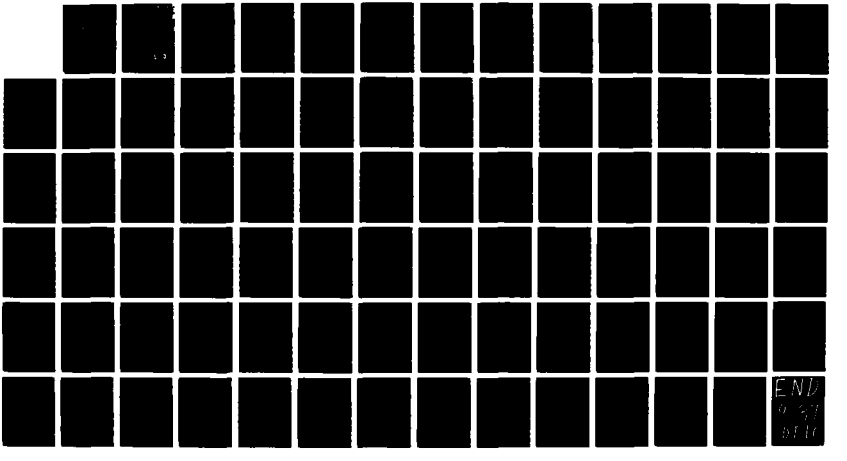
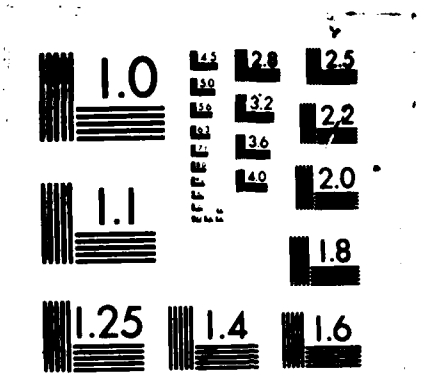
AD-A183 665 USE OF ADA (TRADEMARK) FOR FAA'S ADVANCED AUTOMATION SYSTEM (AAS)(U) MITRE CORP MCLEAN VA CIVIL SYSTEMS DIV V R BASILI ET AL. APR 87 MTR-87W77 DTFA01-84-C-0001 1/1

UNCLASSIFIED F/G 12/5 NL





Use of Ada®
for
FAA's Advanced Automation System (AAS)

MITRE

MTR-87W77

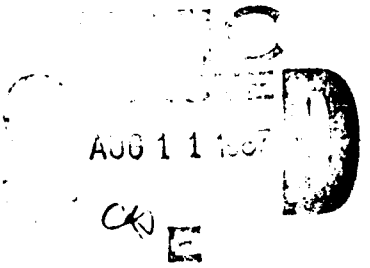
Use of Ada®
for
FAA's Advanced Automation System (AAS)

Dr. Victor R. Basili
Dr. Barry W. Boehm
Judith A. Clapp
Dale Gaumer
Dr. Maretta Holden
John K. Summers

April 1987

MTR-87W77

SPONSOR:
Federal Aviation Administration
CONTRACT NO.:
DTFA01-84-C-00001
PROJECT:
1763C



The MITRE Corporation
Civil Systems Division
7525 Colshire Drive
McLean, Virginia 22102-3481

MITRE Department
and Project Approval: L. G. Culhane

L. G. Culhane

®Ada is a registered trademark of the U.S. Government (Ada Joint Program Office).

ABSTRACT

The Federal Aviation Administration (FAA) asked MITRE to organize and participate in a study on the use of Ada for the Advanced Automation System (AAS) procurement. A team of MITRE and non-MITRE software experts was assembled to address the following: First, if Ada is selected for AAS, how can the FAA judge the ability of a contractor to build AAS in Ada? Second, if Ada is used, how should the FAA change its contract monitoring procedures to best manage the project?

The study participants concluded that the size and complexity of AAS make software engineering, and not the programming language, the key issue. Moreover, if properly used, Ada facilitates good software engineering practices and a sound software development process. Therefore, the participants unanimously recommended Ada (with appropriate waivers for the use of non-Ada code where necessary) as the appropriate choice for AAS. However, there are several qualifications to this recommendation. The FAA must use a modified software development approach to take advantage of what Ada has to offer. In addition, risk reduction activities must be carried out to address the risks associated with Ada. These qualifications, and others, are documented in the report. Also included are a characterization of an Ada software development, identified risks with recommendations for addressing them, and recommended near-term FAA activities.

Suggested Keywords: Ada, AAS, Advanced Automation System, Software engineering, Risk management.

FOREWORD

This report was commissioned by Martin Pozesky, Deputy Associate Administrator for NAS Programs, ADL-2, of the Federal Aviation Administration, who asked me to chair a study addressing the risks of using Ada on the Advanced Automation System procurement. The charge was directed specifically toward making recommendations for assessing development contractor readiness and changes in the contract management process resulting from Ada. The study participants were given a free hand in drawing the conclusions contained in this study. Its results have been presented orally and are recorded in this report. Since a major concern of the study was timeliness, the report is not as extensive or complete as it might have been.

The report was compiled from the experience and expertise of the study participants and the generous information supplied by the eight individuals who were interviewed during the study. These individuals, chosen by the participants, represented an extensive experience and knowledge base in the use of Ada. The study participants were Barry Boehm, Judith Clapp, Dale Gaumer, Maretta Holden, and John Summers. The interviewees were told that they and their comments would be anonymous.

I would like to acknowledge the support of The MITRE Corporation, especially Arthur Salwin, who took on the difficult chore of writing and organizing this report based upon the comments, discussion, and feedback of the study participants; Jack Fearnsides, who acted as liaison and guide through the Federal Aviation Administration; and Barbara Wright, who taught us as much as we could learn about the Advanced Automation System.

Victor R. Basili
Chairman, AAS/Ada Study

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Concluded)

LIST OF FIGURES

LIST OF TABLES

EXECUTIVE SUMMARY

INTRODUCTION

In 1981, the Federal Aviation Administration (FAA) embarked on a comprehensive plan for modernizing and improving the United States Air Traffic Control (ATC) system. The Advanced Automation System (AAS) is the largest of over ninety projects in this National Airspace System (NAS) Plan; it is also the most software-intensive, with over a million lines of code expected to be developed. The Acquisition Phase contractor for AAS will be asked to select a single high order computer language for use on the project. Anticipating that the contractor might select Ada, and citing lack of FAA experience with Ada, the FAA's Deputy Associate Administrator for NAS Programs, ADL-2, asked The MITRE Corporation to organize and participate in a study. He asked that leading MITRE and non-MITRE software experts be assembled to address the use of Ada for AAS.

This report documents the findings and recommendations of the study participants.

CHARACTERIZING A SOFTWARE DEVELOPMENT

The use of the very best in software engineering is imperative for large, critical, and complex systems such as AAS. Therefore, the study participants' consensus view of how a software development might proceed on such large procurements is outlined. Next, the benefits of the described approach are discussed, along with the advantages offered by Ada in achieving them. Based on this framework, qualified recommendations on the use of Ada for AAS are provided.

Elements of an Ada Software Development

While there is more than one way to use Ada in support of good software engineering, one reasonable way to proceed is to use Ada in a software development process characterized by a composite of the following elements:

- Compilable designs
- Small, expert design team up front
- Benchmarks, prototyping, and incremental development
- Appropriate milestones and deliverables
- Early access to mature development environments

- Training in Ada and software engineering
- Ada-oriented Software Development Plan

The above approach also provides a framework for discussing risk and risk reduction activities (as summarized below).

Advantages of an Ada Software Development

Use of Ada does not inevitably lead to good software engineering; a sound development approach must be followed. However, if properly used, Ada provides a natural vehicle for employing sound software engineering practices.

Ada use stimulates a more effective development approach. The richness of the language, which also contributes to the rigor of Ada as a design notation, is a major reason for this. In addition, the multiple individual walk-throughs provide shorter, more focused reviews than a monolithic Critical Design Review (CDR) would provide. Ada also offers greater continuity of activities across phases of a project. Furthermore, the thoroughness of automated compiler checking permits many flaws to be uncovered early in the development process, which in turn permits high-risk elements to be addressed early in the framework of the total design.

Lower life-cycle costs, enhanced software readability and traceability, improved maintenance productivity and system evolution, and software with higher quality and increased reliability are expected benefits of such an Ada development approach. Actual project experience shows that expectations of a shorter, smoother integration phase can be realized. Unfortunately, because Ada technology is new, no large Ada systems are available for verifying the expected life-cycle advantages.

Compilable designs written in Ada Program Design Language (PDL) provide all the advantages of other-language PDLs, and much more. For example, other high order languages do not require compiler checking of module interfaces. In addition, the use of Ada syntax provides increased rigor for interface definitions; compiler checking ensures that these interfaces will be used consistently.

The machine-readable deliverables resulting from compilable designs can be generated with minimal extra effort, since they are produced as a natural outgrowth of the development process. They result in a single representation of the design; manual translations of the products of one phase into the notation of the next (with the inevitable mistranslation errors) are no longer needed. Thus there is better traceability between the products of a given phase and those of the succeeding ones throughout the development process. There is also better traceability between documentation delivered to the customer, and the system that the

documentation purports to describe. Finally, the machine-readable deliverables can be more easily reviewed; in fact, automation can be used to assist in the review process.

Using Ada on AAS: Recommendations

The study participants concluded that the size and complexity of AAS make software engineering, and not the programming language, the key issue. Moreover, if properly used, Ada best facilitates good software engineering practices and a sound software development process. Therefore, the study participants unanimously recommended that the FAA commit to Ada as the appropriate choice for AAS, given four strong qualifications to ensure that Ada's potential is realized:

- The software development process must be modified; the FAA and contractors must use an approach tailored to take advantage of modern software engineering practices and the support that Ada provides for them.

- It should be expected that appropriate selective use of non-Ada code within the framework of an Ada design may be necessary when Ada is found to be inadequate.

- Contractor readiness for an Ada software development must be evaluated.

- Positive risk reduction activities must be undertaken addressing the risk areas associated with Ada.

The large system nature of AAS makes it well suited for the advantages offered by Ada; no other language offers more advantages. If one of the likely alternative languages were chosen, the large system problems posed by AAS would remain, but many of Ada's advantages would not be available for addressing them.

At the same time, it should be noted that the advantages offered by Ada will not accrue by default; positive steps must be taken to realize them. It is of particular importance that both contractor and customer be well-trained in Ada technology. If positive steps are not taken, or if risk reduction activities indicate significant problems (e.g., compilers are inadequate), then the advantages of Ada usage will be outweighed by the associated risks. In such a case, the recommendation to use Ada would not stand. If the FAA commits early to Ada, risks can be assessed early in the development process. Problem areas thereby uncovered can be addressed in a timely fashion, or if need be, a fallback position can be pursued without wasted cost and effort having accrued.

IDENTIFIED RISK AREAS

Although Ada offers the potential of significant advantages on AAS, its proper use can only diminish the risks, not eliminate them. Some of the risks arise from the newness of the Ada technology; others, however, are not peculiar to Ada. The identified risk areas include the following:

- Ability to meet real-time and availability requirements--The hardware architecture, run-time systems, and the need for proper use of Ada's features and non-Ada code pose risks to meeting the demanding AAS performance requirements.

- The inherent large size of the AAS software--The inability of compilers to handle large quantities of code has been a key problem area on other Ada projects.

- Adequacy of computing resources--Ada projects are typically heavy users of development resources.

- Customer readiness--The newness of Ada technology and its many differences pose risks of whether a customer is prepared to award and monitor an Ada contract.

- Software support systems on development and target machines--Their lack of maturity provides increased risk on an Ada procurement.

- Personnel and management--Risk areas on any software procurement include staffing profiles and experience, and management commitment and flexibility.

- Schedule and cost--There is virtually no historical database of Ada projects to provide guidance.

Early risk reduction activities are needed to address the above risks; appropriate risk reduction activities were recommended by the study participants where possible. However, it is realized that, even given very good early risk reduction awareness and associated activities, there will be inevitable modifications to the software environments for the system being developed. Thus the contractors should document in their plans an approach for addressing problems when they arise, and should allocate sufficient time and money resources for proceeding with the documented approach.

To summarize, the proper use of Ada can reduce some of the risks inherent in a large software development. However, the residual risks that remain must be addressed.

ADDRESSING THE RISKS

In addition to specific risk reduction activities designed to address individual risk areas, the following general risk reduction activities are recommended:

- Each contractor's proposal should be required to include a Risk Management Plan.
- Each contractor's proposal should be required to include an Ada-oriented Software Development Plan.
- A software engineering exercise should be conducted. First, the FAA would define a software problem. Then the contractors would be asked to solve it using their AAS personnel, software methodologies, and toolsets. This would test the contractors' planned methodologies and their ability to design and implement software using them. The results could be used by the contractors to improve their methodologies, and by the FAA as technical criteria for assessment in making the Acquisition Phase contract award, as an adjunct to the technical evaluation of the proposals.
- The AAS contractors should be required to develop compilable Ada designs for their proposed systems and deliver them to the FAA in machine-readable format. A corollary to this recommendation is that the FAA should thoroughly review the Ada designs.
- Benchmarks should be required on development and target machines to address many of the identified risks.

NEAR-TERM FAA ACTIVITIES

Multiple near-term FAA activities must be undertaken to realize the benefits of an Ada development and to address the inherent risks:

- The FAA must develop in-house expertise and access outside expertise in Ada and in software engineering.
- The FAA should prepare and dry run a software engineering exercise.
- The FAA should revise the AAS milestones to reflect an Ada development process (e.g., replace a monolithic CDR with incremental reviews).

- The FAA must develop a strategy for implementing the recommendations that it chooses to undertake. For example, benchmarks must be selected, and criteria for evaluating the results must be established.

