formulating Ada waiver policies. Some of the positions taken are likely to be controversial. Our conclusions and recommendations are based on our best professional judgements, made after considerable discussion with interested parties and a review of the available evidence.

Almost everyone agrees that current implementation practices for MIS do not work very well. By far the most serious barriers to progress come from software, not hardware. Many of the existing software problems - high development and maintenance costs, delayed delivery schedules, failure to meet real user needs, and inflexibility - stem in major part from the intrinsic limitations of the conventional implementation process. A serious attack on the growing "software crisis" must bring about basic changes in the way systems are built and maintained.

We are beginning to see some major advances in software development technology. The formulation of sound Ada policies must take account of these advances that have emerged since Ada was introduced about a decade ago. The substitution of Ada for other third-generation

languages (3GLs) will not ease the software crisis very much unless accompanied by other fundamental changes.

A contemporary MIS depends heavily on a massive shared database; a productive development environment must therefore provide the means to link the MIS to a full-featured database management system. It must similarly provide productive ways to manage a communications network, build effective user interfaces, generate a variety of tailored reports, and deal with the sundry tasks commonly associated with an MIS. Ada's attractiveness for MIS applications can be significantly increased through the support of state-of-the-art computer-assisted software engineering (CASE) tools and a comprehensive set of reusable Ada packages for performing common MIS functions. DoD should therefore foster the development and deployment of such an environment.

It cannot be stressed too much that the potential gains from productive tools will not be realized without competent professionals trained in sound software engineering practices. The availability of good people using good practices dominates all other sources of improvements. Trained personnel are in short supply, due to part to DoD's lack of commitment and resources devoted to strong educational programs in software engineering. Management practices, too, must be improved to motivate and coordinate team members.

It may be possible to achieve an even greater improvement - perhaps as much as a tenfold increase in productivity under the right conditions - through the use of one of the integrated "fourth-generation languages" (4GLs) now appearing on the market. With proper management, the productivity gains achievable during an application's initial development can continue throughout its entire maintenance life. Relatively little experience exists to support a major commitment to an integrated 4GL, but the evidence is already favorable enough to justify a serious exploratory effort to apply 4GL technology within DoD. Delaying action until iron-clad evidence is at hand would take several years, at the cost of further compounding the software crisis.

In order to achieve dramatic productivity gains, a 4GL must enable the developer to define the complete design of an application entirely in the 4GL's own high-level specification language. The specification is then translated automatically by the 4GL into an executable program. Any change to the application, during either its initial development or maintenance phase, is made in the specification language, which then gets re-translated into a running program. For various technical reasons, 4GLs suitable for heavy-duty use generally translate the high-level specification into an

intermediate procedural language. It appears that at least some of the new 4GLs will be able to generate Ada code, thus increasing the portability of the application.

A highly productive development environment goes a long way toward correcting the deficiencies in many current information systems. The advantage is not simply a matter of reducing the time and cost of developing and maintaining a system; it can also significantly enhance the effectiveness of an MIS. A productive development process makes it feasible to adapt the MIS to changing needs during the course of implementing and operating the system. It is only through such adaptation and continual enhancements that an MIS can be made truly effective.

DoD policies should encourage the use of a productive software development environment. Traditional methodologies should be discouraged, whether or not the programming is done in Ada. DoD should pursue an active program for selecting and applying the best CASE tools and 4GLs. Present development and documentation standards should be examined carefully, with the view of adapting them to the power and flexibility of the new development tools. Personnel policies - hiring, training, compensation, and career development - should be modified to meed DoD's increased need for talented, well-trained, and motivated technical personnel.

The transition to a new development environment raises some difficult management issues. Despite its high potential payoff, the move is not without risk. The greatest danger is that the shift to new tools will not be accompanied by other necessary changes in the development methodology and the composition of development teams. Failure to make the necessary adjustments would almost certainly lead to disappointing results. Use of a variety of CASE tools and 4GLs also creates some new problems of training and long-term support. If properly managed, though, risks and complexity can be kept to manageable proportions. The potential benefits more than justify taking on some prudent risks and management challenges.

DoD must establish policies for dealing with end-user computing. Ada was designed as a powerful general-purpose procedural language, and as such it is not suitable as an end-user language with which a typical user could program his or her own application. Most end-user computing will require the use of a user-friendly language aimed at a specialized problem domain - a spreadsheet language for simple analytical or presentation tasks, say, or a database management tool for small administrative applications developed on a microcomputer or workstation. This approach meets the Congressional mandate, because it often provides by far the most responsive and cost-effective means of satisfying the myriad small computing needs that crop up throughout DoD. When end-user computing is properly managed and supported, the risks of duplication,

unverified outputs, and unmaintainable code can be controlled. Excessive constraints on such computing would effectively deny many users the benefits of supportive applications of information technology.

Similar issues arise with the use of special-purpose languages for solving particular classes of problems. Decision support systems and expert systems, for example, are almost always programmed - for good reasons - using one of the special-purpose languages aimed at these problem domains. Ada policies should recognize the legitimacy of this form of computing.

Failure to capture the enormous productivity gains offered by modern software technology carries its own risk - the far greater, but perhaps less obvious, risk of continuing on the same path that has led to today's software crisis. DoD cannot meet its software needs for effective MIS without making fundamental changes in the way software is developed and maintained.

With this objective in mind, we recommend DoD take the initiatives of actions given below. A more detailed discussion of these recommendations is presented in Section VII.

- *DoD should move as rapidly as feasible to high-productivity development environments for the implementation of management information systems.*

  - DoD should require the use of an integrated 4GL or state-of-the-art CASE tools for new MIS applications.
  - DoD should establish guidelines for dealing with the introduction of 4GLs.
  - Development of human resources for the implementation of application software should be given a high priority.
  - DoD should revise its software development methodologies, documentation standards, and acquisition policies to make them consistent with a more adaptive and interactive implementation process.
  - DoD should launch a demonstration project to implement a major application (perhaps under the Corporate Information Management initiative) using an integrated 4GL.
  - DoD should initiate a second demonstration project using an Ada environment supported by state-of-the-art CASE tools and design practices.

- *Consistent with the importance attached to Ada, DoD should devote greater effort and funding to support various Ada activities (e.g., Ada Technology Improvement Program, Ada 9X, and Ada education initiatives).*

- *DoD should establish policies and guidelines for the use of programming languages other than Ada for end-user computing and applications for which an established domain-specific language is available.*

  - End-user computing (within appropriately defined boundaries) should be exempted from having to submit a waiver request if not using Ada.
  - Applications for which a well-established domain-specific language is available should normally be granted a waiver when a documented request is submitted.

- *DoD should establish continuing mechanisms for promoting the best software development practices and encouraging software reuse.*

II. BACKGROUND ISSUES

In the 1991 Defense Appropriations Bill, Congress mandated the use of Ada as the standard computer programming language for all new software applications developed within the Department of Defense, "where cost effective." The mandating legislation thus contemplates that a waiver from the use of Ada can be granted in some circumstances. The purpose of this study is to provide a reasoned, objective discussion of the important issues and principles connected with the formulation of DoD waiver policies for the implementation of management information systems (MIS). This section discusses important background issues that bear on setting DoD computing strategies.

A. The Critical Role of Information Technology in DoD

We are rapidly moving from an era of information scarcity to one of abundance. Until very recently, information processing was expensive, time consuming, and error prone; in comparison, it is now cheap, fast, and reliable. Because of these changes, the use of information processing often offers the most attractive means for the Department of Defense to increase its military effectiveness and reduce its use of capital resources and personnel.

The importance of information technology is widely recognized within DoD. This is manifested in the growing proliferation and power of "smart" weapons and sophisticated command and control systems that proved their worth so dramatically in Operation Desert Storm. Information technology is similarly essential for administrative applications. Any fundamental improvements in the efficiency and effectiveness of managing the military services - in logistics, human resource management, financial control, and the like - will almost certainly require the effective use of computer-based systems. DoD has recently committed itself to a major cost reduction program that relies heavily on improved management through the use of management information systems.

B. Problems in Applying Information Technology

The exploitation of information technology within DoD has not been without problems. Requirements from users for additional functionality and integration have grown at a more rapid rate than can be satisfied by existing software engineering practices. As a result, software now constitutes by far the most serious bottleneck to exploiting information technology. Without major improvements in the process of developing and maintaining software, the "software crisis" can only grow worse.

This report is concerned with *management information systems,* or MIS. (The term *automated information systems,* AIS, is also used within DoD, and has essentially the same meaning as MIS). An MIS deals with administrative matters in such areas as logistics, procurement, personnel, and financial planning and control. Command, control, communications, and intelligence (C^3I) systems tend to have characteristics that fall between embedded systems and MIS, and therefore many of the principles that apply to MIS also apply to C^3I systems.

The GAO and others have pointed out the problems encountered in developing and maintaining MIS within the Federal government. A high proportion of development projects suffer from one or more of the following problems:

- Applications are not sufficiently linked with the mission and objectives of the organization.
- Applications are fragmented, providing relatively little cross-functional integration or sharing of common resources (e.g., data and communication links).
- Applications are delivered significantly over budget and behind schedule.
- A high rate of abandonment is experienced, where systems never become operational - in some cases after the expenditure of hundreds of millions of dollars.
- When an application finally does get delivered, users are often disappointed when it does not perform as expected.
- High maintenance costs are experienced over the life of an application, often amounting to over twice the cost of the initial development.

In short, existing practices have been unable to deliver the systems users want at the cost they expected to pay. The software backlog continues to swell and legitimate needs go unsatisfied.

C. Trends in MIS Applications

A number of trends in management information systems, both in the private sector and within DoD, are adding to the complexity of the task of delivering and maintaining effective systems. The following MIS trends are among the more important ones:

- Organizations increasingly view their MIS as an integral contributor to their business strategy and as a primary vehicle for implementing improvements in critical operational activities.

- Success in developing an MIS calls for a partnership arrangement between line managers and the technical staff, with line managers assuming greater responsibility for the functional specification of the system and for linking it with business strategy.

- There seem to be few limits to the growth in the functional requirements demanded by users, except for the organization's ability to deliver the supporting software; as a result, mainline MIS applications are growing rapidly in size, with programs in excess of a million lines of code not uncommon.