- The FAA must assess the feasibility of a fixed price contract, if one is being considered for AAS. In doing so, the FAA should consider whether the estimated costs are realistic and known with a high degree of certainty, and whether design and requirements are thorough and stable. Where these conditions are not met, a cost plus contract--or a contract with a mix of fixed price and cost plus components--should be used to provide the needed flexibility.

- Schedule revision may be necessary for the FAA and contractors to carry out activities needed to ensure a successful software development effort on AAS.

- The FAA should bring the software maintenance organization on board now. The long-term success of the AAS program depends upon the early, active involvement of this organization. Recommended near-term activities for it include the following: writing of a Software Maintenance Plan, involvement in training and preparation of a software engineering exercise, and support in preparing the Request for Proposals (RFP) and Statement of Work (SOW).

- The FAA must prepare for awarding an Acquisition Phase contract and monitoring the post-award development activities. Training, development of contract award criteria, and establishment of FAA computing facilities are needed. Criteria need to be set for each intermediate product at each milestone so that project tracking can occur.

1.0 INTRODUCTION

The United States Air Traffic Control (ATC) system is the busiest and most complex ATC system in the world. By the early 1980s, it had become apparent that the expansion capability of this system was limited. In particular, requirements imposed by the continual increase in aviation activity could not be met indefinitely, and automation improvements to labor-intensive tasks were being deferred because of capacity limits in the automated systems. Furthermore, maintenance of aging equipment was proving to be increasingly costly, with equipment manufacturers unable to guarantee the provision of spare parts indefinitely, regardless of cost.

Therefore, in 1981, the Federal Aviation Administration (FAA) embarked on a comprehensive plan for modernizing and improving ATC services. This multi-billion dollar plan consists of over ninety projects dealing with all aspects of ATC, from runway lights and navigation to communications and computers. The largest of these projects is the Advanced Automation System (AAS). It is also the most software-intensive, with over a million lines of code expected to be developed.

1.1 Advanced Automation System (AAS)

The purpose of AAS is to provide an automation system that includes controller displays, new computer software, and new processors. AAS will provide the capacity to handle projected air traffic loads through the 1990s and beyond, the capability to perform existing and new automated functions, and a high degree of reliability and availability. More details on AAS are provided in the tutorial in Appendix A.

In August 1984, the AAS program entered a Design Competition Phase (DCP). Competitive contracts were awarded to two teams of contractors--one led by IBM, the other by Hughes Aircraft Company--for the design of AAS. In mid-1987, a Request for Proposals (RFP) will be issued for the Acquisition Phase, with contract award for AAS production scheduled for 1988. In addition, a Statement of Work (SOW) will be issued in Spring 1987 for continuation of the DCP until the award of the Acquisition Phase contract. It is the FAA's intention to task both contracting teams with risk reduction activities during this remaining portion of the DCP.

1.2 The FAA Ada Study

The AAS Acquisition Phase contractor will be asked to select a single High Order Language (HOL) as the computer language for use in AAS. Anticipating that the contractor might select Ada as the HOL, and citing lack of FAA experience with Ada, the FAA's Deputy Associate Administrator for NAS Programs, ADL-2, asked The MITRE Corporation to

organize and participate in a study. He asked that leading MITRE and non-MITRE software experts be assembled to address two issues regarding the use of Ada for AAS: First, if Ada is selected, how can the FAA judge the ability of a contractor to build AAS in Ada? Second, if Ada is used, how should the FAA change its contract monitoring procedures to best manage the project? Answers were needed in time to have impact on the Acquisition Phase RFP.

In September 1986, Dr. Victor R. Basili of the University of Maryland agreed to chair the study. In conjunction with him, the remaining study participants, listed in Table 1-1, were chosen. The following ground rules were established and agreed to by the FAA:

- The study participants would be given total latitude regarding outcomes to be considered--ranging from "don't use Ada" to "require the use of Ada"--in making their recommendations.

- No source selection sensitive material would be made available to the study participants; this enabled them to make recommendations freely, without having to be concerned about impacts on the DCP.

On 7 December 1986, the study participants held an organizational meeting. Following a tutorial presentation on AAS, they agreed that, because of the newness of Ada technology, it would be best to access as much real-world Ada experience as possible. Therefore, they decided to gather information from eight outside experts, who would be invited to make short informal presentations to the entire study team and discuss their Ada experiences.

To this end, a list of candidate experts was generated. Fortunately, the top eight choices enthusiastically agreed to appear before the study participants. They were guaranteed anonymity and, therefore, their names do not appear in this report. To focus their preparations, they were provided four questions (see Figure 1-1) that had been generated at the organizational meeting in December.

On 7 and 8 January 1987, the study participants met with the eight experts. The experts did not hear each other's presentations, but all of the study participants were present for all of the presentations. On 9 January, recommendations were formulated and drafted into a briefing. Final versions of that briefing were subsequently presented to senior FAA management, including the Administrator.

The above information-gathering procedure was effective, and a set of excellent recommendations from the experts was adopted by the study participants. However, these recommendations will not be attributed to individuals for two reasons: because the experts were guaranteed anonymity, as noted above; and because they were a source of inputs only,

TABLE 1-1
STUDY PARTICIPANTS

Study Chairman: Dr. Victor R. Basili, Chairman, Computer Science Department, University of Maryland

Dr. Barry W. Boehm, Chief Scientist, Defense System Group, TRW Inc.

Judith A. Clapp,* Assistant Director for Software Technology, The MITRE Corporation

Dale Gaumer, Engineering Scientist, Magnavox Government and Industrial Electronics Company

Dr. Maretta Holden, Manager, Software Technology, Boeing Military Airplane Company

John K. Summers, Director, Washington C^3I Software Center, The MITRE Corporation

Executive Secretary: Dr. Arthur E. Salwin, Group Leader, System Studies, The MITRE Corporation

*Ms. Clapp replaced Richard J. Sylvester, Director of ESD/MITRE Software Center, The MITRE Corporation, who attended the organizational meeting.

How should the FAA judge the ability of a contractor to produce a project with over 400,000 lines of Ada code? Specifically address the support provided by, and experience with, the language, compiler, and run-time environment on host and target machines, as well as project experience, personnel issues, and management structure.

What tests should the FAA conduct to assess the maturity of the Ada technology for such a large system?

How can the FAA obtain early visibility into the risk issues associated with Ada development? Specifically address methodology and, if possible, the development and tracking of a risk management plan.

How should a software development standard, such as DOD-STD-2167, be tailored for a large Ada project?

FIGURE 1-1
QUESTIONS PROVIDED TO THE OUTSIDE EXPERTS

and did not participate in the study's final deliberations or in the preparation of this report. Furthermore, the conclusions of this report are those of the study participants, and do not necessarily reflect the views of any of their employers or of any individual expert. The study participants wish to thank the experts for their time and effort, their candor in discussing lessons learned, and their thoughtful responses to our questions.

1.3 Organization of this Report

Section 2 of the report provides information on the elements and advantages of an Ada software development approach. This material serves as the basis for the study's recommendations regarding the use of Ada on AAS. It also provides a context in which risks can be discussed. Section 3 identifies risk areas associated with an Ada development, while Section 4 provides general approaches for addressing these risks. Section 5 summarizes the near-term FAA activities that must be undertaken if the study recommendations are to be successfully achieved. Section 6 maps these recommendations into the four near-term task areas.

It should be noted that the report intentionally contains redundancies. For example, certain elements of an Ada software development are also cited as risk reduction activities.

2.0 CHARACTERIZING A SOFTWARE DEVELOPMENT

The use of the very best in software engineering is imperative for large, critical, and complex systems such as AAS, with high reliability requirements. Ada is a new programming language with many features designed specifically to stimulate and facilitate the use of good software engineering principles. A key characteristic of the Ada language is that it can be used to represent an understandable statement of the top-level system architecture and software design. It separates the "specifications" or definitions of processes and data from the "bodies" containing the code for performing the processing. It also encourages structuring of a system into modular units, which have been collected together into "packages" of logically related units. Packages are the basic building blocks of a system. Each has a single specification defining how that package can be used; this isolates the internal design and code of a package from the rest of the system design that uses the package. A user of the package does not have to know how it works.

Section 2.1 characterizes the study participants' consensus view of how a software development might proceed on large procurements such as AAS, and the elements it would contain. Material is included on how Ada can be used to support such a development process. Section 2.2 addresses the benefits of the described approach and indicates the advantages offered by Ada for achieving them. Based on these discussions, Section 2.3 provides qualified recommendations regarding the use of Ada on AAS.

It should be noted that Section 2 focuses on positive aspects of Ada usage. Section 3, on the other hand, will focus on the negative aspects, providing a discussion of Ada risks.

2.1 Elements of an Ada Software Development

As experience is gained in using Ada, managers report similar approaches to conducting an Ada software development for a large system such as AAS. It is the study participants' consensus that a composite of the elements described in this section appears to be a good way to achieve sound software engineering and utilize Ada for doing so. This development approach should not be regarded as the only way in which an Ada project can be carried out. Rather, it is a reasonable way to proceed. Furthermore, by documenting an approach, a framework is provided for the discussions on risks and risk reduction activities (see Sections 3 and 4).

2.1.1 Compilable Ada Program Design Language (PDL) Designs

A key characteristic of an Ada development is that the design is reflected in machine-analyzable Ada deliverables. In the following three subsections, "Ada PDL" and "top-level design" are defined, compilable designs are characterized, and their implications are discussed. Their advantages are given in Section 2.2.3.

2.1.1.1 Definitions. There are many varieties of PDL, ranging from structured English to a full programming language. The approach described here recommends the use of the full Ada language as PDL.

At each design level, Ada PDL contains the information appropriate for specifying that level of design, including data structures and interface definitions. Enough information is provided to enable the package specifications and bodies produced during a given design level to be used, without modification, later on as the code appropriate for that level. Only constructs in conformance with the Ada language standard (U.S. Department of Defense, 1983), are used in the Ada PDL so that the design will be compilable (The Institute of Electrical and Electronics Engineers, Inc., 1987, p. 13). Comments are used for supplementary information.

An Ada top-level design can be defined to include the following:

- All top-level package specifications
- The bodies corresponding to those specifications
- The next level of package specifications called out by the preceding two items

In practice, management judgment and flexibility are needed in determining what constitutes top-level design, since riskier and more complex parts of a system should be designed to greater depth earlier than the simpler parts should be. Thus a more operationally oriented definition might be that top-level design is completed when the identified modules can be independently coded; that is, they are sufficiently defined so that they can be turned over to programmers who are not knowledgeable about the total system design. Regardless of the level of detail provided during design, the product of this activity then becomes the top-level code for the coding phase. It should be reemphasized that this requires no translation of the design into a new format.

2.1.1.2 Use of Compilable Designs Characterized. Compilable designs are used as follows. Beginning with top-level design, all design decisions are documented in Ada PDL. The compiler is invoked periodically to ensure that the design is compilable. This implies early access to at

least portions of a mature support environment for these compilations (see Section 2.1.6). The compilations are more than just a syntax check. Rather, since Ada requires that all interfaces be consistently and rigorously defined, the compiler serves as a check on the interfaces, which are the key entities being defined during top-level design. As the design is repeatedly compiled--and, in some cases, executed with stubs--errors will be found and the process iterated until the design finally stabilizes.

2.1.1.3 Implications of Compilable Designs. The consequence of such a process is that many of the integration and test activities can be carried out at the front end, and can continue to be carried out through the design and coding phases. In fact, the distinction between design and coding becomes somewhat blurred, as essentially similar activities occur throughout the development process; only the level of detail changes. By contrast, in a traditional approach, integration activities are deferred until an integration phase later in the development process.

To accomplish stable Ada PDL designs, more time will be needed for requirements analysis and design. Even if the requirements analysis phase has been previously completed, the rigor of this approach often means that certain requirements analysis decisions must be revisited. The justification for this up front time and cost has been demonstrated in a need for less time and cost during integration, at which time more developers and more modules are impacted by design changes.

Developing compilable top-level designs is not "extra" or "wasted" activity. Regardless of the methodology used, the top level of the software will ultimately have to be developed. The objective in the suggested approach is to do this early, when it has the most beneficial impact.

2.1.2 Contractor Use of a Small, Expert Design Team Up Front

In any large software development, it is desirable to have a small design team of the best people up front. Critical decisions are made on the front end of a project, and a small team is needed to ensure that a complete and correct design has been coordinated across the entire system. The team must therefore have extensive software engineering and large system experience.

On an Ada software development, a small, expert design team is particularly important. The team produces the compilable designs. The designers must also be Ada experts, because Ada's richness supports a broader set of design strategies (e.g., tasking and object-oriented design).

Once the designers achieve a stable design, large-scale parallel coding efforts can proceed. A stable design helps ensure that changes made by the programmers are confined to their individual modules. This enables them to work in relative isolation from one another without negatively impacting other parts of the system. Even if a programmer writes code that violates the interface constraints for a module, the compile-time checks do not allow such violations to be entered into the baseline system; programmers cannot proceed until the interfaces are made consistent. This also implies that adequate software development facilities must be provided to support interactive use of the development environment by all of the programmers.

During these large-scale development activities, the small design team serves as the nucleus of expertise for the total system architecture. Having a perspective unavailable to the programmer analysts and coders, the team members determine the system impact of proposed changes and decide when/whether to implement them (see also Section 3.5.3 on commitment to retaining the team). The designers need to understand the lower levels of the software. However, they should be constrained from going into too much detail so each of them can mentally encompass a larger part of the system, helping to ensure a more correct design.

2.1.3 Prototyping and Incremental Development

Prototyping and incremental development are recommended activities whether or not Ada is used.