- Managers increasingly demand that the MIS be designed in such a way that it can adapt to organizational learning and environment changes.

- MIS continue to be converted from batch to interactive systems, although these tend to have relatively modest response time requirements compared to embedded and C^3I systems.

- MIS designers increasingly put the primary design emphasis on a shared database rather than on individual applications; the resulting database typically consists of billions or even trillions of characters scattered across multiple direct access storage devices and off-line files.

- There is a growing interest in distributed and cooperative computing architectures in which much of the processing is performed on powerful but low-cost personal workstations.

- Hardware is declining rapidly as a significant design issue; software issues should almost always dominate design decisions (unless constrained by existing hardware or acquisition regulations).

- A trend exists toward *open systems* using non-proprietary products.

- The complexity of the internal program logic of most current systems is generally not very great, but it seems likely that this will change as applications increasingly incorporate some degree of "intelligence" in the form of embedded decision models and expert systems.

- The development of user interfaces (interactive screens, reports, menus, user dialogues, etc.) accounts for a major share of an application's cost, and has a significant impact on a system's effectiveness, efficiency, and degree of user satisfaction.

- Low programmer productivity, both in the initial development and throughout the continuing maintenance cycle of an application, constitutes a major impediment to the successful use of management information systems.

Most of these trends apply to embedded systems as well as MIS, but often to different degrees. For example, the management of a massive database typically plays a more prominent role for an MIS than it does for an embedded system.

D. Need to Improve Management Information Systems

It is widely recognized that major improvements are required in the process for developing management information systems. It is important to keep in perspective that the choice of a programming language is only one part of an organization's overall approach to deploying information systems (IS). Major improvements in DoD's management information systems have to come from an attack on four fronts: improve information systems planning, build a sound enabling infrastructure, improve the software development process, and reduce software maintenance problems.

*1. **Improving information systems planning.*** IS strategic planning is concerned with establishing a broad blueprint for using information technology as a contributor to the goals and missions of an organization. The planning provides a basis for identifying information requirements and setting priorities among competing uses of resources. Without a unified strategy,

there can be no assurance that the disparate parts of the information system will fit together and contribute most effectively to achieving the organization's mission.

*2. **Building a sound enabling infrastructure.*** This IS infrastructure provides the building blocks that permit the efficient development of individual applications. Its principal components are:

- Mechanisms for sharing common data resources.
- A telecommunications network that facilitates interconnection among computers and terminals.
- A support structure for software development (e.g., suitable standards, an effective software engineering environment, and trained development personnel).

Like the road network of a country, an effective infrastructure does not itself generate direct benefits; rather, it facilitates the development of applications that *do* contribute benefits.

*3. **Improving the software development process.*** The ultimate payoff from good high-level planning and a sound infrastructure comes from an organization's ability to implement applications within budget, on time, and with required functional capabilities. It is here that many organizations stumble. Emerging new development methodologies promise to make a major contribution to improving the situation. These are discussed in the next section.

*4. **Reducing software maintenance problems.*** In DoD, as in most large organizations, software maintenance typically consumes two-thirds or more of an organization's systems analysis and programming resources. As a result, DoD is caught in a vicious circle in which available resources are drained away in maintaining old (and often mediocre) software, leaving few resources left over to make fundamental improvements. Only a relatively small portion of this "maintenance" effort is aimed at correcting defects; most of it goes into making relatively minor enhancements stemming from changes in technology or mission needs.

III. THE SOFTWARE IMPLEMENTATION PROCESS

Achieving more successful MIS calls for some major improvements in the software implementation process. This objective can be attacked along two lines: 1) by refining the existing process, or 2) by making fundamental changes in the process. These are not mutually exclusive approaches: there is certainly room for refining the existing methodologies while simultaneously pursuing efforts to make major improvements.

A. The Sequential Development Process

The widely practiced methodologies for developing systems have not changed much since the widespread use of 3GLs back in the Sixties. Since then, methodologies have been tuned and become more disciplined, but without altering the fundamental nature of the development process.

The key idea behind the conventional *systems development life cycle* (SDLC) is a sequential - also called "waterfall" - development process. The process proceeds from stage to stage, with each stage being completed before continuing on to the next one. The typical SDLC begins with a feasibility study, then continues through requirements analysis, gross design, detailed design, programming, test, conversion, and operations (with its accompanying maintenance). Different names may be used for the various stages, but all conventional methodologies approach the implementation task in essentially the same sequential manner.

Various schemes have been developed to manage this process. Elaborate procedures are imposed to document the output of each stage and to verify that the required work has been done. The team composition tends to change with each stage, and so the documentation serves the additional role of coordinating across stages and assigning responsibility for the correctness of the work done at each stage.

To adhere to a rigid sequential process, requirements must be established up front and frozen thereafter; the subsequent stages merely implement these fixed requirements. This may work satisfactorily for well-understood applications in which requirements are not likely to change much during the course of implementation, but overwhelming evidence suggests that few

organizations are able to establish a satisfactory set of stable requirements for an MIS application. A large project may take several years to implement, and by that time the world has changed significantly. The inflexible waterfall process cannot take advantage of the learning and changes that take place during the course of the development. Furthermore, the abstract nature of requirements specification, in which users are asked to "sign off" on voluminous paper documentation, almost guarantees that any system meeting the formal specifications will not meet the real needs of users. The flexibility of the process within DoD is further limited by standards for documentation and milestone reporting that are based heavily on the sequential waterfall process.

In view of these well-recognized problems, it is not surprising that in practice the sequential model is generally violated to one degree or another. Corrections in early decisions are certainly made when critical errors or omissions are discovered downstream. Furthermore, we are now seeing increased use of other development models that recognize the need for greater flexibility than the pure sequential model allows. The "spiral" model, for example, specifically calls for the reexamination of design decisions to incorporate later learning. Prototyping is becoming more widely used as a means of experimenting with alternative designs and presenting users with a concrete and understandable representation of an application. A design arrived at through this process can then be incorporated in formal requirement specifications with the expectation that the system will more closely meet real mission needs.

Unfortunately, the adaptability of an application is often limited by the tools used in its implementation. An application consisting of a million or more lines of code in a third-generation language, for example, usually cannot be modified without considerable effort. The cost of making a correction is generally quite high, and escalates rapidly as the process moves from stage to stage. As a practical matter, all but the most important changes are discouraged in the interest of economy and stability.

The problems of coordination grow rapidly during the programming stage when the size of the team typically expands rapidly in order to cope with this labor-intensive chore. With average productivity ranging from 10 to perhaps 40 lines of code per person-day, the total effort for a million-line program can easily exceed 100 person-years. In order to deliver an application of this magnitude within an acceptable period of time, a large number of programmers must be employed in the effort. Coordination becomes almost impossible for such a project without a closely controlled and relatively stable design specification.

The difficulties of coordinating the efforts of a huge team places an upper limit on the size of an application. To keep the team at a manageable size, a tradeoff must be made between the size of an application and the time needed to deliver it. Current MIS applications seldom exceed three or four million lines of code because of the inability of the conventional development process to cope with larger applications (to say nothing of the cost). As things stand now, software development is likely to place a definite constraint of DoD's ability to meet the increasing demands placed on its MIS applications.

B. Improvements in the Conventional Third-Generation Process

Since the first extensive use of 3GLs in the early Sixties, great effort has been spent in improving software engineering practices. The improvements have come in a number of forms:

- Team organization and coordinating mechanisms.
- Improved modularity of programs based on advances in the theory of program structure.
- Design techniques that emphasize flexibility (e.g., use of a parameter table rather than "hard wired" program constants).
- "Models" for defining processes and data structures.
- Project management techniques with supporting software.
- Methodologies for managing the system development life cycle.
- Language features and design concepts that support object-oriented data abstraction and inheritance.
- Computer-assisted software engineering (CASE) products to automate various tasks associated with the different methodologies (e.g., generating data flow diagrams).
- A variety of software products that relieve the programmer of detailed tasks commonly found in a given problem domain (e.g., for the MIS domain, database management systems, screen formatters, report generators, etc.).

To this list of important efforts to improve software development must be added to the creation of Ada. Ada was designed to incorporate language features that foster good software engineering practices. Its use within DoD is beginning to provide significant payoffs from these efforts.

Without the improvements made in software engineering, organizations could not have tackled the large development projects that have become increasingly common in the commercial and government sectors. Unfortunately, however, demands have grown faster than application

developers can satisfy the needs. As a result, most organizations, including DoD, have found themselves sinking deeper into the software quagmire.

In discussing these problems, it should be noted that it is difficult to separate the intrinsic limitations of 3GL technology from the way it is applied. A highly skilled programmer can achieve excellent productivity using the best current 3GL practices. Such a programmer tends to structure an application into logical, cognitively cohesive components that permit their consistent reuse throughout the design - in effect, creating a "language" quite specific to the application. This calls for a relatively rare ability on the part of the programmer. Actual design practices generally fall far short of this ideal.

C. Fundamental Changes in the Development Process

Knowledgeable observers in the MIS field believe that fundamental changes must be made in the software development process if organizations are to meet the growing demands placed on software. As James Martin, a widely acknowledged authority on information systems, writes, "there is widespread agreement that IS organizations cannot continue to build computer software systems using traditional development techniques."

There is an emerging consensus that a new development *paradigm*, or model, is needed to replace the obsolescent third-generation process. Although the exact nature of the new paradigm is by no means clear, it is quite likely that major improvement requires an attack on the three principal components of any development paradigm: human resources, development methodology, and software development tools.

1. Human resources. A great deal of experimental and anecdotal data suggest that getting the right development people with the right training and motivation is the single most important contributor to software quality and productivity. No improvements in methodology or tools are likely to provide a big payoff unless accompanied by a capable professional staff. Management must pay much more attention than it typically does to such matters as recruitment, training, pay scales, attractive career paths, and the development of a creative and empowering working environment.