Prototyping can be defined as the experimental design/coding of various system functions for the purpose of evaluating the requirements, ascertaining design implications, or studying the effectiveness of the design in its interaction with humans. The code produced for such an endeavor is often discarded upon completion of the evaluation; an operational version must be produced for the actual system.

Whether or not Ada is involved, prototyping is generally encouraged on all software procurements, where it can provide useful information to reduce risks in full-scale development. On an Ada procurement, additional prototypes may be useful. These can investigate the impact of the language and the code generated for the target machine on the performance of the system, in turn enabling the degree of uncertainty and the risk associated with language and compiler to be assessed as early as possible. Even though such activities may require more up-front time before coding begins, it is felt that this is justified by the long-term benefits.

Incremental development can be defined as the development of subsets of full system functionality for the purpose of design control and feedback. Incremental development permits verification of the design at early stages, when any errors detected may be corrected more easily and

thoroughly than would be possible after the entire implementation has been completed. The result is an evolving product whose functionality and quality have been verified throughout the development, instead of remaining unknown until near the very end. Thus incremental development can be considered an aspect of risk reduction. Its product is often operational code that will be used in the final system.

The use of simulations is also to be encouraged. However, at this time, other languages may be a more appropriate vehicle for their development, even on an Ada procurement.

2.1.4 Different Milestones

The classical life cycle calls for major milestones that provide demarcations between various development activities. For example, Preliminary Design Review (PDR) separates top-level design from detailed design, while Critical Design Review (CDR) separates detailed design from coding. Yet these milestones provide a somewhat unnatural description of the actual progression of activities, since different parts of the project may proceed at different rates. In fact, it may be more desirable to carry certain high-risk areas forward through coding, while deferring design on low-risk, noncritical areas.

As mentioned in Section 2.1.1.3, rigid milestone distinctions can become somewhat blurred in the development process suggested in this report. Prototyping and incremental development result in different milestones from those found on a traditional software development effort, whether or not Ada is used. PDR becomes the critical review since it examines the compilable top-level designs, which serve as the basis for all further development activities. The use of incremental development means that a monolithic CDR is replaced with multiple individual design walk-throughs as various portions of the system become ready for review. For contractual reasons, a formal CDR may be still be held, but its characteristics differ from the traditional: it is used to review action items resulting from the individual design walk-throughs.

2.1.5 Different Deliverables

The products resulting from the new activities and milestones can also differ. Ada's readability does away with the need for some of the traditional design specifications and much of the manually written documentation. Instead, compilable Ada--whether it takes the form of PDL during design, or executable software during coding--is used throughout the life cycle to provide various levels of system decomposition. Thus each phase of the development process results in a machine-readable Ada product that is used by the developers during succeeding phases; the same product can be delivered to the customer for review. Consequently, similar tools and techniques are used throughout the life cycle.

A good software engineering technique for large systems is to provide supplementary information for use in conjunction with the PDL. For design, overview material such as Booch or Buhr diagrams (Booch, 1987 and Buhr, 1984) can be used. During coding, detailed information, such as material on unit testing and on the rationale for implementation decisions, is provided in software development folders maintained by the programmers; these folders should be kept up-to-date throughout the life cycle.

2.1.6 Development Environment

2.1.6.1 More Demands Placed on the Hardware. An Ada software development is characterized by high demands placed on the development environment. This is true even if run-time performance is not impacted by the use of Ada, since run-time performance is a characteristic of the target, and not the development, machines. For example, Ada compilers provide extensive compile-time interface checking capability not available with other language compilers, which contributes to the Ada compilers' being heavy users of computing resources. The interface checking can result in computationally intensive compilations. If the entire system is recompiled, week-long compile times are not uncommon. Furthermore, unlike other languages, Ada requires that these system-wide recompilations be carried out whenever top-level interfaces are redone. This is sometimes referred to as the "compilation ripple effect" and is unique to Ada; other languages allow individual modules to be compiled in isolation, and thereby fail to detect new errors that may have been introduced. Thus careful resource planning is needed to ensure that enough workstations with sufficient speed are available to carry out compilations in a reasonable time, and that adequate disk space is available for developers to have on-line access to tools, databases, and the compiler. The anticipated requirements for these resources are later tracked against actual usage in order to provide proper resource growth.

2.1.6.2 Early Access to the Software Support Environment Essential. Adequacy of resources also implies access to mature support environments early in the development process. For example, such access is needed to support the compilable design activities early in the life cycle. Early access is also needed to support training exercises in advance of system development. Because of the standardization provided by validated Ada compilers, training can be carried out on compilers other than those to be used on the actual development.

2.1.6.3 Comprehensive Toolset Needed. The comprehensiveness of tool functionality--on all machines during all phases of the development effort--is an important consideration. A set of integrated tools is needed in addition to the compiler.

During coding, it is desirable to have a tool that indicates the potential impact of compiling a given module (i.e., how many other modules will have to be recompiled because of interdependencies denoted by "with" statements). Managers can use such a tool to make informed decisions on whether changes should be incorporated or be deferred to minimize compiler ripple effects. Another way to minimize ripple effects is through the use of a sophisticated compiler: if a compiler is "smart" enough to realize the scope of the changes, then unnecessary recompilations can be avoided. (For example, if only the comments are changed in a given package specification, then no other package specifications need be recompiled, even those that "with" the specification undergoing the change.)

During integration and testing, coverage analyzers and performance analyzers are very useful. They can provide statistics on which program units have been executed for a specific set of test data, as well as performance information on the amount of execution time spent in each program unit. Target machine coverage analyzers that supplement such information with the identity of the callers of the invoked units are particularly attractive.

Interactive debuggers on the target machines are essential. The alternative--encountering problems on immature target environments, documenting them, and then trying to reproduce them on a more mature development environment--is a time-consuming process. This is true on all large software development projects. For an Ada project, the debuggers should support features unique to Ada (e.g., tasking).

Other tools are used throughout the life cycle. The library systems and configuration management tools must have the demonstrated capability of handling a large software development effort. Management tools must be provided as well, so that progress on system development can be tracked. For example, a simple capability that provides the number of compiled package specifications can be used as a management tool for plotting progress vs. time. Other management tools providing budget and feedback information are important. Tools that measure code complexity can help identify problems and aid in test planning.

2.1.7 Training

Since there is a general shortage of Ada experience, an Ada software development includes Ada training for all associated personnel, including managers, with a heavy emphasis on software engineering. This emphasis is needed because Ada supports new strategies (e.g., object-oriented design) and new software architectures (e.g., one built around tasking).

Management training is crucial because of the significant differences between an Ada development process and processes used with

other languages. Managers must be able to understand the new deliverables (see Section 2.1.5), which look different from the ones they are accustomed to. They need to understand what is new, and how to plan the necessary activities and allocate the appropriate resources (including people and computers). Knowledge of Ada's "generic" feature, for example, can permit planning for more reusable code. Most important, managers need to know how to tell whether things are going well or not, and what to do in the latter case.

Training is given to all personnel on the tools they will be using. For example, managers are trained on the use of budgeting and tracking tools, librarians on the use of configuration management tools, and coders on the use of compilers and interactive debuggers.

Since a considerable amount of time is required to learn Ada--estimates range from 6 to 18 months for programmers--sufficient lead time must be provided; such training should take place before contract award. Following formal courses, experience can be gained by development of the less sophisticated parts of the system (e.g., tools, benchmarks, prototypes, and test drivers).

2.1.8 Software Development Plan Document

A Software Development Plan, documenting the development methodology, should be required on any large software development project. However, for an Ada project, this plan has an Ada orientation, and is tailored to an Ada development process. For example, it reflects compilable Ada designs by small design teams, the new milestones, and new deliverables. Other items incorporated include an approach for designing reusable software, and an approach for identifying and incorporating available reusable software. The plan also includes risk reduction activities, such as the use of benchmarks, or is coordinated with Risk Management Plans calling for such activities.

2.1.9 Methodology

An important element of an Ada software development is a methodology for Ada language usage, clearly defining the design strategies chosen for a particular Ada development.

The methodology also provides guidance on the use of Ada language features. Such guidance might take the form of encouraging the use of some language features, constraining or discouraging the use of others, and prohibiting the use of yet others. Features that may be encouraged include private types (which encourage information hiding) and structured programming constructs. Features that a project might want to constrain (for certain applications or to certain individuals) include tasking, exceptions, and packages. Consideration may be given to prohibiting such

features as the "use" clause (Booch, 1987, p. 222 and Bryan, January, February 1987, pp. 25-28) and the "go to" statement. Prohibitions would in effect subset language usage, even though only validated compilers (not subset compilers) would be used on the project.

Also included are an approach for using non-Ada code if needed, and encouragement for the use of commercial off-the-shelf software where possible. If such software is used, the methodology should indicate how it will be tested, integrated into the system, and maintained.

2.2 Advantages of an Ada Software Development

Ada was designed to enforce, and provide easier implementation of, sound software engineering practices. Most of these practices have been promoted for many years. In effect, Ada's contribution is that, if properly used, it provides a natural vehicle for employing these practices. The reasons for this are discussed in the following subsections. However, it should be cautioned that the use of Ada does not inevitably lead to good software engineering; a sound development approach, such as the one described in the preceding section, must be followed as well.

2.2.1 More Effective Development Approach Stimulated

2.2.1.1 Richness of the Language. Ada is a very large and rich programming language. Packages separate interfaces from implementation and encapsulate relevant operations for an abstract object or type. Tasking allows real-time aspects of the design to be explicitly stated. Exceptions allow error conditions to be explicitly trapped. Generics allow programming templates to be created for reusable code. Strong typing is enforced both within and across modules. Separate compilations support bottom-up and top-down development approaches. At a more detailed level, one finds that all the structured programming constructs are supported. In fact, the syntax for these constructs is better in some respects than that found in other popular languages, such as Pascal. Precisions and ranges of numeric quantities can be specified, and the precisions of quantities resulting from numeric operations are precisely defined. Long variable names allow for readable code.

2.2.1.2 Rigor of Ada as a Design Notation. Ada provides many features not commonly found in other high order languages. This richness allows it to be used as a rigorous design notation as well as a programming language. For example, real-time and parallelism aspects of the design may be expressed through tasking. A significant advantage is that interface definitions can be separated from the underlying implementation mechanisms, and extensive abstraction of data and typing can be provided. Exceptions allow the processing of error conditions to be explicitly defined.

2.2.1.3 Compilable Designs. Compilable, machine-readable designs are a significant advantage of Ada. Because of their importance, they are addressed below in a separate subsection (2.2.3).

2.2.1.4 Shorter, More Focused Reviews. The multiple individual walk-throughs discussed in Section 2.1.4 are generally of shorter duration than a monolithic CDR would be. This is an effective management technique that has several advantages whether or not Ada is used. First, the walk-throughs can be more focused and productive. Second, they have a beneficial impact on the schedule. Since they are held when needed, the feedback from them is more timely. Once a review for a given portion of the system has been passed, related development activities can proceed. In a traditional review process, the work is suspended and awaits completion of the entire CDR. Furthermore, a traditional CDR monopolizes personnel and interrupts normal activities, thereby impacting the schedule unfavorably; it is also tiring for the personnel involved. As a result, there is implicit pressure for completion, which in turn means that some design deficiencies may be overlooked.

The benefits arising from shorter and more focused reviews are realized because of the thoroughness provided by automated compiler-checking of compilable designs. This checking is a significant improvement on time-consuming manual checks for "bookkeeping" items such as inconsistent naming conventions in English-oriented designs. Thus there is enough time to take several perspectives that focus on more significant issues, such as software growth and reusability.

2.2.1.5 Continuity of Activities Across Phases. Ada provides greater continuity of activities throughout the software development process. Manual translations of the products of one phase into the notation of the next (with the inevitable mistranslation errors) are no longer needed. This is due, in large part, to the richness of the language, which is sufficient to allow Ada to be used for top-level design, detailed design, and coding. Thus a continuity across project phases is achieved; rather than having a top-level design notation translated into detailed design, which is then translated into code, the same notation is used throughout. This similarity of activity means that, for the most part, the same tools and techniques can be used throughout the project. Thus there is more opportunity for resources to be devoted to achieving tool maturity rather than tool proliferation. In addition, having fewer tools means that developers can become proficient with a limited number of tools rather than trying to learn the use of many.

2.2.1.6 Better Traceability. The continuity of activities described above results in better traceability in two ways.

First, there is better traceability between the products of one phase and those of the succeeding ones throughout the development process. The machine-readable deliverables produced during a given phase are used by the developers during the next one. Since there is no longer a need to translate the products of one phase into those required for the next, there is no need for manual determinations of which derived requirements have been implemented at the next phase.

There is also better traceability between documentation delivered to the customer, and the system, as actually implemented, that the documentation purports to describe (see also Section 2.2.3.8).

2.2.2 Other Alternatives Less Attractive

The expectation is that a more effective development approach will result in lower life-cycle costs. This is because software readability and traceability will improve maintenance productivity and system evolution. When Ada systems now under development enter maintenance several years from now, sponsors will have access to Ada maintainers who will will then be in the mainstream of expertise on Government contracts. Conversely, if an older language is used, such expertise may be difficult to find at that time. Furthermore, Ada's support of sound software engineering practices should result in software with higher quality and more reliability.

Actual project experience shows that the expectations of a shorter, smoother integration phase can be realized. Unfortunately, because Ada technology is new, no large Ada systems are available for verifying the expected maintenance phase and full life-cycle advantages.

2.2.3 Advantages Provided by Compilable Designs

Compilable designs offer many significant advantages, as described below.

2.2.3.1 Better Interface Definitions. Ada has much more rigorous requirements for interface definitions than do other high order languages or PDLs. In Ada, detailed interface definitions must be explicitly set forth in the package specifications. The thoroughness of automated compiler checking ensures that these definitions will be used consistently.