2. Development methodology. Existing 3GL methodologies place heavy emphasis on getting up-front requirement specifications right so that expensive changes can be avoided when flaws are revealed in the later SDLC stages. A successful new development paradigm must

recognize the intrinsic impossibility of fixing stable requirements early in the life of a complex development project. The organization learns a great deal from the very act of creating an application. An effective methodology must actively solicit feedback to earlier stages in order to adapt the design to take advantage of organizational learning. The process must foster continual interactions with users throughout the entire life cycle, not just during the early requirements analysis stage. In this way, managers can continually refine their administrative practices as they learn during the course of a system's development and operation.

Interaction with users is greatly facilitated if design alternatives can be presented in the form of a concrete working prototype. The development environment should permit the iterative modification of a series of prototypes that eventually converge to a "final" production version (which thereafter undergoes a continuous process of enhancement during the system's operational phase). An evolutionary methodology of this sort is only feasible if the development environment is productive enough to develop and modify software far more quickly and cheaply than is possible with traditional methodologies.

*3. **Development tools.*** A necessary condition for making significant improvements in the implementation process is the availability of software tools that greatly leverage the efforts of the human developers. Although these tools clearly do not provide a "silver bullet" for eliminating all software problems, they do provide essential support for a talented staff and a sound development methodology.

A really significant improvement requires more than the typical CASE product that automates only a relatively small part of the development process. Instead, an integrated set of tools is needed to support the entire systems design life cycle and provide "seamless" links across stages of the process. A critical component is a *repository* - also called a *dictionary* or *encyclopedia* - that provides a complete, common, authoritative, and consistent description of the enterprise's information systems. Included in the repository are such things as the names of all data elements, the complete structure of the database, processing modules, data flows among applications, screen and report formats, and security authorizations. To be most effective, the repository must be *active* - i.e., any change in the system specification is made through a change in the repository, and then the change is automatically reflected throughout all components of the system. The repository becomes all the more valuable as the central focal point of development and maintenance if the information it contains can be assessed selectively through an interactive retrieval system.

D. High-Productivity Environments

Two alternative approaches can be used to achieve high productivity in the development process. One of these might be called the "3-1/2GL" approach, and the other the 4GL approach.

1. The 3-1/2GL environment. The 3-1/2GL approach takes the best of contemporary tools and puts them together in a highly productive software engineering environment. The primary language within DoD would be Ada, as the 3GL most suited for producing very large, high-quality applications. Aiding the Ada programmer would be an integrated set of software tools that support the full life cycle development process. A heavy emphasis would be placed on sound software engineering practices and the use of the best contemporary design techniques, such as object-oriented design. A comprehensive repository and powerful database management functions would both be indispensable ingredients for the MIS development environment. Sound project management would coordinate activities throughout the life of the project.

The full power of the 3-1/2GL approach comes with the systematic reuse of Ada packages. A well-designed suite of packages can serve as a repository of powerful extensions to Ada, greatly reducing the cost of implementing standard MIS functions. With this approach, Ada statements are used primarily to invoke the execution of the standard functions. For tasks not included in the repertoire of packages, the programmer can provide an Ada procedural specification. Experience suggests that it is a reasonable goal to generate over half of the code from reusable packages.

Such benefits from reuse cannot be realized without a substantial effort to develop reusable software and an infrastructure to support its reuse (Appendix A3). Reuse outside the boundaries of a project or work group is generally difficult to achieve. Merely putting components into an Ada package repository, for example, is not likely to lead to much reuse. In order to achieve widespread sharing, a significant effort would have to go into the design, development, and support of a repository of Ada packages that provide a close fit with the common tasks encountered in MIS Applications. .

Putting such a repository together would be a professionally demanding task - equivalent, in fact, to designing a high-level language for the MIS domain. It is essential that the contents of the repository be carefully catalogued and indexed, so that the effort to locate a reusable component does not exceed the potential savings. Close attention has to be paid to the maintenance and updating of reusable components. The quality and documentation required for a reusable

component are, in general, more demanding than for comparable component developed for a single application at a single site.

2. The 4GL environment. One of the difficulties in discussing fourth-generation languages is that there is no consensus as to what constitutes such a language, nor even the name we should attach for the emerging development tools. Products offered in the market often come under the general label of 4GL, but terms such as *application generator* and *integrated* CASE *(or I- Case) tool* are also used.

Many products sold as 4GLs were not designed for implementing mainline systems of the sort encountered within DoD. A number of such products are well suited to deal with common MIS tasks, such as generating flexible reports or handling ad hoc inquiries. Some of them are even user-friendly enough to be suitable for end-user programming. However, these relatively low-level 4GLs are not powerful or comprehensive enough for the professional staff to use for the development of high volume interactive systems. For such use, one of the powerful new integrated products is required.

In order to achieve maximum productivity using the 4GL approach, the product's language must be capable of specifying a complete application. The specification must then be translated automatically into an equivalent object program (possibly through an intermediate procedural language such as Ada).

Although we use "4GL" to label this new class of development tool, the term needs further definition. This can best be done by listing the capabilities that ideally should be provided by a 4GL of the type needed to support a powerful new development paradigm. They are:

- A language capable of defining the complete specification of a system, which can then be translated automatically into a program for execution on a selected target computer.

- A set of built-in language functions for defining the type of computational tasks that occur frequently in MIS applications, such as creating screen formats for interactive terminals, defining automatic error checks for input data, generating reports or responses to user queries, and designing "user-friendly" interfaces (e.g., a menu structure).

- Language functions that permit terse specification of a computational task, often best achieved through a *nonprocedural* language that allows a programmer to specify *what* task is to be accomplished rather than defining a *how-to-do-it* procedure for doing it.

- Automatic consistency and completeness checking of a design specification.

- Integrated database management tools for managing the system's database.

- An active central repository, with interactive retrieval capabilities that facilitate access to selected information about the entire system.

- Integrated communication functions for controlling a telecommunications network, handling remote terminals, transmitting data to and from other computers, performing error checks on transmitted data, etc.

- Facilities for managing a secure on-line environment, such as those for keeping track of transactions in their various stages of processing, maintaining a journal of all events within the system, and recovering from a system failure.

- Facilities for integrating the new system with its environment (e.g., other existing applications or networks) and keeping track of multiple versions of an application.

- Integrated project management tools for scheduling and coordinating development tasks as defined in the repository.

- A set of design tools with a strong graphical orientation to aid the developer in visualizing relations among system components.

- An assortment of analytical and documentation tools for the support of sound software engineering practices.

- Built-in testing facilities (e.g., for generating simulated test data and managing regression testing).

- Capability of generating sufficiently efficient programs to permit the system to handle a high volume of transactions at a feasible cost.

For use within DoD, an important additional need is for the 4GL translator to generate an intermediate version of an application in Ada.

No product currently on the market satisfies all requirements; each of them suffers from at least one of the following limitations:

- They are proprietary, requiring a relatively long-term commitment to a single vendor.
- They lack the functionality to define a complete system within the 4GL's specification language.
- They are not integrated, making them incapable of linking the various parts of the system.
- They are very expensive in terms of hardware requirements and/or software license fees.
- They are inefficient in the use of machine resources.
- They are immature, without a solid record of successes to lend credibility to the 4GL approach.
- They require a significantly different approach to software design, and may thus require several months for even an experienced developer to gain full knowledge of their capabilities.

The situation is improving rapidly, however. Some powerful 4GLs are already on the market and proving their worth in developing and maintaining a variety of large MIS applications. Several of them are already valid contenders for use within DoD, and new products or enhancements to existing ones are announced frequently.

Experience with existing 4GL applications suggests that tenfold productivity gains are possible under the right circumstances. Furthermore, the gains are likely to carry on throughout the maintenance phase. With the evolutionary approach to application development made feasible by a 4GL, maintenance becomes merely a continuation of the initial process. Many of the features associated with a 4GL provide valuable maintenance capabilities. For example, the availability of a central repository, when combined with interactive retrieval mechanisms, greatly simplifies the

process of understanding and modifying an application. The relatively concise specification language of a 4GL also aids in understanding.

The major drawback to adopting a 4GL is that it requires substantial changes in other aspects of the development and maintenance process if it is to yield substantial payoffs. Merely introducing the new 4GL into a conventional 3GL process will give disappointing results, and in fact could even be destructive.

An effective 4GL process must be much more flexible and adaptive than the conventional sequential approach, and requires continual interaction with users through an application's life cycle. But an evolutionary approach of this sort runs the risk of inviting sloppy up-front requirements analysis. The iterative process may not converge effectively on an acceptable final design if users continue to seek an illusive "perfect" system. The required skill level of the development team is higher than is found in a typical large staff of programmers and analysts.

The risks of moving to a 4GL can be contained with the proper structuring and monitoring of a development project. Whether within the constraints and culture of DoD the necessary adjustments can be made remains an unsettled question. At the very least, the introduction of a 4GL needs to be managed with caution to recognize the risks involved.

IV. THE PROGRAMMING LANGUAGE ADA

Ada was developed in the late Seventies and early Eighties in response to the problems encountered in developing and maintaining massive computer programs within DoD. A broad group of computer scientists contributed to its specifications, and the final design was settled through an international competition. It was designed to incorporate the best features of procedural languages. By focusing on a single standard language, DoD hoped to eliminate many of the monumental problems of supporting a hodgepodge of languages.

A. Characteristics of Ada

For the purpose of this report, it is possible to identify the salient features that give the Ada language its special capabilities:

- A general-purpose third-generation procedural language, suitable for programming applications drawn from a wide class of problem areas.
- Features especially aimed at dealing with time-critical tasks, of the sort found in embedded weapon systems and certain C^3I systems (e.g., managing concurrent tasks and coordinating among them).
- Constructs to support good software engineering practice (e.g., prevention of operations on incompatible data, use of a clear-cut modular structure among program components, and reuse of program code).
- Features that provide strong support for large programs (over 500,000 lines of code, say) having stringent quality requirements and a long operational life.
- Features aimed at providing a professional programmer with a variety of powerful capabilities for dealing with complex applications.

Ada has been successfully taught in introductory programming courses. For example, the Computer Science Department at the Naval Postgraduate School has effectively taught Ada as the first language to a large number of students for a number of years. Everyone agrees, however, that the most difficult part of learning to use Ada lies not in the language itself, but rather in good software engineering principles needed to fully exploit Ada's advantages. As is the case for any programming language, an unskilled programmer can still write sloppy, unstructured Ada code. Ada does not guarantee, but tends to encourage, sound software engineering practices.