2.2.3.2 Early Interface Checking and Error Detection. The use of Ada during the design phase means that thorough compiler checking uncovers many flaws early in the development process. It has been well documented that the earlier flaws are uncovered, the less costly they are to correct. For example, correcting a design deficiency during testing

is over an order of magnitude more costly than correcting it during the design phase (Boehm, 1981, p. 41).

Machine verification of early design, especially interfaces, is a significant aid to communication between the individual designers, or design teams on a large project. It forces more explicit statement of decisions, thus reducing the erroneous assumptions that can go undetected. Since only a small design team and a relatively small number of modules are impacted by the redesign/recompile ripple effects, multiple iterations are possible. Thus a high level of design stability is achieved before coding begins. This is a long-sought goal that has previously been difficult to attain.

2.2.3.3 Design Rigor. The use of Ada syntax, instead of English-like descriptions, provides increased rigor in the design process. Ada PDL provides all the advantages of other-language PDLs, and much more. These arise from the strong typing requirements, the use of package specifications, and the interface definitions. It should be noted that the latter is a key advantage offered by Ada over other PDLs; other high order languages do not require compiler checking of module interfaces. Since Ada PDL is used at all levels, the rigor is enforced by compiler checking throughout the design and coding process. Partial executions of the design and code can be used to enforce the rigor further.

2.2.3.4 Early Incremental Top-Down Testing. A compilable design enables incremental top-down testing to begin early. For a design written in English, there is nothing to test early in the life cycle; with Ada designs, actual code is available for early testing. The top-level Ada package specifications provide a framework within which new modules can be tested as they are created. In contrast, other methodologies often resort to bottom-up testing, even if top-down design is used. With Ada, as new pieces are brought together, they interface more smoothly. With other languages, pieces that work in isolation often do not interface properly when integration is attempted.

Thus the expectation has been that stable designs would provide a smoother integration phase. In fact, the history of recent Ada projects indicates that this is indeed true, and that significant cost savings are being realized.

2.2.3.5 Early Addressing of High-Risk Elements. The use of compilable designs and incremental top-down testing provides the advantage that high-risk elements can be addressed early in the framework of the total design. That is, certain threads of the design can be carried out in more detail, down to the coding of the lowest-level modules, while other, less risky, parts of the system are still defined only at a high level. This provides greater assurance that high-risk

elements will work than does an approach in which such elements are "bread-boarded" and executed in isolation. As an added benefit, the customer gets earlier insight into the design.

2.2.3.6 More Natural Progression from Design to Coding. As mentioned in Section 2.1.1.3, allowing for a more natural progression of activities on a software development can blur the various phases of the development process. This may at first be perceived as a drawback in that it will no longer be as easy to determine when top-level design ends and detailed design begins (or when detailed design ends and coding begins). In the past, this determination was somewhat simpler since the activities carried out during one phase were clearly distinct from those carried out in the next. In fact, this perceived drawback is readily addressed with a rethinking of milestones and is no drawback at all; the old rigid milestones were, in practice, artificially imposed on a more fluid development process.

2.2.3.7 Traceability Throughout the Design and Coding Phases. As discussed in Sections 2.2.1.5 and 2.2.1.6, compilable designs lead to better traceability across phases, and better traceability between documentation, deliverables, and the system.

2.2.3.8 Better Deliverables. The Ada PDL products of the various development phases can be transmitted to the customer as machine-readable deliverables. They offer several advantages over traditional products.

First, the deliverables are guaranteed to track the design; in fact, they are the design. More traditional alternatives to PDL result in multiple representations of the design. Even if these representations track one another initially, it has proved impossible to keep all of them up-to-date. With a single representation provided by Ada PDL, this should no longer be a problem.

Second, the deliverables can be more easily reviewed. Since they are in machine-readable format, automation can be used to assist in the review process.

Third, minimal extra effort is needed in generating the deliverables. They are produced as a natural outgrowth of the development process rather than as an adjunct to it. Furthermore, much of the required English-narrative deliverables can be automatically generated.

Finally, since the Ada language is the documentation, many software developers find it more natural to produce than English narratives.

2.2.4 The Realization of Ada Advantages

It should be noted that the advantages offered by Ada described above do not accrue to a project by default. They are available, but realizing them requires that the contractor and customer be well-trained in Ada technology, and that they act upon that training.

2.3 Using Ada on AAS: Recommendations

The study participants concluded that the size and complexity of AAS make software engineering, and not the programming language, the key issue. Moreover, if properly used, Ada best facilitates good software engineering practices and a sound software development process. Therefore, the study participants unanimously recommended that the FAA commit to Ada as the appropriate choice for AAS, given four strong qualifications to ensure that Ada's potential is realized:

- The software development process must be modified; the FAA and contractors must use an approach tailored to take advantage of modern software engineering practices and the support that Ada provides for them.

- It should be expected that appropriate selective use of non-Ada code within the framework of an Ada design may be necessary when Ada is found to be inadequate.

- Contractor readiness for an Ada software development must be evaluated.

- Positive risk reduction activities must be undertaken addressing the risk areas associated with Ada.

The large system nature of AAS makes it well suited for the advantages offered by Ada; no other language offers more advantages. If one of the likely alternatives--FORTRAN, C, or Pascal--were chosen, the large system problems posed by AAS would remain, but many of Ada's advantages would not be available for addressing them. Furthermore, if JOVIAL (the language used to code much of the existing system) were chosen, the FAA would likely be its only major user by the 1990s; JOVIAL's original sponsor, the Air Force, has switched to Ada.

At the same time, as noted above, the advantages offered by Ada will not accrue by default; positive steps must be taken to realize them. If such steps are not taken, or if risk reduction activities indicate significant problems (e.g., compilers are inadequate), then the advantages of Ada usage will be outweighed by the associated risks. In such a case, the recommendation to use Ada would not stand. If the FAA commits early to Ada, risks can be assessed earlier. Problem areas thereby uncovered

can be addressed in a timely fashion, or if need be, a fallback position can be pursued without wasted cost and effort having accrued.

2.4 Summary

An Ada software development process can differ in many significant ways from the one traditionally used on software. These differences enable the advantages of Ada to be fully realized, resulting in a more effective development approach and in lower life-cycle costs.

However, in order to realize these advantages on AAS, it is essential that the FAA modify the software development process, allow appropriate use of non-Ada code, evaluate contractor readiness, and undertake positive risk reduction activities. The latter, which should commence as soon as possible, is the topic of the next two sections of this report.

3.0 IDENTIFIED RISK AREAS

Although Ada offers the potential of significant advantages on AAS, its proper use can only diminish the risks, not eliminate them. Many of the risks arise from the newness of the Ada technology: staff must be trained in Ada, tools must be developed and mature, and there are no historical data for estimating cost and schedule. However, it will be noted that many of the risk items are not peculiar to Ada; they would apply even if another language were chosen as the implementation vehicle.

This section of the report addresses the risk areas identified by the study participants. Details are provided on the risk items within these areas, and appropriate risk reduction activities are recommended where possible. Both Ada-specific and general software engineering recommendations are provided, though the latter are couched in the context of using Ada. These recommendations can be used by both the FAA and the contractors: by the FAA as guidance in writing the Request for Proposal (RFP) and Statement of Work (SOW), evaluating proposals, and monitoring contracts; by the contractors as input to consider in establishing their development process.

It should also be noted that only the AAS procurement was considered in formulating these recommendations, and their applicability should thus be considered appropriate only in that context. Nonetheless, the general nature of the identified risks implies that the recommendations may have applicability in a larger context, and could serve as the starting point for formulating risk reduction activities on other large procurements.

3.1 Ada Performance Risks

Ada was designed to be used for meeting the performance requirements of real-time systems. However, risks remain in this area that need to be addressed if Ada is to be employed effectively. Two types of performance risks are considered--real-time and availability. It should be noted that these risks are likely to diminish over time as compilers mature.

3.1.1 Meeting Real-Time Requirements

AAS must meet many demanding real-time performance requirements. Indeed, as one study participant noted, a timely but imprecise indication of an impending midair collision would be far superior to a precise indication given after the fact.

Four real-time performance risk items were identified and are discussed below.

3.1.1.1 Impact of Tasking. The foremost real-time performance risk item identified was the use of the Ada tasking feature. Even though object code generated by Ada compilers is generally as efficient as that generated from other high order languages, industry experience leads to two concerns about tasking: its object code often has poor run-time performance, and there is a tendency to over-rely on it.

To address tasking risks, execution speeds of various tasking features should be measured on the target machines. Based on these measurements, contractor Risk Management Plans should provide a methodology for the use of tasking; should indicate how performance bottlenecks resulting from tasking will be avoided, and how they will be detected and attributed to tasking when they do occur; and should provide alternative paradigms that may have to be substituted for tasking.

Even ignoring its performance implications, Ada tasking has the same risks that apply to any other concurrency facility. The problems addressed by tasking are generally complex, and it is difficult to design solutions such that there is no possibility of deadlock, starvation, etc. among competing and cooperating tasks. Furthermore, tasking is one of the most difficult Ada features to learn. Therefore, there should be a methodology for its controlled use. Using tasking only where it is specifically needed can reduce the associated risks. When this is done, tasking expertise can be limited to a few senior designers for higher-level use, and need not be of concern to most of the coders as they implement lower levels of the code. Controlled use can also simplify debugging since tasking errors can be difficult to find and repair.

3.1.1.2 Impact of Storage Management. Industry experience indicates risks involved in meeting Ada's implicit need for the availability of a large contiguous memory. The use of overlay techniques on a 16-bit architecture may in all likelihood prove too restrictive. Therefore, hardware maturity would be evidenced by the availability of large contiguous memory. Moreover, the allocation and deallocation of storage for tasks and other dynamic entities is of concern.

To address these risks, run-time storage management (i.e., the allocation and deallocation of storage) and the amount of space required by the run-time support system should be checked. If this is not done, the amount of working storage for application software may be considerably less than was planned when the hardware memory was sized. It is further recommended that the Risk Management Plans use the results of such checks to show that the memory subsystems of the chosen hardware families are appropriate for Ada, or to indicate what work-arounds will be employed for overcoming the problems.

3.1.1.3 Impact of Run-Time Support System. Even if Ada's features, such as tasking, are used in an optimal fashion, and the compiler

generates efficient code for them, the run-time support system on the target machines poses a run-time performance risk. System services, and the methods for invoking them, must be able to meet real-time requirements. This is, of course, true for any real-time system; it is of particular concern on an Ada development because of the newness of, and lack of experience with, the Ada run-time support systems.

3.1.1.4 Impact of Under-Use or Over-Use of Non-Ada Code. As noted earlier, there was unanimous agreement that in spite of all of Ada's advantages, there may still be areas of AAS that will require the use of non-Ada code. Low-level display software (but not the graphics applications built on top of it), device drivers, and in-line expansion of assembly language bit manipulation routines were cited as possible areas where non-Ada code may be needed for meeting real-time requirements. Anticipating such needs in a Risk Management Plan should be considered a positive sign of contractor awareness. Furthermore, the proposed use of commercially available tools should be considered positively, whether or not such tools happen to be written in Ada.

On the other hand, there is also the risk that performance will be used as a convenient excuse for justifying an over-reliance on non-Ada code. Since experience indicates that it is very difficult to anticipate where performance bottlenecks will develop, benchmarks should be used to help make this determination. Another possible approach is to design the entire system in Ada, providing Ada code for all package specifications. Then in those areas where performance becomes a problem, the package bodies can be recoded in another language, with the result that the non-Ada code will be developed in the context of an Ada design.

To address the above risks, the contractors' plans should include methodologies for using non-Ada code if required. These methodologies should also document which other languages will be used for the non-Ada code, and should reflect performance issues, as well as the difficulty of interfacing non-Ada code to Ada. For example, in some instances there are indications that interfacing to high order languages may be more difficult than interfacing to assembly language. This leads to three recommendations on non-Ada language selection:

- The FAA waiver process for using non-Ada code should be flexible enough to allow its use when appropriate.

- If a contractor has already identified where non-Ada code will be needed, through the use of benchmarks or other activities, or has at least developed the benchmarks for doing so, this should be considered positively.

- The methodologies should provide for tool sets that can be used in determining where performance bottlenecks arise.

Demonstrations of the early availability of such tools and their level of sophistication should be part of the FAA's evaluation process.

3.1.1.5 Summary of Real-Time Performance Risks. As the discussion in the preceding subsections indicates, early risk reduction activities are needed to ensure that AAS real-time performance requirements can be met. These activities are needed to identify which risks are of serious concern in the hardware/software environment chosen by the contractor, whether or not Ada is used. If these concerns are not addressed by prior satisfactory experience with the same systems on similar projects, then benchmarks are needed. They should be used to determine how tasking and other paradigms are best employed, to identify memory constraints and work-arounds for them, to indicate where non-Ada code should be provided, and to determine whether the run-time support system can support large systems development.

3.1.2 Meeting System Availability Requirements

AAS has very stringent availability requirements, with only three seconds of down-time allowed per year. This is an area where Ada is seen as a strength. However, two aspects of the language--exceptions and elaboration--should be considered in designing the system, for they offer the potential to improve reliability. In addition, the run-time support subsystem, previously cited as a real-time performance risk, poses availability risks as well.

3.1.2.1 Impact of Exceptions. The first aspect of Ada to be considered with respect to reliability is the Ada exception feature, which is used to trap error conditions. In designing reliability into AAS, careful attention must be paid to such conditions, regardless of the chosen language. If Ada is used, the role for exceptions in handling these conditions must be considered. If all of the conditions are not considered, then Ada will perform default handling of the error, which may cause the error to propagate upward. This will result in loss of the local information needed for handling the error and could even stop the system. Note that such error effects are typical of other languages, which generally do not provide any exception-handling capability to the application programmer. Therefore, Ada provides the capability to eliminate system stoppage when an exception occurs, but there is considerable risk because it is very difficult to ensure that all possible exceptions have been properly anticipated.

Another concern regarding exceptions arises from compiler optimization, which can re-order, or even eliminate, statements in the object code. Thus the order in which exceptions are invoked, and the information that is available to the exception handlers, can be implementation-dependent. The result is a danger of unpredictable

behavior. Furthermore, if some of the testing takes place on the development machines, there is a risk that development and target machines will optimize differently, resulting in different execution time behavior for the same code. (See also Section 3.4.5 for other risks involving differences between development and target machines.)

The contractors should show evidence of fully understanding how exception handling works on their machines. Their methodology should address mechanisms for dealing with the cited concerns and for using exceptions. Exceptions should be used for true error conditions, and not as a convenient way to implement normal occurrences, such as end-of-file or loop termination.

3.1.2.2 Impact of Elaboration. The second aspect of Ada that should be considered with respect to reliability is elaboration, a run-time process that takes place when a program unit is invoked. Depending upon the error-recovery techniques employed in AAS, the elaboration time during restarts of the system could be too time-consuming. Unless the compiler provides pre-elaboration (i.e., the results of the elaboration execution are mostly attained before the software is loaded into the target), stringent availability requirements may not be met.

3.1.2.3 Impact of Run-Time Support System. The run-time support system on target machines poses availability risks since it is the foundation upon which application code is executed. If bugs contained within such systems cause the software to abort, then meeting AAS availability requirements may be at risk, regardless of how bug-free the application code may be. The newness of the Ada run-time systems, coupled with their required high level of sophistication (Ada real-time support systems, unlike those associated with other high order languages, are effectively mini operating systems), makes this a particular concern. Even the compilers pose risks because their relative newness increases the likelihood that object code will be incorrectly generated from correct source code. Thus it is important that contractors demonstrate the reliability of their support systems and be sufficiently familiar with them to find work-arounds when troubles arise. Demonstrated previous use of the same hardware/software environments on large projects should be considered a plus.

However, regardless of what precautions are taken, the newness of the Ada environments increases the likelihood of bugs. Therefore, it is important that the contractors have a viable, documented plan for working with vendors and having problems fixed in a timely manner, as they arise (see also Section 3.4.6).

3.2 Risks Arising from AAS Software Size

The largeness of the AAS software--hundreds of thousands of lines of code--results in its own risks. These often arise because the experiences gained on small systems simply do not scale up to larger amounts of software in a practical amount of time. Although such risks would also arise in non-Ada developments, the use of Ada makes them potentially more troublesome because of the newness of the Ada compilers, environments, and tools.

3.2.1 Compiler Limits--a Key Problem Area

The first software size risk, which has caused significant problems on other procurements, is the ability of the compiler to handle large amounts of code in the context of the chosen hardware/software environment. For example, a compiler may work quite well on a small- to medium-sized application, but run out of internal heap space without warning when a very large compilation is attempted. It is therefore essential that benchmarks be applied early for stressing the large system aspects of the chosen compiler, on exactly the same hardware/software environment that will be used for application code development.

Several options must be considered in choosing such benchmarks. First, it must be determined which ones will be chosen by the contractor and which by the FAA. For those chosen by the FAA, a previous large Ada project could be supplied. The contractors would make minimal modifications and then compile the benchmark to demonstrate that the compiler correctly handles large quantities of code. Another possibility for an FAA-supplied benchmark is a synthetic one whose overall characteristics (e.g., number of packages, number of procedures per package, nesting depth, interdependencies arising from "with" statements) are similar to those anticipated for AAS. Since such a benchmark is intended as a compilation test and not as a run-time test, stubs could be provided for the package bodies. In some cases, these stubs could be constructed in such a way that they would stress the compiler for memory space or for time.

Even if the compiler has no internal size limitations, its speed may result in de facto limits as to how much code can be handled. The same is true for the linker, library manager, and other tools: they may be too slow to handle large volumes of code. Thus benchmarks are needed on all these development tools to ascertain how quickly they can handle large volumes of code.

Other types of compiler limits manifest themselves at run time. For example, a compiler implementation may support a limited amount of dynamically spawned tasks. These limits should be known early in the

procurement, so that designs can be constrained to remain within these limits, or steps taken to increase them.

3.2.2 Resource Adequacy of Development and Target Machines

The adequacy of computing resources is another risk because Ada developments typically require more development resources than do non-Ada ones, partly because of the large amounts of checking that must be performed by the compiler. (This should not be confused with the efficiency of the generated code; in fact, the more efficient the generated code is on the target machines, the more computing resources are required to produce it on the development machines.) In effect, one is trading off large personnel costs and time at integration for more complete computer checking throughout the software development. Machine time is traded for human labor, and reliability is gained in the process.

An important aspect of resource adequacy is how the contractors intend to proceed with their designs. Because of the interdependencies among modules arising from the use of "with" statements, a small system change can result in large recompilation ripple effects; total recompilations of a week's duration or longer have been observed for very large systems with slow compilers. Early benchmarks should be carried out indicating the time required for such recompilations. Furthermore, the development methodology should address this issue and display an awareness of the need for stable top-level designs before larger staffs are brought on board for parallel development activities: a week of idle time forced by total recompilation is more costly the larger the team size. Other issues to be addressed by the methodology include the necessity for deferring some design changes until the next major recompilation, the use of incremental compilers, and the availability of tools that indicate what the effects of a particular change will be. This information, combined with compilation speed benchmark results, enables management to make informed decisions about proceeding with design changes or deferring them.

There is also a need to address the adequacy of computing resources on host and target machines. One concern, for example, is that sufficient disk space must be provided to hold the anticipated number of system releases (including source code, object code, and on-line documentation) that will be required at any given time. By comparing the planned resource usage with that actually encountered, controlled resource growth over the project life cycle can be achieved. The adequacy of the computing resources for the number of development personnel and the anticipated number of recompilations must be considered as well. This includes central processing unit (CPU) speeds, the amount of memory, and the number of workstations to be made available. Again, tracking of projections with actual experience provides the mechanism for informed growth planning.

3.3 FAA Readiness for Contract Monitoring

Considering the newness of Ada technology and the many differences inherent in an Ada development process, there is a risk on any Ada procurement that the customer will not be ready to award the contract or to monitor and manage the project. An understanding of Ada and software engineering is needed so that contractor and customer can communicate, with the result that the activities, the design notation, and the items being tracked (e.g., number of package specifications) have meaning for the customer. Furthermore, there is a risk that lack of readiness could jeopardize the development process. For example, unnecessary deliverables could be called for, or a contractor could be unfairly penalized for identifying and proposing positive risk reduction activities.

Contractor readiness is also needed to make informed decisions about waivers for the use of non-Ada code (see Section 3.1.1.4), and to tailor DOD-STD-2167. In the latter case, there is a risk of not allowing the contractor to tailor this standard towards an Ada development process.

3.4 Software Support System Risks

Software support systems are the foundation upon which code is developed and executed. Thus they are a key concern--on both the development and target machines. Always a risk item, they are especially so on an Ada development because of their newness. This is particularly true on the target machines, since vendors often put most of their emphasis on the development environment; support for the target machine may be limited to a cross-compiler back-end for their product line. The result can be a "bare-bones" target environment not suitable for testing and running a large system.

Indeed, the history of Ada indicates that a likely problem area is the unavailability of environments with adequate maturity; this has been the cause of failures on past projects. Therefore, an early assessment, such as through the use of benchmarking, is made of the environments. If they prove to be unacceptable, there is still time to find alternatives; if their inadequacies are manageable, steps can be taken to find work-arounds or to have the environments mature on the job through corrective measures.

Guidance in addressing environment risks is available. For example, the Software Engineering Institute is working on criteria for evaluating environments, and the National Aeronautics and Space Administration (NASA) Space Shuttle program has undertaken development of its own Software Support Environment.

3.4.1 Functionality

There is a risk that insufficient functionality will be available in the support environment. This risk pertains whether or not Ada is used. In fact, incomplete toolsets are a common problem in the software industry. On an Ada development, this may be especially true since the newness of the technology means inadequate time may have been available for developing the toolsets. (See also Section 2.1.6.3 for a characterization of an Ada toolset.)

3.4.2 Performance

As discussed in Sections 3.1.1.3 and 3.1.2.3, the support systems present risks for meeting real-time and availability requirements.

3.4.3 Maturity

Ada technology has clearly advanced to the point where production quality compilers are available for use on a wide range of computer architectures. Nonetheless, the maturity of a compiler and its associated tools and environment remains a concern for any particular hardware suite; overall maturity of the technology is of little solace if the technology is unavailable, or is not of production quality, on the hardware chosen for the application. "Maturity" is hard to define, but implies, at a minimum, being bug-free, having the ability to work with large amounts of code, and offering a comprehensive set of tools. Validation of a compiler is necessary, but validation alone is insufficient to ensure production quality.

A mechanism for handling exponential compile time growth would be considered evidence of maturity. Elaboration should be optimized for real-time systems. A demonstrated mechanism or technique for interfacing with other languages should be available.

Since maturity cannot be achieved rapidly (e.g., several years are needed for compilers to attain maturity), prior successful use of the exact hardware/software environment on other large projects is advantageous. If this experience is not available, then other techniques (e.g., benchmarks) will be needed. Early availability (see Section 3.4.4) and contractor preparedness for modifications (see Section 3.4.6) will also allow maturity to be achieved.

3.4.4 Early Availability on Development and Target Machines

The early availability of a run-time support system, on both target and host machines, is essential. Although such availability is strongly recommended regardless of what language is used, it is especially important for Ada: because of the difficulty in producing Ada compilers,

there is a history of projects never achieving access to satisfactory compilers and support systems for the chosen hardware/software environments. Therefore, an early demonstration of such availability should be required of the contractors. Such demonstration should include satisfactory results from the compiler sizing benchmarks discussed in Section 3.2.1.

3.4.5 Portability

Although it is not anticipated that AAS code will be reused on other projects, two types of portability within AAS are of concern. The first of these--the ability to transfer code from development to target machines--is essential if some of the testing takes place on development machines. The second type--the ability to transfer code from one major subsystem (e.g., the Initial Sector Suite System [ISSS]) and reuse it on another (e.g., the Tower Control Computer Complex [TCCC])--is desirable.

Many Ada features were designed to enhance portability, and the use of a validation suite was intended to enforce it. Indeed, Ada is more portable than other high order languages. Nonetheless, operating system dependencies remain, and Chapter 13 of the Ada language reference manual (U.S. Department of Defense, 1983), allows for machine dependencies. For example, code optimization schemes used by the compiler may result in different run-time behavior for exceptions on development and target machines. The contractors should demonstrate an awareness of these differences, perhaps through the use of benchmarks.

In order to ensure that the developed code will run on the target machines, the contractors should provide an early assessment of the commonality of Ada's implementation-specific aspects on target and development machines. The results of such an assessment should be incorporated in their methodology guidelines on minimizing the use of implementation-specific features. Where such features are necessary, their use should be limited to the common intersection only.

In some cases, such an assessment may indicate that an essential feature is lacking. If this is so, then early access to such information allows the contractor to have this feature developed in a timely fashion, or to find alternative compilers. The commonality assessment can also address the impact of optimization on such items as exception handling (see Section 3.1.2.1), testing, and program correctness.

3.4.6 Contractor Preparedness for Inevitable Modifications

Early demonstrations of tool capabilities, such as through the use of benchmarks, are considered important since surprise limitations late in the life cycle can have disastrous consequences. However, it is realized that, even given very good early risk reduction awareness and

associated activities, some tailoring of the environments to the system being developed is inevitable on systems as large as AAS. (This is true whether or not Ada is employed.) Thus the contractors should document in their plans an approach for addressing problems when they arise, and should allocate sufficient time and money resources for proceeding with the documented approach. One attractive solution, which has been used previously, is to have all tool and environment vendors under subcontract on the procurement, in order to ensure responsiveness. It may even be desirable to provide them with work areas on the contractor's development system so they can verify and fix bugs quickly. This is preferable to a cumbersome Problem Trouble Reporting procedure in which difficulties must be overcome in reproducing bugs on a different system before the bugs can be fixed.

To summarize, an indication that troubles are anticipated in the support environments should not be considered a weakness in the proposals, but rather realistic addressing of a risk. This is not a contradiction to the early availability recommendation in Section 3.4.4: enabling existing environments/compilers to mature is acceptable; developing new ones from scratch is not.

3.5 Personnel Risks

Several risks regarding contractor personnel have been identified, as discussed below.

3.5.1 Ada, Software Engineering, and Large System Experience

For the contractors to proceed successfully with AAS, it is necessary that staff have sufficient experience and training. Software engineering training is needed--not just Ada training. The Ada experts repeatedly stressed to the study participants that such training is needed for management personnel as well as for development personnel. (See also Section 3.3 on the importance of a prepared customer.)

Sufficient lead time must be allowed for the requisite expertise to be acquired. For example, estimates range from six months to several years for the time needed to get programmers up to speed on Ada and the associated methodologies. Thus a one-week training course after contract award, while better than nothing, does not suffice.

3.5.2 Staffing Profile with a Small Experienced Front End

In order to achieve the full benefits of a sound software engineering approach, including a stable top-level design with well-defined interfaces, a small team of experienced designers is needed up front. Again, sufficient lead-time is needed for bringing this team on board, and more than Ada experience is required. Judicious use of such people up front

allows successful deployment of a much larger team of less experienced personnel later in the life cycle.

3.5.3 Commitment to AAS and Retaining the Team

The contractors should indicate a commitment to retaining key personnel needed to exercise control over the design and implementation throughout the project (see Section 2.1.2). For example, it is important that the participants in a software engineering exercise (see Section 4.3) continue to play key roles throughout the development process. Because of the relative scarcity of Ada personnel, attrition issues, always a problem on software development, are particularly acute. The contractor needs to indicate what incentives will be used to retain key personnel and how personnel will be replaced if lost.