B. The Current Status of Ada

1. Signs of success. Standardized in 1983, Ada is now a mature language with robust compilers available for numerous computer platforms. Translation time has been reduced significantly, although the heavy error checking performed by the compiler inevitably adds to the time (but with compensating reductions in debugging time and execution time due to the detections of errors at translation). Execution time appears to be comparable to other efficient languages. A variety of utility programs, CASE products, and training aids are now available to support Ada development.

A number of important successes have been achieved in developing large Ada systems within DoD. Many of these applications have been in the embedded system domain, where Ada has its clearest comparative advantage. A few large MIS-type applications have been delivered or are in development - most notably, the Army's STANFINS-R system - with apparently satisfactory results (Appendix A1).

Solid evidence exists that Ada has largely achieved its goal of providing a first-rate development environment for very large systems. Ada's successes are due to an enforced standard for the language, as well as features that promote portability. In a number of cases, subsystems have been developed by separate groups and have then been integrated successfully with many fewer problems than encountered with other languages. Ada programs have been ported to different hardware platforms with little or no difficulties. Although hard comparative data are scarce, Ada's productivity appears to be competitive with other 3GLs.

There have been relatively few applications of Ada outside of DoD, but here too Ada has demonstrated its value. Most of these have been in embedded applications (control of a printer, for example) or in C^3I-like applications (e.g., FAA's air traffic control system or Bofors' shipboard fire control system). There have also been notable successes in the development of MIS

applications, both in the United States and in other countries (e.g., Wells Fargo in the U.S., Nokia Group's major banking system in Finland, and NTT's broad range of applications in Japan).

*2. **Lack of widespread use of Ada.*** Ada's relatively sparse use in large commercial applications reduces its value in the MIS domain. A limited Ada market not only restricts investments in support software, but it also reduces the availability of Ada-related training materials, textbooks, and educational programs. This lack of interest feeds on itself, because limited training capabilities reduces the supply of skilled Ada programmers, which in turn inhibits the use of Ada.

Even within DoD, the number of MIS applications has been quite limited. Although the potential size of the DoD market is very large in absolute terms, it still constitutes a small part of the worldwide information technology market. If the use of Ada continues to be confined largely to DoD, commercial vendors of third-party software products will invest their resources in other markets presenting more attractive commercial prospects. This would put Ada at a growing disadvantage compared to more popular languages that offer a rich choice of supporting products. That would be unfortunate, both for the DoD (because it would limit the number of Ada-related products) and for industry (because it would limit the private sector's use of a genuinely valuable language for important classes or problems, such as process control applications).

It is important to understand why Ada has experienced such limited use for non-mandated MIS applications, because the reasons may have a bearing on setting DoD policies. Ada has reputation - deserved or not - as a complex language aimed at time-critical applications, which for many commercial organizations is not a pressing need. The huge existing investment in COBOL and FORTRAN programs, as well as the supporting development tools and pool of trained personnel, make the cost of transition to Ada a non-trivial matter for most commercial organizations. Consequently, few of them have felt that the transition to Ada would provide an acceptable return on the required investment. The benefits of the switch are viewed by many as risky, intangible, and long-term - serious impediments in a commercial environment that looks primarily to next quarter's reported earnings.

A more intrinsic limitation to Ada's use in MIS applications is its procedural character that requires many lines of codes to perform common functions found in an MIS application. Leading organizations increasingly recognize that it makes no sense to develop large applications through traditional labor-intensive methods. Although a well-managed Ada environment offers the prospect of a substantial improvement in productivity over conventional development methods, a

move to Ada might merely delay the still more dramatic gains potentially achievable with an integrated 4GL.

Management of a database is a critical component of almost all MIS applications. Interfacing Ada programs with commercial database systems is not a difficult problem; a large number of database manufacturers have proprietary Ada interfaces to access their databases. The problem is specifying a portable Ada interface to an industry standard.

The Structured Query Language (SQL) is an ANSI and DoD standard for accessing relational databases, and most database vendors have incorpo.ated an SQL interface to their databases. SQL is, however, incompatible with Ada. The challenge becomes one of specifying an Ada-to-SQL interface that is acceptable to both the SQL and the Ada communities. Since Ada is a third-generation procedural language, while SQL is a non-procedural data access language, several varied and complex technical issues need to be addressed in designing an Ada-to-SQL interface. Three major approaches have been proposed for binding Ada to SQL, but each as some drawbacks (Appendix A2). However, the SEI SQL Ada Module Extension (SAME) approach is emerging as the preferred one, as is currently undergoing ISO standardization.

The 1983 version of Ada lacks some features felt to be useful in the MIS world. The Ada 9X project is considering these needs, and may be able to provide a satisfactory solution for many of them. Among the additional features most relevant for MIS applications are those dealing with decimal arithmetic, an expanded character set, more flexible data access that facilitates sharing files and interfacing with existing applications, support for the use of Ada subprograms within programs written in other languages, support for object-oriented programming, and the elimination of subtle platform dependencies. These will provide useful enhancements to Ada, but the current lack of the features is a fairly minor issue and should not be used as a justification for delaying the use of Ada until the changes have been fully implemented (probably no sooner than the latter half of the decade).

C. Impediments to Increased Use of Ada in DoD

1. DoD support for Ada eduction. Despite some notable successes in a few institutions, Ada has not been widely adopted in computer science curricula. According to one faculty member who has monitored Ada's use in academia, there are at most 50 educational institutions that have expanded the use of Ada beyond a mere language course, out of hundreds of

computer science undergraduate programs nationwide. There are a number or reasons for the limited adoption of Ada:

- Low demand for Ada from the private sector.
- Costly compilers and tools.
- Ada's advantages for developing large computer programs are difficult to illustrate and exploit in a one-semester college course, in which students can develop programs of only a few thousand lines of code.
- Inertia in adopting new material or concepts.
- Viewed as a DoD language for specialized weapon systems.

DoD needs to review the role it might play in furthering the expansion of Ada. One of the most fruitful steps might be assistance in providing inexpensive compilers and tools for universities. In contrast to Ada, UNIX and C tools have been made widely available to academia, accounting in major part for their great popularity. Another useful step would be to support the development and dissemination of first-rate courses and instructional materials.

The Ada industry is still relatively young and without deep financial pockets. A few companies have offered to colleges and universities free compilers and development environments, charging only a yearly maintenance fee. Unfortunately, response from academia has not been overwhelming. At this point it appears that additional DoD support is required if we are to overcome academic barriers.

*2. **Under-funding of Ada language improvements.*** Two major Ada language improvement efforts are constrained by a lack of funding. The Ada Technology Improvement Program (ATIP) traditionally has helped the services and agencies overcome technical hurdles in implementing Ada applications. More recently the Ada Joint Program Office (AJPO) has initiated ATIP efforts emphasizing binding of Ada to other languages and developing new software engineering educational programs using Ada. Of over 70 service and agency requests, only 14 were able to be initiated in FY-91 because of funding constraints. The AJPO has had to repeatedly seek outside funding from the services and agencies to make up for the funding shortfall. The Ada 9X program is also constrained in future years.

The FY-92 AJPO budget request was limited to $6.9 million. DoD is estimated to be spending approximately $30 billion on software each year. Thus, for every $1000 we spend on software, we are only willing to spend about 25 cents to support the further enhancement of Ada and its use. This lack of support is inconsistent with the mandated use of the language and the high expectations for the benefits it can provide.

*3. **User pays the incremental cost of implementing Ada.*** One of the frequent reasons cited for the resistance by project managers to use Ada is their perception that their project will have to bear added costs for the transition to Ada. These incremental costs include training, learning time, and the purchase of Ada-related tools. It would be much more palatable for a project manager to use Ada if all or part of the additional costs were borne by higher command.

V. COMPLIANCE WITH THE ADA MANDATE

It is not unreasonable to establish DoD policies that allow varying degrees of dependence on Ada. Any policy must, of course, comply with the Congressional mandate, but the mandate contemplates circumstances under which other languages offer a more cost-effective approach. We will use the following categories of computing in analyzing the issues:

- Programming and maintenance in Ada, with the aid of CASE tools for improving the conventional sequential development process.

- Programming and maintenance in Ada, with the aid of extensive CASE support and a managed set of packages designed to provide extensive support of MIS functions (the 3-1/2GL approach discussed earlier).

- Programming and maintenance in a 4GL, with translation into Ada as as intermediate language.

- Programming and maintenance in a problem-specific language.

- Programming and maintenance in a 3GL other than Ada (e.g., COBOL, FORTRAN, or C++).

We will discuss briefly each of these categories, and the circumstances, if any, under which a given category might be appropriate. It should be pointed out, though, that the boundaries between the categories is sometimes quite fuzzy, and so it may be difficult in practice to judge the category into which a given project falls.

A. Conventional Programming and Maintenance in Ada, Aided by CASE Tools

Every Ada programming project should take advantage of the growing set of CASE tools now available in the market. The emphasis should be on integrated tools that provide seamless

links among the various stages of development. A powerful repository should also be made available.

Although the use of CASE tools offers important advantages, this approach still has some serious limitations. Since no fundamental change is made in the conventional development paradigm, the CASE tools merely automate the sequential SDLC approach, and therefore do relatively little to eliminate the disadvantages of conventional project management.

An important distinction between this category and the next is the absence of a serious management effort to develop a comprehensive set of reusable packages. A significant level of reuse is unlikely to occur unless an explicit program exists for the design, classification, indexing, retrieval, maintenance, and support of reusable packages. Without such reuse, however, development productivity will suffer (Appendix A3).

The use of a CASE tool opens up a problem that Ada was seemingly aimed at reducing. Selecting and supporting a set of CASE products is not a trivial matter. Without establishing a DoD standard for CASE tools - unlikely, at least in the near term - each using organization must confront the problem of supporting its selected products. As CASE tools assume a growing fraction of the burden of application development, problems of proliferating non-standard products will be magnified. The only way to avoid the problem is to standardize on the CASE tools or bar their use - and neither is an attractive solution.