3.5.4 Subcontracting Approach

The relationship of the contractor to its subcontractor personnel is important. All training and experience requirements that apply to the contractor apply equally to the subcontractors. The latter should, moreover, be as proficient as the contractor in the chosen software development methodology, and their use of it should be enforced. Software development methodologies should show how the subcontractors are truly part of the development team (e.g., indicate their participation in the design and review process).

The contractor/subcontractor relationship is of concern on any procurement. The newness of Ada methodologies that allow full realization of the Ada potential makes this relationship of particular concern on an Ada development effort.

3.5.5 Commitment to Tool Usage, Including Management Tools

A sophisticated toolset is no advantage if it goes unused. For example, simply dropping a six-inch user manual on a manager's desk will not be sufficient to guarantee use of a project tracking tool. Training can be considered evidence of a commitment to using the provided tools. Another indicator is a track record of prior usage, which also helps ensure tool maturity.

3.6 Management Risks

Strong, disciplined management is essential for success in any large software development process. For example, all personnel must be familiar with, and use, the chosen methodology. The managers must understand the methodology well enough to enforce it, and to permit refinements when, but only when, they are necessary. As new personnel are brought into the project, they, too, must become part of the project culture.

One of the advantages of an Ada development process is that prototyping and the rigorous design notation will make design flaws more readily apparent earlier in the development process. However, realizing the benefit of this potential advantage requires that the contractor have flexible management to act upon this information. As work at lower levels of the project reveals top-level design flaws, management must be willing to back up and correct the higher-level problems. Flexible management is also needed to forgo the use of the design methodology in the limited number of cases where it is not applicable; for example, object-oriented design may not be appropriate in implementing well-defined communications protocols. Furthermore, nice-to-have, but inessential, design changes may have to be deferred to reduce the number of major recompilations. Management flexibility is also needed in approaching the use of non-Ada code. For example, even though guidelines may suggest the use of C over assembly language for implementing a certain function, the use of assembly language may be necessary if productivity will suffer because of difficulties in interfacing the C code to Ada.

Implicit in the preceding discussion on the need for flexible contractor management is that FAA procedures allow for this flexibility to occur without the contractors' expending undue effort in gaining permission. Even if the contractors have properly approached the use of non-Ada code, they may perceive an implied stigma in asking the FAA for waivers from using a single language. Also, as noted in Section 3.1.1.4, the FAA waiver process for using non-Ada code should be flexible enough to allow it to be invoked when appropriate. Moreover, timely FAA responses to requests for waivers should be given. Thus the FAA may want to consider the use of general guidelines rather than case-by-case waivers.

3.7 Schedule and Cost Risks

3.7.1 No Historical Data Available

Schedule and cost risks, always high on a software procurement, are increased for an Ada development. The newness of Ada means that no major procurements have gone through an entire Ada development process, including the maintenance phase. Thus there is no familiar application or environment that can be used for guidance in generating AAS cost and schedule estimates. This is the foremost cost and schedule risk resulting from the use of Ada.

3.7.2 Lines of Code a Poor Estimation Technique

Software estimation techniques have traditionally centered around lines of code estimates. There has always been controversy as to what constitutes a "line of code": Should comments be included? blank lines? data statements or only executable lines of code? Unfortunately,

the situation is worse in Ada. The most apparent, but probably least important, complication is that there is not a one-to-one mapping between Ada statements and source lines. For example, an if-then-else statement can span multiple lines; conversely, multiple statements can appear on the same line, although this practice is generally frowned upon. An easy solution to this problem is simply to count semicolons. However, this solution ignores more significant issues. The extreme strong typing in Ada requires that all data structures be clearly laid out in "type" statements and that all variables be declared. This is not done in some of the older languages such as FORTRAN and JOVIAL. The cost and schedule effects of producing such statements, relative to the effects of producing a line of executable code, have yet to be determined. Fortunately, past experiences in Pascal may provide guidance in this area. Conventions, such as those provided in Boehm, 1981 (p. 59), should be established for counting lines of code. Whatever convention is chosen, its consistent use throughout the project life cycle is essential.

A more significant issue is how to count package specifications, since there are no corresponding constructs in other languages. At first glance, specifications may seem somewhat insignificant since, in size, they may be only one-quarter to one-third as large as the corresponding package bodies. However, this perception is deceptive, because they are more difficult to write. This, in turn, is because they embody important design aspects such as encapsulation, information hiding, and data abstraction, and even form the structure of the overall software architecture. In fact, they may often be written by the design team early in the project, and are thus not produced during the formal coding process. Furthermore, some of the information in package specifications is repeated if there is a corresponding body. Should such code be counted doubly? Again, there is no right answer here, but at a minimum the contractors should show awareness of the problems and provide an intelligent means for addressing them. Counting the semicolons may be a naive approach.

By far the most serious impact on estimation techniques is the use of Ada generics. When properly used, generics are a boon to productivity, since a single line of code can be used to tailor and reuse ("instantiate") a code template generated elsewhere on the project. How such instantiated code should be counted is an open question--as a single line of code, or as the perhaps thousands of lines that it generates? Even writing the generic template poses estimation problems, since writing such a template takes somewhat more effort than writing the equivalent nongeneric version; the payoff comes in its reuse. Because of these difficulties, on some previous Ada procurements, contractors used other estimates (e.g., staff months), and then "backed out" an equivalent line of code measure corresponding to the the measure being used. Such "lines of code" might correspond to the number that would have been produced on, say, a FORTRAN project; however, they do not necessarily

correspond to those actually produced on an Ada procurement. Unfortunately, we have no experience to serve as a basis for mapping FORTRAN lines of code into those for Ada, or for determining the relative amounts of effort needed to produce them.

In summary, the limited past history with Ada means that there is no prior project experience to draw on; such historical data are the cornerstone of popularly used commercial models. In lieu of this experience database, one is forced to resort to analogous experience with other languages. However, the more significant the estimation problem becomes, the less guidance is available in relating this problem to experiences gained with other software development languages. Nonetheless, if Ada is to be used on AAS, the FAA may want to consider alternative measurement techniques. Ada-oriented measures, such as number of completed package specifications or bodies, may be more appropriate. The advantage of using such Ada-tailored techniques is that the same ones can be employed throughout the life cycle, including the design phase, to derive early quantitative measures of progress.

3.7.3 Impact of Ada Methodology and Milestones

The revised milestones resulting from incremental development provide a natural development progression, reflecting the fact that different parts of the system achieve maturity and completion at different times. However, this approach tends to blur the distinction between various phases. Again, there are no historical data available to indicate the resultant cost and schedule impact.

3.7.4 Tracking

The ability to track a project is essential since such feedback can be used to provide early warning of needed changes. Past experience indicates that far too often, both contractor and customer fail to track progress and update costs based upon that tracking. Failure to track could be especially troublesome on an Ada development, because the lack of prior experience makes initial estimates less reliable.

Thus it is strongly recommended that both contractor and customer commit to a tracking process that will enable them to learn from the experience database being generated throughout the procurement. At all times, indicators of planned vs. actual progress should be available. Machine-readable deliverables provide the opportunity for doing this with the assistance of automation; for example, the number of package specifications (planned and actual) could be tracked vs. time.

In summary, to realize the tracking potential offered by an Ada-oriented development requires the following:

- That the proper measures be defined
- That the proper tools for providing these measures be developed
- That both contractor and customer commit to using these tools in a tracking process

3.8 Other Risk Areas

Several other risk areas have been identified, as discussed below.

3.8.1 User Interface and Graphics

Acceptance of the finished AAS product by air traffic control personnel is essential for its success. Achieving such acceptance, and meeting the goal of increased controller productivity, requires an outstanding user interface to the system, regardless of the language used to implement it. Such interface development requires multiple iterations with the controllers in the loop. Thus the use of prototyping is strongly encouraged in this area. As is noted earlier in Section 3.6, management flexibility is the key; the ability to back up and address mistakes is particularly crucial. One possible impediment to such flexibility would be the use of a fixed-cost contract (see Sections 5.5 and 5.6). Since a requirement such as "be user friendly" is difficult to define precisely, the use of a fixed-cost approach in this area can easily lead to a "design to cost" solution.

Another risk in this area is the question of whether Ada is the most appropriate language for implementing graphics; as assessment may be needed to determine whether another high order language or assembly language would be more appropriate, at least for the low-level routines such as device drivers.

3.8.2 Ada Orientation of Design Specifications

One risk is that the design specifications will not be suited to an Ada procurement. The RFP should allow for, and the contractors should propose, machine-readable deliverables that are a natural consequence of a compilable design process. Other types of deliverables could be worse than useless; not only are they costly, but their production detracts from the more meaningful work that is needed.

3.8.3 Ada Orientation of Software Development Plans

The methodologies proposed in the Software Development Plans should show cognizance of, and exploit, the many differences that will result with an Ada procurement. A "business as usual" proposal will mean that

all of Ada's risks will be inherent in the procurement without the offsetting advantages.

3.9 Risk Summary

The proper use of Ada can reduce some of the risks on a large software development described in this section. However, the residual risks that remain must be addressed. This section of the report has focused on specific risk items organized into eight general areas. Wherever possible, solutions addressing the specific risk items have been proposed. In Section 4, some general risk reduction activities are recommended for addressing risks in a broader context.

4.0 ADDRESSING THE RISKS

This section of the report recommends five general risk reduction activities: Risk Management Plans, Software Development Plans, a software engineering exercise, compilable designs, and benchmarks. If properly carried out, they can minimize the effects of the risks identified in Section 3.

4.1 Require Contractors to Develop Risk Management Plans

Recommendation: The contractors should be required to submit Risk Management Plans as part of their proposals.

The Risk Management Plans should address each of the risk items identified in Section 3 of this report; in addition, they should identify and address other risks. For each item, the following material should be included in the plans:

- Documentation of the item, including an indication of its significance
- A risk resolution approach, including milestones and schedules
- Assignment of responsibility to individuals or organizations
- Identification of the resources (personnel, computing, and other) required for addressing the risk item

The plans for the individual items should be coordinated with each other and with the overall Software Development Plan.

4.2 Require Contractors to Develop Software Development Plans

Recommendation: The contractors should be required to submit Ada-oriented Software Development Plans as part of their proposals.

Any program as large and complex as AAS should require a Software Development Plan, encompassing material beyond the scope of this study. Nonetheless, the plan should indicate how Ada will affect the software development process and should include the following Ada-oriented material:

- A methodology showing how Ada is used, including the following:
  - Use of the the language's tasking, exception, and generic features
  - An approach for handling the ripple effects arising from redesign
  - An approach towards total recompilations
- A substantial tailoring of DOD-STD-2167, if that standard is used on AAS
- An emphasis on the use of benchmarks, prototyping, and incremental development, with a clear indication of the relationship between the results of these activities and the overall methodology
- An approach for using non-Ada code (see Section 3.1.1.4 for specific recommendations)
- A discussion on the use of Ada-oriented tools
- A method for transferring the Ada technology and methodologies to all the subcontractors involved in the software development effort
- An approach for reusable software that includes the following:
  - Development of reusable software
  - Identification and incorporation of available reusable software

To summarize, the Software Development Plan should contain material that characterizes a sound development approach, such as the one described in Section 2 of this report. Its contents should be familiar to all development personnel, and its use should be enforced throughout the software development.

4.3 Conduct a Software Engineering Exercise

Recommendation: A software engineering exercise should be conducted.

A software engineering exercise (sometimes informally termed a "contractor take-home exam") is a novel approach to risk reduction. The recommendation for its use is based on initial indications of its viability and success.

Briefly, in conducting a software engineering exercise, the FAA would define a software problem that the contractors would then be asked to solve. In carrying out the exercise, the contractors would be required to use their AAS personnel, their software methodologies, and their toolsets. More details follow.

The study participants recognize that they are not familiar with the detailed software progress to date by the design competition contractors, the software related deliverables provided during the course of the competition, or the detailed plans for expected deliverables to the time of contract award. As a result, the actual performance of the software exercise by the contractors may vary from the description below, but the intent of the exercise is very significant, and its conduct should be required by the FAA.

4.3.1 Description of the Exercise

A software engineering exercise would be carried out as described below.

4.3.1.1 Preparing the Exercise. First, the FAA would prepare a short problem statement, perhaps four pages, defining the requirements for a relatively small-scale software development effort. It is important that such a problem statement be tailored towards the AAS application, addressing some of the perceived risks of AAS. In fact, if properly constructed, the exercise could have spin-offs, perhaps a prototype, that would have been developed anyway, somewhere else on the AAS procurement. Thus, problem statements written for exercises on other procurements are probably not appropriate for AAS.

It should be noted that preparing the problem statement would not be an easy undertaking. If the problem were too simple, the results could be misleading: the contractor could use a few expert software engineers for the exercise, whose performance would not be indicative of what could be expected from the entire contracting team on the AAS procurement. Conversely, if the exercise were too complex, then little progress would be made on it during the limited time allowed. (Even worse, so much time and effort could be devoted to the exercise that it would detract from the overall AAS development effort.)

To best ensure that the problem statement is of proper scope, it is strongly recommended that an FAA team dry run the exercise before presenting it to the contractors. Doing so could uncover defects in the problem statement, so it could be revised before presentation to the contractors. A dry run could also provide the FAA with insight into the problem, allowing for better evaluation of the contractors' results.

4.3.1.2 Conducting the Exercise. The contractors would be given the problem statement and asked to carry it out. Only a limited amount of time, perhaps three weeks, would be allotted for this purpose. To obtain meaningful results, it is essential that the exercise employ key AAS personnel who will be involved on the actual procurement, as well as the software development methodologies and toolsets defined in the Software Development Plan. Typically, ten contractor staff members, including subcontractor representation, would be involved.

During the exercise, the contractors' activities would be in the form of an inverted triangle. That is, compilable Ada would be used to develop the following: a top-level design and software architecture for the entire problem, a partial detailed design, and code for perhaps one key thread. The entire effort would be documented in accordance with AAS documentation standards.

4.3.1.3 Timing of the Exercise. The exercise would be carried out after proposal submission, but before contract award. Thus, the contractors would not have to split their team between two activities--responding to the RFP and carrying out the exercise. Furthermore, the software engineering exercise activities would then dovetail nicely with other procurement activities: while the contractors are preparing their responses to the RFP, the FAA can be preparing the exercise. Then while the FAA is performing technical evaluation of the proposals, the contractors can be carrying out the exercise.

It is suggested that the exercise be required as an adjunct to the RFP response; however, it could be FAA-funded as part of the DCP risk reduction activities taking place between April 1987 and the Acquisition Phase contract award.

4.3.1.4 Evaluating the Exercise. The final step of the exercise would be an FAA evaluation of the exercise results, in accordance with AAS contract monitoring procedures.

4.3.2 Results of the Exercise

A software engineering exercise would have several results. First, it would provide a test of the contractors' planned methodologies; this is because the contractors would be required to adhere to those methodologies in carrying out the exercise. Second, it would provide a test of the contractors' ability to design and implement software using their methodologies. Third, the results could be used by the FAA as technical criteria for assessment in making the Acquisition Phase contract award, as an adjunct to the technical evaluation of the proposals. Finally, the exercise could result in an improved Software Development Plan; any contractor activities deviating from those laid out in the plan during this, its initial use, would have to be documented.

4.3.3 Benefits of the Exercise

There are many benefits to be realized from conducting a software engineering exercise.

First, by requiring that the contractors adhere to their Software Development Plans, the FAA would, in effect, be requiring them to turn "dead" paper plans into living methodologies. In so doing, the contractors would undoubtedly find flaws in those plans. Thus, necessary revisions to the plans should be allowed, provided that such revisions are well documented and become part of the final Software Development Plans used on AAS. This would result in better, partially tested, methodologies for use on the actual procurement. Similarly, problems with the various toolsets could be identified early, thereby removing them from the critical path of the AAS development effort.

Another benefit resulting from the exercise is that contractor and subcontractor personnel would be forced to become familiar with the methodologies and tools they had proposed for use on AAS. Thus the exercise team members would become experts who would serve as an important nucleus of expertise for the larger procurement effort. The exercise would also serve as a partial mechanism for keeping the integrated software team together and current on AAS during the time between proposal submission and contract award.

A corresponding benefit would result for FAA personnel: based on their experiences in evaluating the exercise results, the FAA would gain familiarity with its own contract monitoring procedures, and could modify them if necessary. As part of this evaluation, the FAA could assess the adequacy of the deliverable requirements, and determine whether they should be altered. A spin-off benefit of the exercise is that an FAA dry-run of it would provide valuable training in software engineering and in Ada.

Another benefit is that the exercise results would provide the FAA with an opportunity to observe how the two contractors perform on a real-world problem. Proposing compilable PDL designs is desirable, but actually producing them on such a problem would provide greater assurances.

Finally, FAA review of the exercise would indicate whether the Software Development Plans were followed and how many revisions were needed to carry out the exercise. This could, in turn, indicate the viability of the plans and of the contractors' commitment to using them.

4.4 Require Compilable Designs

Recommendation: The AAS contractors should be required to develop compilable Ada designs for their proposed systems and deliver them to the FAA in machine-readable format.

As discussed in Section 2, compilable designs are an essential part of an Ada software development. They may also be considered part of the risk reduction activities. Therefore, this recommendation on compilable designs stands even if it means that some or all of the existing designs must be redone.

A corollary to this recommendation is that the FAA should thoroughly review the Ada designs. This review should include verification of compilability, and a check of how much of the design appears in comments as opposed to how much is expressed in Ada. In effect, an Ada-oriented PDR is called for. The size of this activity is not necessarily as large as it may at first appear. Because of the compactness and rigor of Ada notation, it is anticipated that a redesign in Ada would be considerably more compact than any corresponding English-oriented designs that may previously have been produced. Furthermore, if a moderately sophisticated toolset were employed, much of the deliverable documentation could be generated automatically from the Ada designs, and would not have to be regenerated.

As noted earlier, compilable top-level designs are considered a big plus. Their advantages include providing evidence of the contractors' ability to perform, providing early consistency checking of the design, and providing the opportunity for the use of tools as part of the FAA reviews. For more details on these and other advantages, see Section 2.2.3.

4.5 Benchmarks Should be Used

Recommendation: Benchmarks should be required for use in addressing many of the identified risks.

Benchmarks were previously suggested as a means of achieving risk reduction for many of the items discussed in Section 3. For the contractors, they identify problem areas, which can then be remedied; for the FAA, they provide an indication of contractor readiness. Because of their importance, benchmarks are discussed in some detail in the following three subsections.

4.5.1 Use of Benchmarks on Development Machines

On the development machines, early application of benchmarks is needed to address the large system size aspects of the AAS procurement.

For example, the compilers should be stressed to their limits to find out how large a system they can successfully compile.

Similarly, the library and configuration management tools should be stressed to determine their adequacy for a large system procurement. Since there will be a large team of developers, the tool assessment should include tests of simultaneous access by multiple users.

Early benchmarks demonstrating the functionality of the support environment should be carried out as well. These can be used to generate changes to the system in time for development activities.

Benchmarks measuring performance are needed for ascertaining how much code can be compiled within a given amount of time. Measurements on disk space, and on memory and CPU utilization, are needed. A determination can thereby be made as to the adequacy of the hardware resources assigned to development personnel.

4.5.2 Use of Benchmarks on Target Machines

The large system aspect of AAS should also be addressed by target machine benchmarks. For example, if spawned tasks are to be used, then the run-time limit of the number of such tasks that can be run on the target machine must be ascertained, and the system design constrained to be within those limits. Similarly, memory limitations should be ascertained.

Target machines frequently have only "bare-bones" environments, as contrasted with the often much richer environments on the development machines. Thus benchmarks addressing the adequacy of the development machine toolset are needed.

AAS must run under the framework of the target machine run-time support system. Thus this system poses risks with regard to both performance and reliability, and benchmarks are needed to address these two areas. Performance measures on the target machine can be used in determining how best to employ tasking; the results from similar benchmarks on the development machine would be largely meaningless. These performance measures can also ascertain memory constraints imposed by the hardware, and determine whether enough memory has been provided. In addition, performance measures can give an indication of where non-Ada code should be used. Benchmarks illustrating the best mechanisms for interfacing to non-Ada code are needed as well.

Fairly elaborate benchmarks may be needed to test graphics and the user interface. For example, as part of the efforts involved in prototyping various user interfaces, measurements of the graphics hardware performance (e.g., time needed to refresh a screen or to zoom) should be obtained.

4.5.3 Use of Benchmarks That Run on Both Machines

Benchmarks that run on both development and target machines are needed to identify which Ada features are supported differently on the two machines. Results from early application of such benchmarks can be be fed back into the Software Development Plans and methodologies. In some cases, this information would take the form of prohibitions (e.g., "Don't use feature X because it is not supported on all the machines."). In other cases, some restrictions might have to be invoked (e.g., "The largest-sized array that will fit on all machines is <n> words; thus no array should be larger than that, even though there is no practical limit to the size of arrays on the development machines."). Assuming the availability of adequate documentation, these benchmarks might be simple verifications of documented limits.

4.6 Summary

The use of Ada, or any other language, on a large systems development has many inherent risks. However, recent experience indicates that these risks are manageable if properly addressed. To this end, the above recommendations for risk reduction activities have been provided.

5.0 NEAR-TERM FAA ACTIVITIES

This section of the report summarizes multiple near-term FAA activities that must be undertaken to accomplish the following:

- Carry out an Ada software development on AAS.
- Realize the advantages of a software development approach such as the one characterized in Section 2.
- Address the risk items identified in Section 3.
- Carry out the risk reduction activities suggested in Section 4.

To put these recommendations into the proper temporal context, it should be noted that when the recommendations were being generated and this report written, the AAS had been under Design Competition Phase (DCP) for several years. There were two competing contractor teams, and the FAA was actively pursuing four goals:

- Developing a Statement of Work (SOW) for activities to be carried out during the remainder of DCP. Risk reduction activities for both contractors were being considered for inclusion.
- Developing a Request for Proposal (RFP) for the Acquisition Phase.
- Defining criteria for the proposal review and Acquisition Phase contract award.
- Defining work activities to be carried out during the Acquisition Phase. (These are specified in the Acquisition Phase SOW, which is part of the RFP; however, for purposes of this report, the acquisition phase RFP and SOW will be discussed separately.)

It is intended that the following recommended activities will influence these four goals. Unfortunately, it has proved impossible to present these recommended activities in the order in which they should be undertaken: all of them should be performed in the near future.

The study participants recognize that they are not familiar with the details of FAA's on-going or planned risk mitigation activities over the near-term. In this context, the activities described below may correspond to activities currently in progress.

5.1 Acquire Ada and Software Engineering Expertise

Because of the newness of Ada technology, there is a scarcity of experienced Ada personnel throughout government and industry. However,

the activities and recommendations discussed in this report depend upon both Ada and software engineering expertise. Therefore, it is necessary for the FAA to acquire such expertise in two ways: develop in-house expertise, and access outside expertise.

5.1.1 Develop In-House Expertise

For the FAA to develop in-house expertise, formal training will be required in Ada and in software engineering. Throughout the deliberations of this study, it was repeatedly emphasized that such training is necessary at all staff levels, including (or perhaps especially for) upper management. The U.S. Army Ada Training Curriculum (U.S. Army, 1984) is an example of an available training program that provides methodology and language courses at the appropriate levels. An excellent follow-on for gaining expertise would be for FAA personnel to dry run a software engineering exercise before giving it to the contractors (see Section 5.2 below). It may be desirable for FAA support contractors, along with internal staff, to be involved with the training and the dry run.

5.1.2 Access Services of Outside Expertise

In spite of Ada's risks, there have been several notable Ada successes in recent years. To capitalize on this real-world experience, outside Ada experts should be actively sought. Sources of such expertise include the Software Engineering Institute (in particular, their Software Environment Evaluation Project); NASA personnel (joint Ada activities with NASA personnel involved on the Space Station may be considered); and consultants (several companies specialize in Ada expertise). The FAA may also want to consider another study, similar to the one that resulted in this report, to take place after contract award. That follow-on study could formulate recommendations after reviewing the contractor's design. If this is to be done, the proper contract mechanisms must be provided in the RFP so that such a review, and any appropriate remedial response to it, will be within the scope of the contract.

5.2 Prepare and Dry Run a Software Engineering Exercise

As discussed in Section 4.3.3, there are many benefits to conducting a software engineering exercise. It is therefore recommended that one be used on AAS. However, preparing the exercise will require a considerable amount of planning, perhaps four to six staff for four to six months. Part of this preparation includes giving considerable thought to which risks should be addressed by the exercise. Furthermore, to provide maximum benefit, the exercise must be dry run. Doing so will yield insight into the problem statement, and allow it to be refined before being presented to the contractors. Other aspects of the preparation are the provision of an Ada environment for FAA use and the establishment of

evaluation criteria for assessing exercise results. Given these preparation needs, staff should be designated for the software engineering exercise, and planning begun on it, as soon as possible.

5.3 Prepare Ada-Oriented RFP and SOW with Revised Milestones

As noted in Section 2.1.4, an Ada-oriented development process is characterized by somewhat different milestones than those typically associated with large software procurements in the past. Thus, if Ada is to be used on AAS, the FAA should revise the AAS milestones to reflect an Ada development process. Specifically, these milestones should reflect the new design approach, requiring compilable top-level designs. Multiple reviews and walk-throughs, instead of a single, monolithic Critical Design Review (CDR), should also be indicated. As the milestones are revised, they must be coordinated with one another.

The coordination of milestones must include a decision on how the attainment of compilable designs is to be coordinated with contract award. Ideally, the contractors should generate designs, and the FAA should review them, before the contract is awarded. This would allow the designs to be used in the source selection process. However, if budget or other constraints preclude this approach, then compilable designs should be included as a milestone soon after contract award. This approach may involve revising the scheduled date for CDR (see Section 5.7).

5.4 Develop a Strategy

For the recommendations that the FAA chooses to undertake, a strategy is needed. For example, it must be determined which benchmarks will be selected by the contractors and which by the FAA; for those selected by the FAA, a determination is needed as to the exact benchmarks to be used. For all benchmarks, criteria for evaluating the results must be established.

The RFP should be revised to specify which risk areas the FAA wants addressed in the Risk Management Plans and which by other means. The contractors should also be asked to identify additional risk areas in their Software Development Plans and Risk Management Plans, and propose methods for addressing them.

Revision of the RFP should be coordinated with the software engineering exercise. The latter should address as many risks as is feasible; these risks must be determined and the remainder addressed in the Risk Management Plans.

5.5 Assess the Feasibility of a Fixed Price Contract

If a fixed price contract is being considered for AAS, then the feasibility of such a contract must be carefully assessed. In making this assessment, it is important to consider whether the estimated costs are realistic and known with a high degree of certainty. An important consideration is whether design and requirements are thorough and stable. If certain areas of the system are expected to evolve (e.g., user interfaces) because they cannot be defined precisely up front, then fixed price contracts for those areas are not feasible; a cost plus contract should be used in such cases to provide the needed flexibility.

Other items needing precise definition in a fixed price contract are the milestones (e.g., exactly what is contained in CDR) and their criteria for success. If success criteria are not well defined, or if certain areas are expected to evolve, then fixed price contracts are not feasible: either the items will be designed to cost (not necessarily undesirable), or costly contract modifications will be needed to allow for enough iterations to achieve customer satisfaction.