B. The 3-1/2GL Approach

The 3-1/2GL approach is similar to the previous one, but with an important difference. Like the previous category, the 3-1/2GL uses sophisticated CASE tools to increase the efficiency and effectiveness of the development process. Greater emphasis is placed, however, on reducing the extent of new Ada coding. A library of packages is designed explicitly to provide a comprehensive set of functions to support the development of MIS applications. The aim should be automatic code generation of a high proportion of an application. The 3-1/2GL approach probably offers an attainable goal of doubling or even tripling productivity.

There are difficulties with this approach, however. The design of a comprehensive set of packages is likely to be difficult and expensive. It is also a somewhat risky undertaking, because we do not have strong evidence that widespread reuse will be achieved. The use of standard packages might ease the maintenance burden for application developers, but it then shifts the

responsibility to another party who would have to deal with such challenging management issues as keeping track of multiple versions of a package. Furthermore, an application developer who finds it necessary to modify a package in order to tailor it to a user's specific needs would generally have to take on maintenance responsibility for the modified package.

The 3-1/2GL approach also raises some difficult training problems. Developers would have to be trained in the use of the available packages as a central part of the design and implementation process. This would represent a considerable change for most experienced programmers. Training in the CASE tools used to support the 3-1/2GL environment would also be required.

C. Programming and Maintenance in a 4GL, with Translation into Ada

Some of the new 4GL products offer the prospect of developing heavy-duty mainline applications entirely in a high-level specification language. All maintenance changes are similarly effected in the 4GL (followed by re-translation into object code). With proper management, using a well-trained development/maintenance team and an evolutionary implementation methodology, this approach offers the prospect of a tenfold gain in productivity in the initial development and maintenance, with associated improvements in reliability and responsiveness to user needs.

This approach would be more acceptable within DoD if the 4GL translates an application specification into Ada as an intermediate language. None of the strongest 4GL contenders can currently do this, but at least one major vendor has announced its intention to provide such a capability by June 1992. Other vendors may follow, because translation into Ada does not appear to present any insuperable technical problems. This appears to satisfy the formal Ada mandate, and would offer the additional advantage of providing a standard, stable target language portable to a variety of hardware platforms.

A number of barriers to the use of 4GLs must be overcome before this approach is likely to achieve its potential benefits within DoD. A dramatic success in applying the 4GL paradigm calls for some significant changes in the mind set of managers and developers. The development environment should be considerably less rigid and disciplined than is customary in DoD, which may create concerns about risk and loss of control. A certain amount of ambiguity is inevitable with a paradigm that explicitly encourages learning and adaptation during the development process. The focus on a small, capable development team is not a common approach for software development within DoD (although the Department and its contractors have achieved very

favorable results with "swat teams" and "skunk works" that rely on many of the same concepts). Considerable effort would have to be spent in educating the technical staff in the use of 4GL tools and design concepts.

Ambiguity in an application's specification can raise some difficult contractual issues. Writing a traditional development contract with a well-specified final product is impossible in an evolutionary development environment. An adaptive design could probably be handled within existing regulations (such as a time-and-materials contract for the initial prototype development, followed by a fixed-price contract for the "final" version). However, DoD should give considerable attention to its contracting regulations to accommodate a more flexible development process.

Not the least of the problems of adopting the 4GL approach is choosing the particular 4GL to use. All of the contending full-featured 4GLs are proprietary, which opens a number of contracting issues. It is conceivable that DoD could fund the development of a 4GL, but the history of noncommercial projects of this sort do not give much room for optimism. We are hopeful, though, that a 4GL could be selected in open competition in a way that satisfies the requirement for competitive procurement. Even though a product is proprietary, its vendor can publish standards that allow other vendors to interface with it (as Lotus Corporation did, for example, with Lotus 1-2-3).

Reliance on a 4GL for the development and maintenance of applications moves away from one of the Ada goals of having a single standard programming language within DoD. However, as we have seen from the earlier discussion, support of multiple development tools will be required for *any* productive environment. Standardizing on a single 4GL would mitigate the training problem, but Ada experience shows that getting agreement on a standard is a time-consuming process. If a single standard cannot be attained, a specified set of required features for any candidate 4GL could at least narrow the choice considerably and thus reduce selection and support problems. Furthermore, if a 4GL maps into Ada, much of Ada's advantage of stability and portability would still be preserved despite the lack of a standard 4GL.

D. Programming and Maintenance in a Problem-Specific Language

Problem-specific languages cover a very wide spectrum of applications, but each one tends to be aimed at a relatively narrow problem domain (Appendix A5). For example, spreadsheet languages are designed to perform modest-scale numeric and data manipulation tasks and generate

a variety of graphical and tabular outputs. Micro-based application generators (dBase, Paradox, etc.) make it easy to give a work group flexible access to its own local database. Statistical packages focus on handling masses of numeric data and performing a variety of statistical functions. Other special-purpose languages are designed for the development of decision support systems and expert systems. The list of these language domains could go on and on.

A special case of problem-specific languages are *application packages* or *commercial off-the-shelf* (COTS) software. These can be viewed as focused languages that deal with a very narrow application domain, such as payroll, production scheduling, or computer-assisted design. Any tailoring of COTS software for a particular installation would normally be achieved merely be setting appropriate parameter values for the product; actual modifications of the code should generally be avoided. As long as DoD does not have maintenance responsibilities for such a product (as it might if code modifications were required), the use of COTS products is allowed within current or planned guidelines.

Problem-specific languages have achieved great success because they satisfy important needs. Often emphasizing ease of learning and ease of use, they put in the hands of functional specialists (as opposed to professional programmers) the means of developing their own applications to serve their own requirements. Most observers feel that "end-user computing" of this sort is likely to continue to grow as a component of a successful information system strategy.

No one argues that specialized languages do not have a place within DoD; the arguments begin when one tries to set boundaries on their use. A clear-cut case for using a specialized language can be made for a small single-user application developed by the actual end user, as is often the situation for a typical spreadsheet application. Here the advantages of productivity and responsiveness overwhelm any disadvantage of language proliferation. Under these circumstances, a *prima facie* case can be made in favor of the specialized language, because Ada would clearly not be a cost-effective competitor; in fact, it would never be used at all by the typical end user (Appendix A5).

As the size of the user group grows, and the need for technical support expands, the case for end-user computing becomes more controversial. Even commercial organizations thoroughly committed to end-users computing and decentralization have a difficult time formulating policies on these matters. On the one hand, freedom to use the language of one's choice offers the advantage of responsiveness, creativity, and (in most cases) high productivity. On the other hand,

uncontrolled systems development by amateurs raises very serious problems of validity, integration, duplication, and maintainability of the software.

A moderate-size work group should normally be allowed to develop its own unique application in one of the user-friendly end-user languages. DoD would pay a heavy price if Ada policies ruled out, for example, the development of a small database application to keep track of a unit's work assignments. Such an application tends to be quite idiosyncratic and self-contained, and would therefore be an unattractive prospect as a professionally-developed standard application used by many units. The application is not likely to be critical to the success of the unit's mission, and so the penalty of an error or system "crash" would probably be quite modest. A policy that bars the development of such an application would effectively deny to the group many of the advantages of computer-based systems. Here again, the case for use of the specialized language appears to be strong enough that it should not be necessary to justify a non-Ada solution; it can be assumed under these circumstances that Ada cannot meet the Congressional test of cost-effectiveness.

The use of a specialized language should not necessarily be confined to small local work groups. Even a widely used specialized application developed and maintained by a technical staff might best be developed using a domain-specific language. Decision support and expert systems, for example, would generally fall into the class of problems for which a variety of specialized development languages offer advantages sufficient to overwhelm any possible disadvantages. Hardly any knowledgeable professional would suggest that these systems should be developed in a 3GL. Not only would the cost and development time be prohibitive in most cases, but the need for adaptation and evolutionary growth would put a 3GL at great disadvantage. Because of these considerations, DoD policies should certainly allow fairly liberal use of specialized languages designed for specific problem domains.

Unfortunately, it is not easy to formulate a clear-cut set of policies that make it easy to decide whether in a given case Ada should be used. Each case should be considered on its own merits. The decision should be based on such factors as the size and dispersion of an application's prospective user population, the duration of its use, the amount of support required to maintain it, and the risks and costs of a defect or complete failure. The use of Ada should be favored for an application that tends toward a large dispersed user population, a long expected life, complex support requirements, or critical mission dependencies. There are certainly tradeoffs between controlled professional development of an application in Ada and the risks of unfettered

development by user groups, but the resolution of the issue should not always be settled in favor of standardization in Ada.

E. Programming and Maintenance in a 3GL other than Ada

Almost all existing MIS applications within DoD are developed in a 3GL (principally COBOL). Some maintenance will no doubt have to continue using the initial 3GL, but this should be kept to a minimum. The potential tenfold increase in productivity of a 4GL drastically reduces any advantage of preserving an obsolete application, because an entirely new system can then be developed in the 4GL for less than it typically costs to maintain the old one for a year. Preservation of an old program should require justification by comparing its future maintenance costs (over the next five years, say) with the cost of developing and maintaining an entirely new system using the best available environment.

Some interest exists in developing "re-engineering" software that can automatically translate a COBOL program into Ada. These tools are just emerging, however, and are unlikely to have a major impact on the issues in setting Ada policies. Furthermore, re-engineering an obsolete COBOL program is likely to become little more than an obsolete Ada program (although some cleaning up of a badly structured old program can now be done by some of the more "intelligent" re-engineering translators).

In the case of new applications, DoD policy should be quite simple: except in very exceptional cases, in which the cost-effectiveness of a non-Ada solution can be demonstrated convincingly, programming in a third-generation language other than Ada should not be allowed. The arguments still apply that gave rise to the creation of Ada in the first place, and so there should be little reason to compound existing software problems by developing still more 3GL programs in non-Ada languages.

Conceivably, this situation could change if Ada continues to be largely ignored by the commercial sector. The rapid growth in the interest and use of C++, for example, raises this language as a possible future contender. If other languages widely used in the commercial sector continue to grow in power and third-party support relative to Ada, standardization on a single 3GL for MIS may have to be re-examined.

VI. CONCLUSION

This report has examined the use of Ada in programming MIS applications. Ada will (and should) remain for the foreseeable future the DoD's sole standard third-generation language. A major share of software applications within DoD deal with large embedded programs and time-critical C^3I systems, for which Ada is the best choice available. That fact alone should insure that Ada will remain the paramount programming language within DoD.

DoD can strengthen Ada's position even further by providing additional funding for the development of Ada-related educational materials, extensions of the language in the Ada 9X activity, and absorbing some of the transition costs generally incurred when a programming staff moves to Ada from one of the other 3GLs. The development of a sharable Ada repository should also receive greater attention, which might be accomplished either through direct DoD funding or by providing greater contractual incentives for the private sector to take on this task. The relatively meager support provided Ada at the central DoD level is inconsistent with the critical role it must play in the application of information technology.

Despite Ada's undeniable strengths as a 3GL, its use as the sole language for developing large-scale management information systems is open to serious questions. Powerful 4GLs and domain-specific products on the market generally offer a more fruitful approach, because of their potential for achieving very large productivity gains in the development and maintenance of MIS applications. Although standardization must remain an important goal of DoD computing policies, it is not the *only* one; DoD must also consider the tradeoffs between standardization and the increased productivity of the new 4GLs.

The use of a domain-specific language is relatively easy to justify when its capabilities match closely the needs of an application. Many of these languages are mature, with a large user base and pool of experienced developers. For programming end-user applications, such as a modest-sized spreadsheet or local administrative system, the focus of a language should be on ease of learning and ease of use. Ada was never designed to meet these requirements, and therefore can seldom compete with the specialized languages. For larger applications that call for professional development skills, a domain-specific language may still be the appropriate choice even if it is not

especially user friendly. A complex decision support system or AI application, for example, can often be developed much more efficiently with a language designed for this purpose than it can with a general-purpose 3GL like Ada. These applications rely utterly on the adaptability and productivity that a specialized language can provide.

In practice, it is often difficult to draw the appropriate boundaries within which a domain-specific language is the appropriate choice. As an application grows in size, complexity, geographic or organizational dispersion, response-time demands, required reliability, or expected longevity, there comes a point at which it should be taken over by professional developers working in a productive Ada enviroment. Defining precise criteria for making the judgment on these matters is probably not possible (or even desirable), but at least general principles should be established that permit relatively liberal use of the specialized languages in situations in which they offer the most cost-effective means of developing an application.

Considerable skepticism remains in the minds of many concerning the possible role of a powerful 4GL in implementing a large-scale MIS. Despite a number of demonstrated successes, 4GLs do not yet enjoy widespread use. In a few years, substantial and widespread evidence may well exist to support ambitious claims for the 4GLs, but that is not the case right now. The products are still evolving rapidly, making it difficult to choose the best one or to know when is the right time to make the move. Thus, even though the technology already looks capable and robust enough to build successful systems, it is not clear to many that the DoD should move in this direction at the present time.

Although a cautious waiting game would avoid the risks associated with using a rapidly - evolving technology, it would carry substantial risks of its own. The most serious risk - and in our opinion a very likely outcome - is that without fundamental changes the DoD will dig itself further into a hole, compounding the present software crisis. Conventional practices are no longer tenable now that much more attractive approaches are available. With the 4GL approach, risks can generally be managed; the same cannot be said of the conventional process, in which costs, schedules, and specifications frequently get out of control no matter how much effort management devotes to keeping things in reign.

The faults of the conventional process would undoubtedly be substantially alleviated if a wholesale switch to Ada could be made - provided it is accompanied by other necessary changes in methodology, training, and tools. A change of this magnitude would, however, also create some risks and require considerable transition time. It appears to us that the potential for making

dramatic improvements through the use of 4GLs more than compensates for a possible increase in risk compared to the Ada approach. The problems of managing MIS applications are great enough no matter what course of action is taken, so we should at least follow a path that offers a good opportunity to make a real difference.

DoD is currently undergoing substantial change in its management of MIS activities. The new leadership offers a new vision and aspiration level. The CIM initiative opens up an important mechanism for effecting change. The confluence of these favorable factors creates a great opportunity to make measurable and meaningful improvements. It is by no means out of the question for DoD to establish a leadership position in the effective management use of information technology. This opportunity will not last forever; it would be a pity not to grasp it while it is here.

VII. RECOMMENDATIONS

A. DoD should move as rapidly as feasible to high-productivity development environments for the implementation of management information systems.

1. DoD should require the use of an integrated 4GL or state-of-the-art CASE tools for new MIS applications.

2. DoD should establish guidelines for dealing with the introduction of 4GLs, covering such matters as their required and desirable features, skill requirements to employ them, and the avoidance of an excessive number of products that have to be supported.

3. Development of human resources for the implementation of application software should be given a high priority.

   - DoD should make training programs readily available for upgrading the skills of Ada programmers, 4GL developers, and those maintaining existing software. The emphasis in these programs should be placed on principles of sound software engineering and application design, rather than on the details of a particular language.

   - Educational programs should be developed and offered widely to equip managers with necessary knowledge to be involved effectively in the application software development process.

   - DoD should improve Ada educational programs in colleges and universities by supporting the development of training materials and making Ada compilers and tools available at an acceptable cost to institutions of higher education.

   - DoD should improve its personnel policies dealing with the recruiting, compensation, and career management of personnel involved in software development (including military personnel).

4. DoD should review its software development methodologies, documentation standards, and acquisition policies in light of the emerging trend toward a more adaptive and interactive implementation process.

5. DoD should launch a project to implement a major application (perhaps under the Corporate Information Management initiative) using an integrated 4GL. In addition to developing a needed, high-priority application, the project should aim at demonstrating the technical and economic feasibility of the 4GL approach and creating procedures and standards compatible with, and supportive of, a 4GL implementation. To the extent possible, this project should be "fast tracked," using selected development personnel, ready access to users throughout the development process, and the avoidance of unessential bureaucratic constraints.

6. DoD should initiate a second project to implement a major application using an integrated 3-1/2GL. The project should aim at demonstrating the technical and economic feasibility of the 3-1/2GL approach and creating procedures and standards compatible with, and supportive of, a 3-1/2GL implementation. To the extent possible, this project should be "fast tracked," using selected development personnel, ready access to users throughout the development process, and the avoidance of unessential bureaucratic constraints.

B. DoD should establish policies and guidelines for the use of programming languages other than Ada for end-user computing and applications for which an established domain-specific language is available.

1. DoD should establish exemption policies for the development of applications using specialized user-friendly languages suitable for end-user computing (spreadsheet programs, micro-based application generators, etc.). In defining end-user computing, exemption policies should balance the economy and responsiveness of this approach against its added risk of erroneous outputs, service disruptions, or undue dependency on user-maintainable programs.

2. For applications not exempted as end-user computing (e.g., because of their size, longevity, criticality, or widespread use), DoD should establish policies for the use of domain-specific languages that can be expected to achieve substantially greater productivity and responsiveness than Ada. The use of Ada should normally be waived

for AI, expert system, and DSS applications, but each case should be judged on its own merits based on a detailed development and maintenance plan submitted by the proposing agency. Other problem domains having a similar well-established body of knowledge and specialized languages should also normally have the Ada requirement waived.

3. Consistent exemption and waiver policies should be applied to all the services and DoD agencies.

4. Enforcement procedures should be established to insure compliance with DoD Ada policies.

5. The draft policy recommendations of the AEO/AJPO Working Group on Ada Waiver Policy should be used as a starting point for formulating specific DoD waiver policy.

C. DoD Should establish continuing mechanisms for promoting the best software development practices and encouraging software reuse.

1. Comparative analyses should be performed to assess results using different approaches to software development and maintenance. The data collected should include cost, schedule performance, productivity measures, software quality, and software maintainability. The analyses should be made available to policy makers and practitioners to aid them in setting software policies.

2. Software practices and standards should be reviewed periodically, and revised when appropriate in light of actual experience and advances made in software development technology.

3. Coordinated goals for software reuse should be established. Consideration should be given to such mechanisms as DoD funding of reusable Ada components and modifying contractual terms with software contractors to provide incentives for them to develop reusable packages.

A3. CODE REUSE

Code reuse is considered to have great potential as a source of productivity gains in software development. It has been estimated that approximately half the code in an information system is reusable - and the Department of Defense is currently spending over $4.5 Billion per year on administrative software.

In our discussions with many senior officials in DoD, we have concluded that there is a considerable lack of understanding of many of the issues dealing with software reuse. For instance, one senior officer from a computer-intensive domain, when asked what he envisioned as reuse, described what in essence would be a DoD repository into which all software acquired by DoD be deposited and made available for reuse. This approach would almost certainly prove unmanageable and of little practical use. As one general officer recently commented, reusability cannot remain the abstract buzz word it has been in recent years.

Ada is often described as a language that is particularly suited to take advantage of code reuse. Its portability means that Ada code written anywhere is potentially reusable, not just code written for a specific system. Ada is, by design, well suited to the integration of system components from multiple sources. Packages from other sites can be incorporated into a system with guarantees of interface compatibility and of freedom from side effects.

Ada has features (e.g., generics and packages) that make it easier to design code for reuse. A number of Ada repositories - libraries of code intended for reuse - have been established. Whether they will prove useful depends on the degree to which they can overcome the obstacles that have historically hindered software reuse.

A. The Costs of Reuse

Establishing an effective repository is expensive. A repository is not merely a bin into which programmers can toss software donations for the benefit of other programmers. Industry experience has been that programmers do not find it worth the time and trouble to reuse other

people's code in such cases. Rather, designing and stocking a repository is an exacting process with its own life cycle.

Domain analysis is required in order to identify packages or components suited to the target application systems, and to identify candidate system architectures.

The software, whether it is specifically written for the repository or whether it is taken from elsewhere, requires stringent testing. Programmers will not continue to use a repository that contains modules with errors - debugging other people's code is too expensive. The repository must also be catalogued and indexed so that users can identify potentially reusable packages. The packages must be well enough documented that these users can incorporate them into their systems with confidence. If minor modifications are needed, much of the benefit of reuse may be lost. It is, however, far easier to find a close fit than a perfect one, and so it is often necessary to modify code. (Object oriented design, to which Ada is well suited, may make the reuse of isolated architectural elements more practical in the long run.)

Thorough testing, good documentation, and cohesive and reusable modules are generally good practices that can greatly reduce long-run software costs, yet we are referring to them here as extraordinary efforts. Since programmers are often penalized for taking the extra time these practices require in the short run, it is often difficult to get programmers to pay the "tax" of making software reusable. Until and unless there are fundamental changes in the short-term incentives of project managers, it may be necessary for the repository to bear these added costs. It is probable that a repository will also have to maintain and support its software.

Once a repository is established, the potential user must invest the time and effort required to locate reusable modules. At the current level of cataloguing and indexing, the search effort is still substantial. (We have been describing reuse in terms of a programmer searching for a module to perform a given function. Greater levels of reuse can be achieved if the target system has been designed to take advantage of the repository.)

B. The Scope and Granularity of Code Reuse

We have been presenting code reuse as a mechanism for saving a programmer the time it would require to write and test code that is already available. This saving, however, is not as great as it might seem. For every dollar spent developing software, about 40 cents are spent on analysis and design, before the code is written, and about 35 are spent on integration and installation after

the modules have been written and tested. (Experience with modern practices suggests that early-phase expenses will continue to grow at the expense of coding.) This means that if we could construct a system entirely out of reusable code we would still only be reducing our development costs by some 25 percent (ignoring the added costs of reuse). At the more attainable level of 50 percent reuse, we would be reducing costs by one eighth. This is a respectable saving, but it is nowhere near what we might hope to achieve through reuse.

Significantly greater savings can also be realized if we can reuse the products of earlier analysis and design efforts. This would be relatively simple if variation in the architecture of ostensibly similar systems (e.g., two inventory systems) were due primarily to arbitrary design decisions. In practice, systems at different sites, even if they serve similar functions, must interact with different environments.

A related issue is the *granularity* level of reusable components. At the highest level, we could have an entire application intended for reuse - which is exactly the purpose of an application package, such as a payroll package. (These are also called "commercial off-the-shelf," or COTS, applications.) At the lowest granularity level, we might have very small reusable chunks of code, as we find, for example, in the typical subroutine library of mathematical functions.

The choice of granularity involves a tradeoff. On the one hand, a large granule of code yields a very high productivity gain when it is reused, but it is generally hard to find a close fit between a user's needs and large aggregations of code. On the other hand, a library of many small granules generally provides chunks of code that match a user's needs, but the productivity gain is smaller and the problem of indexing and search is greater. (Mathematical subroutines have been used for a long time and yield tremendous benefits despite their generally fine granularity because they are relatively easy to comprehend, standardize, and index.)

Most systems inherit the data in which they must work and the other systems with which they must interact, so that adapting one architecture to another environment may not be practical on a single-project basis. Also, in administrative systems, half the development effort may be spent on the user interfaces. If these do not fit existing preferences and habits, the result will often be a high level of user dissatisfaction.

C. Maintenance

For every dollar spent developing a system, two dollars are subsequently spent maintaining it. Very little of this is spent fixing flaws; most of it goes to modifying the system to keep it useful and usable in a changing world, often over a lifespan of ten to twenty years.

Code reuse can indirectly reduce maintenance costs. The characteristics of a module that make it reusable - good documentation, thorough testing, high cohesion, clean and minimal coupling - are also the characteristics of good maintainable code, whatever the language used to code it. Multiple use of a module within the same system further reduces maintenance costs by reducing the number of modules that need to be modified to accommodate a given change.

The fact that a module was originally an instance of reuse will not reduce the cost of modifying it. (Savings will be realized, of course, if the same modifications must be made to multiple instances of a module, and those instances are being maintained by the same programmer.) Much of the maintenance effort will, however, involve the writing of enhancements, and this presents new reuse opportunities. If code reuse during development reduces downstream life cycle costs, it is primarily by being associated with good programming practices.

D. Industry Experience

A number of organizations (with non-Ada environments) have attempted to achieve high levels of code reuse. They have developed repositories and mandated design-for-reuse, and achieved reuse levels in the 30-50 percent range.

There are some intrinsic advantages to such single-organization repositories. There are common architectural standards, common styles, and related application domains that make it much more likely that a given module will be usable elsewhere within the one organization. The continuity of personnel makes it more likely that a reuse opportunity will be recognized or remembered from a previous project, greatly reducing the cost and frustration of searching the repository.

The advantages of familiarity and commonality appear to be crucial. One organization that established a company-wide repository found that less than 10 percent of the reuse involved reuse across application areas. It also found that, beyond a certain (and modest) repository size, the level

of reuse reached a plateau, and did not increase as the repository grew. Programmers reused little software with which they were not personally familiar.

It is worth noting the most common instances of successful non-repository-based reuse: application generators and off-the-shelf systems. Application generators capture most of the benefits of reuse by restricting their domain. They succeed by identifying common functions within their domain to meet the needs of enough customers to establish a profitable market. Off-the-shelf systems tend to offer their user less flexibility than they would prefer - but at a greatly reduced cost and risk. Such systems are often customizable, to a greater or lesser degree (generally through the specification of appropriate parameter values).

Thus we have three competing reuse paradigms. In the application generator, the line between reuse and custom-programming with a 4GL is blurred. In the off-the-shelf application, we accept an imposed design for the sake of the savings this brings. Repository-aided custom programming is effective to the extent that there is a good fit between our architecture and that supported by the repository.

In all three cases, significant saving through reuse are only realized when we take advantage of previously performed domain analysis and design. Only minor savings can be realized through the spare-parts-bin approach to code reuse.

E. Conclusions

The amount of new code that must be written may be cut in half through a program of code reuse. Even if the only benefit of reuse is the reduction of programmer effort, this can still mean a reduction of over 10 percent in development costs. In DoD, this translates to an annual saving of over $250 million in the development and enhancement of management information systems. Greater savings may be realized if reuse is extended to all phases of the software life cycle.

Ada simplifies some of the technical barriers to code reuse, and may reduce the costs of integrating reusable software into a system. (This advantage may be compromised if a developer using an integrated CASE environment attempts to reuse Ada code produced and maintained by a different integrated CASE environment.) Code reuse, however, will not simply "happen," no matter what language is used. Successful code reuse, in various languages, requires the construction of well-supported and relatively specialized repositories, with funds and backing.

Management information systems have great potential for reuse within DoD. They are well understood. There are great commonalities of function across sites. Unlike embedded systems, most of the MIS work is done in-house. The CIM initiative is aimed at capturing the great advantage of achieving high reuse while still preserving adequate flexibility.

While it is not practical to mandate a single architecture at this time - for one thing, systems must continue to interact with the billions of lines of old code that have accumulated over the past decades - it is possible to encourage movement in that direction:

- DoD can develop one or more relatively open Ada-based MIS architectures. Investments in repositories of compatible, reusable software, and even in compatible application generators, can make their use financially attractive. It might be worth while to provide incentives not only to make new systems compatible with these standards, but to re-engineer or replace old non-Ada systems. (A number of pilot repositories have, in fact, been established. It is still too early to evaluate their success.)

- A more modest goal might be to encourage code reuse at the local level. Any site supporting several million lines of code can benefit from code reuse, as long as it is willing and able to bear the overhead of establishing and maintaining the repository. External support might take the form of funds, training, and incentives. Note that this approach is not inconsistent with the centralized-repository approach. Indeed, central repositories could add to their responsibilities the support and encouragement of local ones.

Code reuse, like many other software engineering techniques, achieves most of its leverage as part of an overall management approach to instituting sound software engineering practices. Systems that are designed with reuse in mind, and that produce software that is reusable, realize the advantages of reuse in conjunction with the benefits of other advanced methodologies.

A4. COST-EFFECTIVENESS ISSUES

The objective of this appendix is *not* to demonstrate whether Ada is or is not the most cost-effective language for all projects. We believe no one can at this point. Not only is the total number of DoD projects using Ada unknown; the costs and benefits of Ada implementation are also unknown.

Our objective, instead, is to present a framework to support (not replace) a decision maker in conducting a cost-effectiveness evaluation of Ada. Specifically, we present a set of issues that we feel need to be addressed, as well as summarize available empirical findings.

To assess the economic impacts of Ada, one must look beyond a *single* program's development cycle. "The major costs associated with software management encompass the total lifecycle. Accordingly, the economic and productivity aspects must encompass the post-deployment support as well as initial development and acquisition costs" (McPherson, 1991). Furthermore, the analysis should not be constrained to a "project-in-isolation" perspective. Instead, one needs to consider the economic implications on the aggregate portfolio of software programs in the DoD.

Our discussion will cover the following points:

- Non-deployment opportunity cost.
- Aggregate post-deployment issues.
- Development/acquisition cost issues.

A. Non-Deployment Opportunity Cost

One of the major challenges facing the DoD is the imbalance that exists between DoD's demand for software systems and the available supply of programming manpower. While the average productivity of individual developers is increasing over time (e.g, with training, new CASE tools, etc.), the rate of increase is rather small. Most researchers point to about a 4 percent average increase in programmers per year, with somewhere around 12 percent required to stay

even (Charette, 1986). As a result, both DoD as well as the commercial sector are experiencing mounting backlogs of new applications.

The productivity of a single programmer is measured in terms of some output (say in lines of code) per person-month. The productivity of an organization, on the other hand, is measured in terms of aggregate output per unit of time, where:

Organizational Productivity = (productive members) x (average productivity)

As argued in the main body of this report, integrated 4GLs are powerful tools that can significantly enhance programming productivity in the 1990's (Emery and Gremillion, 1991). DoD's *organizational* productivity may be enhanced further, and perhaps more significantly, by increasing its pool of "productive members." Many organizations have successfully achieved this by promoting and facilitating *end-user computing*. The DoD can achieve similar gains, at least for a subset of its applications (e.g., spreadsheet applications and small administrative systems).

Experience in the commercial sector suggests that end-user computing has been facilitated, in large part, by the availability of user-friendly 4GLs and domain-specific software development tools. Such tools allow end-users to create entire applications themselves, and to create them quickly. The tools provide such capabilities as graphics, spreadsheets, modeling, and ad-hoc information retrieval. They are easy to learn and easy to use.

Ada, in comparison with successful end-user tools, is not easy to learn or use. "Ada requires more experience than other languages before personnel can become proficient. This is because of the unique features of the language (generics, tasks, overload operators, exception handlers, etc.) that are not present in other languages. It was generally agreed that it will take longer to appreciate the trade-offs in determining which feature of the language is best to implement a particular algorithm" (IIT, 1989).

We, therefore, recommend that consideration should be given to relaxing the Ada enforcement directive for those application domains amenable to end-user computing.

B. Aggregate Post-Deployment Issues

The economic and productivity aspects of Ada encompass the post-deployment support as well as initial development and acquisition costs on the aggregate portfolio of software programs in

the DoD. From an aggregate long-term perspective, standardizing around Ada is a powerful incentive. Lacking empirical findings to quantify this, we can only suggest that aggregate post-deployment benefits can be significant enough to favor Ada even where *building* in Ada may be slightly more expensive.

The long-term gains from standardizing around Ada fall in two areas: higher maintenance productivity and higher portability and reusability.

DoD systems (especially embedded systems) have long lifespans - typically 10 to 15 years - and must often be quickly adapted to frequent changes in the environment. As a result, maintenance costs of DoD systems have been, and will probably continue to be, high. "The features, functions, and structure of Ada are purported to make it easier to develop software that is more readable, more modular, more understandable, and, in general, more easily maintained" (Smith, 1991).

Second, respondents in the Smith (1991) study reported that portability of Ada systems was significantly greater than that of systems written in other languages:

> "Ada code is more portable than other languages for two reasons. First, the DoD is enforcing Ada as a standard by requiring that all Ada compilers used for military software pass a suite of tests that verifies conformity with the language definition. This requirement insures, as much as is possible, that dialects of Ada are not used for the development of DoD Software... Second, Ada has language features such as packages and representation specifications that allow software developers to isolate machine dependencies in the software" (Smith, 1991).

C. Development and Acquisition Cost Issues

We start our analysis by summarizing the available empirical data pertaining to Ada's cost-effectiveness. These empirical results should be used with caution, however. The problem, in part, is that none of the studies was designed specifically for the purpose of assessing the economic benefits of using Ada vis-a-vis other languages. Such a study is urgently needed.

1. A summary of empirical results. Four findings summarize the results of the empirical studies published to date.

In a study by Reifer (1991), "Ada projects which have their productivity numbers public are showing about a 20 percent decrease in costs (over time) after they have made the transition to

what is call the 'Ada mindset.' The mindset involves learning and applying new software engineering principles, modern methods like OOD (Object oriented design), and advanced packaging concepts and tools, as well as the Ada programming language itself."

There is a significant *learning delay*. A finding by RCI researchers (Reifer, 1991), suggests that it takes at least three to five Ada projects worth of Ada experience before significant productivity gains can be achieved.

Similar results were observed by the Navy Center for Cost Analysis (Gross, 1990):

	FORTRAN Project	1st Ada Project	2nd Ada Project	3rd Ada Project
LOC/man-day	12-30	22	26	34
% code reuse	5-20	0	15-35%	25-42
Errors/KLOC	3.5-8	2	1.5	1

The learning curve is driven by two issues: the difficulty of learning the language and the amount of time to become proficient in it.

Because Ada is a relatively new language and Ada programming support environments are still evolving, a body of trained personnel are not available to develop new Ada systems.

> "Ada projects will not be able to benefit from lessons learned in previous projects which in turn will lengthen the development schedule. In addition, ... Ada requires more experience than other languages before personnel can become proficient. This is because of the unique features of the language (generics, tasks, overload operators, exception handlers) that are not present in other languages" (IIT, 1989).

The *effort distribution* of an Ada project may differ from projects using other languages because of the emphasis placed on the early stages (requirements and design) of the lifecycle.

A great amount of effort is devoted to the requirements and design phase of Ada software development projects when compared with traditional software development projects. This increased initial investment is offset by a corresponding decrease in later phases, such as when writing and testing the code (GAO, 1989).

One of the controversial findings is that Ada projects get cheaper as they get larger (Reifer, 1991). The general relationship relating cost and project size takes the form:

$$COST = a\,(SIZE)^p$$

The term *p* is referred to as the *power law*. If the power law is less than one, an economy of scale exists. The analysis reported by Reifer (1991) for more than 140 Ada projects "has yielded a power of 0.95, after the transition to Ada has taken place. This finding has been substantiated independently by a recent detailed statistical analysis of AFATDS data."

*2. **Significant cost drivers.*** While the above empirical findings do provide useful insights into the economics of Ada, they certainly fall well short of providing all the answers. In fairness, it must be stated that the above studies were not designed to specifically assess the economic benefits of using Ada vis-a-vis other languages. In this section, we seek to present a complementary "tool" to tackle the cost effectiveness issue: the set of cost drivers that can exert a significant impact on Ada software development. Specifically, we will discuss the following set:

- Effective size.
  - Programming language level.
  - Reuse.
  - Rework.
- People issues.
- Development tools.

a. Effective size. "The most significant influence on software costs is the number of source instructions one chooses to program" (Boehm, 1987). Besides the inherent size of the system being developed (a language independent driver), the *effective* number of instructions to be built will be affected by the level of the programming language, the degree of reuse, and the amount of rework.

Programming language level. A 1987 SEI report compared the size of Ada programs vis-a-vis other languages. It was found, for example, that Ada could yield 50 percent more lines of code than languages such as FORTRAN. A number of explanations were presented:

> "Ada is generally more specific and more complete than (FORTRAN), resulting in more lines of code. The benefit is that Ada code should be more reliable and easier to read, understand, and maintain. Ada is richly declarative and descriptive regarding data structures. These characteristics increase lines of code and also allow more errors to be caught by the compiler as compared to languages such as FORTRAN that do not possess these capabilities" (SEI, 1987).

There were no studies that *directly* compare Ada to 4GLs. Capers Jones' (1986) comparison of the number of source statements required to code one Albrecht function point provides one benchmark. His results indicate that 71 Ada statements were required to code one function point. On the other hand, only 16 statements were needed by 4GL query languages, and even less (6 statements) by speadsheet languages.

Still, Ada does have several features that could potentially enhance productivity: packages, overload operators, strong typing, generics, tasks, and exception handlers. The complexity of the features to be used on a given system depends strongly on the application type of that system (avionics, business, command and control).

> "For example, a business system and a command and control system would both contain exception handlers but the difference would occur in the complexity of the exception handler. In a business system, the exception handlers may be less complex because a critical error would cause only loss of data. In a command and control system, the handlers would be more complex because a critical error could cause loss of life (IIT, 1989).

Reuse. Smith (1991) found that Ada code can be easier to reuse than the code of other languages. Two factors contribute to this. First, DoD's validation of all Ada compilers to verify conformity with the language definition minimizes the dialect proliferation problem. Second, Ada has language features (packages, representation specifications) that help isolate machine dependencies.

Rework. A high level of rework on a project translates into lower productivity, as more and more resources are diverted from producing new products to reworking old faulty ones.

A great amount of effort is typically devoted to the requirements and design phase of Ada software development projects when compared with traditional software development projects. This increased initial investment often translates into a corresponding decrease in requirements and design errors (which often constitute the most expensive type of rework). Furthermore, the Smith (1991) study found that:

> "... bugs in Ada code were relatively easier to find and fix during development... In part, this is due to the extra checks that Ada compilers are required to do during compilation. The structure of the language itself also appeared to contribute to the ease of locating and correcting code errors. The constructs of the Ada language facilitate the compartmentalization of the code. If used properly, these features help to localize the effect of changes to the code that must be made to correct code errors."

b. People Issues. Because of the emphasis placed in Ada projects on the early stages (requirements and design) of the life cycle, Boehm and Royce (1987) found that analyst capability differences were more significant on Ada projects, while programmer capability differences were less significant. In addition, the richness and complexity of Ada creates wider productivity ranges. This translated into:

- Larger productivity benefits from acquired Ada expertise.
- Larger cost penalties from Ada ignorance.

Organizations that are relatively advanced in the practice and processes of software engineering will thus have an easier time adopting and utilizing Ada. Smith (1991) recommends that organizations have at least one in-house Ada guru, that is - a person that has a very deep understanding of the language and its features.

c. Development Tools. Many vendors are now offering tools for designing, controlling, documenting, testing, and maintaining Ada software. The Ada Programming Support Environment is a set of coordinated tools for the Ada language. It contains such software tools as text editors, compilers, linkers, static analysis tools, dynamic analysis tools, terminal interface routines, file administrators, command interpreters, configuration managers, etc.

The GAO study (1989) suggests that while many of Ada's *basic* tools are now available, the problem is that many tools are not yet mature. Once they do mature, the cost of software development in Ada would be expected to decrease further.

D. References

Boehm, B., "Improving Software Productivity," *Computer*, September, 1987, pp. 43-57.

Boehm, B. and W. Royce, "Ada COCOMO: TRW IOC Version," *Proceedings, Third COCOMO Users' Group Meeting*, SEI, Pittsburgh, PA, Nov. 1987.

Charette, R.N., *Software Engineering Environments: Concepts and Technology*, Intertext Publications, Inc., 1986.

GAO, "Programming Language: Status, Costs, and Issues Associated with Defense's Implementation of Ada," United States General Accounting Office, Washington, D.C., March, 1989.

Gross, S., Presentation at the Naval Postgraduate School, Fall, 1990.

Emery, James C. and Lee L. Gremillion, "Adopting a New Software Development Paradigm: A Strategic Imperative," Naval Postgraduate School, April, 1991.

IIT Research Institute, "Estimating the Cost of Ada Software Development," IIT Research Institute, Lanham, Maryland, April, 1989.

Jones, C., *Programmer Productivity*, McGraw-Hill Book Company, 1986.

McPherson, M., Personal communication.

Reifer, D., "Economics of Ada - Guest Column," *Ada Strategies*, Vol. 5, No. 1, January, 1991, pp.8-9.

SEI. "Ada Adoption Handbook: A Program Manager's Guide," Software Engineering Institute, Carnegie Mellon University, Technical Report CMU/SEI-87-TR-9, May, 1987.

Smith, G. N. "Firms' Adoption and Use of New Information Technologies: A Theoretical Framework and an Empirical Investigation of Ada in the U.S. Defense Industry," GSIA, Carnegie Mellon University, April 18, 1991.