In making this feasibility assessment, one should not be tempted to infer that if costs are underestimated, then the Government enjoys a bargain price while the contractor loses--a "win-lose" situation. In fact, underestimates invariably result in a "lose-lose" situation. In the worst cases, with no incentive to perform, a contractor would remove its best people from the project and design to cost, not to requirements, while an antagonistic relationship might well develop between contractor and customer.

5.6 Develop a Fixed Price vs. Cost Plus Strategy

Once the assessment of fixed price feasibility has been completed, a strategy of fixed price vs. cost plus contracting should be formulated, providing for a mix of fixed price and cost plus components. For those items meeting the feasibility criteria described in Section 5.5, a fixed price costing arrangement can be developed; for those items involving more uncertainty, cost plus contracting can be used.

For fixed price items, decisions must be made on synchronizing the costing strategy with contract award. That is, a determination must be made as to the best time to go fixed price: on the one hand, even if a given item is best procured on a fixed price basis, existing uncertainties may mean that it is too early to do so; on the other hand, waiting until these uncertainties are resolved (perhaps after completion of the software engineering exercise or CDR) might require a delay in the Acquisition Phase contract award. Therefore, a strategy for synchronizing contract award with design stability must be formulated.

5.7 Required Activities That May Involve Schedule Revision

The FAA must carry out the following activities in order to ensure a successful Ada software development on AAS:

- Assemble and train staff (internal and support contractors).
- Prepare the RFP package and contract award criteria.
- Develop evaluation capabilities (tools and checklists).
- Prepare Ada progress indicators and milestones.
- Review the Ada-oriented compilable designs.
- Prepare and dry run the software engineering exercise.

Depending on when contract award takes place, one or both contractors must be tasked with carrying out the following activities to ensure a successful Ada software development on AAS:

- Prepare thorough Ada-oriented compilable designs.
- Prepare thorough Software Development and Risk Management Plans.
- Assess the maturity of tools and compilers.
- Perform the software engineering exercise.

In order to achieve the advantages described in Section 2.2, the FAA must allow enough time and funding in the schedule for the contractors to complete thorough requirements analysis and design. Not doing so would tend to negate the advantages offered by compilable designs. In fact, under such circumstances, those designs could prove a hindrance as the ripple effects introduced during the coding phase would result in many time-consuming recompilations. Because of the complexity and importance of the requirements analysis and design activities, they must be allotted adequate time. A determination must be made as to whether more time and money than originally anticipated are needed to implement this recommendation. Any such schedule revision must be coordinated with contractor and FAA risk reduction activities.

On the other hand, even if the schedule must be revised, this offers the potential for long-term savings, since less time and money should be needed during integration. Furthermore, product quality should be increased and risk reduced, partly because the ability to achieve stable top-level designs can be used as a criterion in the source selection process.

5.8 Bring the Maintenance Organization On Board Now

The long-term success of the AAS program depends upon early, active involvement of the eventual maintenance organization. Therefore, the software maintenance organization should be brought on board in the immediate future. This is important because several near-term activities should be undertaken by the maintenance organization.

First, a Software Maintenance Plan should be written now, even though it will have to be revised after an Acquisition Phase contractor has been selected so that it will work with the contractor's Software Development Plan. Thus the Software Maintenance Plan should be considered a living document.

Second, the maintenance organization should be involved in two near-term FAA activities: training (see Section 5.1) and the software engineering exercise (see Section 5.2).

Third, the maintenance organization should be involved in RFP and SOW preparation. This is especially important in the areas of documentation standards and deliverables, since maintenance personnel are the eventual users of these products. As such, their perspective on what should be produced differs somewhat from that of those involved in contract monitoring during the development phase. The maintenance organization may desire certain deliverables that would enhance maintenance productivity, such as high-level "pointers" telling where particular items are located in the software. Conversely, the maintenance organization can provide guidance on which anticipated deliverables are really unnecessary for their purposes. For example, detailed design documentation can be costly, tedious, and time-consuming to produce, yet much of it is often found to be of little use to the maintenance personnel for whom it was intended. This is because the documentation does not always track the actual system, especially when maintenance updates are made to the system, but the corresponding documentation is not revised accordingly. Thus the actual code proves to be a better source of information. This is especially true for an Ada software development because of the readability of Ada code.

Finally, an early determination should be negotiated with the maintenance organization as to its Acquisition Phase role. One possibility is for maintenance personnel to review AAS designs for items that may be overlooked by those responsible for monitoring the development process. Clarity of error diagnostics and ease of code change are examples of such items. Since the maintenance personnel will be using these items, they have a different perspective and are more likely to flag deficiencies. Furthermore, an Acquisition Phase role will familiarize maintenance personnel with the system, giving them the required background and capabilities when maintenance activities begin.

6.0 SUMMARY

The study participants unanimously concluded that Ada is the appropriate language choice for AAS, given the following qualifications: the software development approach is modified, appropriate use of non-Ada code is allowed, contractor readiness is evaluated, and risk reduction activities are undertaken. Recommendations related to each of these qualifications have been provided, as have near-term FAA activities for implementing these recommendations. Tables 6-1 to 6-4 map these activities into the four task areas--DCP risk reduction, Acquisition Phase RFP, contract award criteria, and Acquisition Phase SOW--that are currently being pursued. Of course, there are many additional activities associated with these task areas that are beyond the scope of this study.

TABLE 6-1
RECOMMENDATION SUMMARY:
DCP RISK REDUCTION TASKS

- Require compilable top-level design in Ada.*
- Develop software development and Risk Management Plans.**
- Perform benchmarks stressing large system aspects of compiler and support environment.

*Defer to Acquisition Phase SOW only if budget considerations do not allow for this as part of DCP risk reduction tasks.

**May be part of response to RFP.

TABLE 6-2
RECOMMENDATION SUMMARY:
ACQUISITION PHASE RFP

- Require submission of Risk Management Plans addressing each risk area identified in Section 3.

- Establish that contractors provide basis of cost estimation, while FAA provides cost estimation guidelines.

- Identify benchmarks and prototypes.

- Require contractors to take Software Engineering Exercise.*

  Use Ada-oriented designs, schedules, milestones, and deliverables.

- Require evidence of commitment to using tools and retaining key personnel.

*May be done as part of DCP risk reduction tasks.

TABLE 6-3
RECOMMENDATION SUMMARY:
CRITERIA FOR CONTRACT AWARD

- Software Development Plans
- Risk Management Plans: assessment of approaches for each risk item as to realism, scaling issues, and progress
- Realistic understanding of Ada impact on the following:
  - Compiler and environment maturity for specific hardware
  - Costs and schedules
  - Incremental development approach
  - Recompilation requirements
  - Need for modifying environments
  - Need for use of non-Ada code and approach for implementing and containing it
- Results of software engineering exercise
- Design team profile (small, experienced team)
- Early availability of mature support environments

TABLE 6-3
RECOMMENDATION SUMMARY:
CRITERIA FOR CONTRACT AWARD
(Concluded)

- Evidence of readiness
  - Benchmarks
  - Published readiness criteria
- Design evaluation of top-level Ada design
  - Check for completeness, feasibility, consistency, traceability, reusability, and growth
  - Compliance of design with requirements
- Independent cost estimates

TABLE 6-4
RECOMMENDATION SUMMARY:
ACTIVITIES IN ACQUISITION PHASE SOW

- Carry out Software Development Plan methodology (emphasis on prototyping).
- Adhere to Risk Management Plan.
- Use incremental development (use of prototypes).
- Implement incremental reviews (including reuse and growth reviews).
- Apply revised milestones and activities.
- Produce machine-readable, Ada-oriented deliverables.
- Require use of benchmarks and prototypes.

APPENDIX A

TUTORIAL ON THE ADVANCED AUTOMATION SYSTEM (AAS)

A.1 Today's System

Before considering the role of AAS in the future Air Traffic Control (ATC) system, three types of FAA facilities used in today's system for a typical commercial airline flight are described.

A.1.1 Typical Flight Scenario

The first facility usually encountered on a flight is the Air Traffic Control Tower, typically located above a busy airport and staffed by the air traffic controllers looking through the large glass windows. Approximately 400 towers direct the movement of traffic at and in the vicinity of airports. Controllers in the tower are in radio contact with the pilot before takeoff, providing verbal guidance on maneuvering the aircraft from the gate to the runway. The tower controllers are also responsible for directing actual takeoffs and landings. It is they who issue the "Cleared for Takeoff" command.

Upon takeoff, the pilot switches radio frequencies to contact a controller in the Terminal Radar Approach Control (TRACON) Facility, often located one floor below the tower. There are almost 200 of these facilities. Several controllers staffing the displays in this darkened "radar room" are responsible for maintaining an orderly flow of traffic to and from the airport within the "terminal area" radius of 30 to 40 miles. In some locations, a single TRACON is responsible for several adjacent airports.

As the flight approaches the boundary of the terminal area, control of the flight is handed off to one of the 20 Air Route Traffic Control Centers (ARTCCs), that are located in the continental United States. These large en route facilities have regional responsibility for more than 100,000 square miles of airspace, with sometimes 50 or more controllers on duty at a given time. It is their responsibility to monitor an aircraft's route of flight between terminal areas. They provide separation services, traffic advisories, and weather advisories. While en route, a typical flight may be handed off several times between controllers handling different portions of the airspace within an ARTCC's area of responsibility, or between controllers in adjacent ARTCCs. In each case, the procedure is virtually identical to that employed in the original transfer of control from the TRACON to the initial ARTCC.

As the aircraft descends, the process is reversed, with control being handed off first to the terminal responsible for the destination airport, and finally to the tower, which clears the aircraft for landing.

A.1.2 Limitations of Current System

ARTCCs presently use IBM 9020 computers (essentially IBM 360s) developed in the 1960s to process radar and flight data. Aircraft separation is based on radar data display and flight data printed on paper flight progress strips. Despite their capabilities, the current 9020 computer systems are not expected to be adequate for handling the projected growth in aviation traffic beyond the late 1980s.

Minimal automation capabilities exist in today's tower. Although paper flight progress strips and a radar display may be available, these are driven by computers in the associated ARTCC or TRACON. The TRACON also has paper flight progress strips and radar displays. However, the automation equipment at the TRACONs is significantly different from that found at the ARTCCs.

The present system is labor-intensive. A great deal of manual effort is required of air traffic controllers. Even though higher levels of automation are being developed to reduce operational costs, improve safety, and provide fuel savings for aircraft users, implementation of these enhancements is not possible because of capacity limits. There are high hardware and software maintenance costs associated with the operation of the current system. Moreover, equipment manufacturers cannot provide parts indefinitely, even regardless of cost.

A.2 Evolution of System

The FAA's current en route modernization programs are aimed at replacing existing ATC computer systems with modern technology. New software will be implemented to enhance safety and increase productivity, and permit the integration of a number of functions now performed separately.

The new computer system is being designed for approximately 100 percent functional availability and reliability of services. Moreover, the new computers, software programs, and displays now being developed will be capable of providing both en route and terminal services. During the 1990s, this will enable the FAA to consolidate TRACONs and ARTCCs into new Area Control Facilities (ACFs), which will be located at the existing ARTCC locations. Some ACFs will be similar to the present ARTCCs in that they will be responsible for the control of a large volume of en route airspace. However, they will also include the terminal control responsibilities of some present TRACONs that are contained within the airspace that will become the ACF airspace. Other

ACFs will be formed by combining several adjacent TRACONs and the appropriate adjoining low-altitude en route sectors into a single facility. These ACFs will be concerned primarily with terminal ATC and will be responsible for the control of a smaller volume of airspace. Thus, the airspace of each of these ACFs will consist of a combination of the individual volumes of airspace presently controlled by a number of TRACONs, plus some present en route low-altitude airspace.

The resulting reduction in the number of facilities will eliminate, or considerably diminish, the present demarcation of services, thereby reducing operational overhead. Airport traffic control towers will still provide airport services; however, this will be done with increased automation.

A.2.1 Host Program

The first step towards modernization is replacement of the 9020 computers in the ARTCCs with host computers. (A host computer is a replacement computer that uses existing software from another computer system.) These host computers will be capable of running the present 9020 software package with minimal modifications. An acquisition contract was awarded in July 1985 to IBM, which will install 9020 instruction-compatible IBM 3083 computers from late 1986 to 1988. The purpose of this procurement is to provide needed computer capacity for the present en route system as early as practicable, and improve computer reliability and availability. This will provide for projected air traffic growth until the AAS is in place.

A.2.2 AAS Program

The purpose of AAS is to provide a total automation system that includes the controller Sector Suite, new computer software, and new processors to augment the host computers. AAS will provide the capacity to handle the projected traffic load through the 1990s and beyond, the capability to perform all of the new functions to be introduced into the system through the mid-1990s, increased productivity through the introduction of new Sector Suites at the earliest practical time, a high degree of reliability and availability, and the capability for enhancement to perform other functions subsequently introduced into the system.

The transition to AAS will consist of four steps. First will be the replacement of the 9020 computers at the ARTCCs with the new host computers described above. Second is the implementation of the Initial Sector Suite System (ISSS). Third is the provision of terminal consolidation capabilities. The final step provides en route automation capabilities.

The Sector Suite will consist of common controller workstations called common consoles, used for both en route and terminal functions. It will incorporate an improved man-machine interface, including the use of color displays and electronic presentation of flight data, to enhance controller productivity. A typical Sector Suite will consist of displays that present a plan view of the current situation, such as (1) radar returns of aircraft position, and real-time weather information; (2) electronic display of flight data (eliminating the need for manual flight strip handling); and (3) the display of planning information and advanced functions. The Sector Suite will be installed initially in en route facilities, and interfaced with the host computers and rehosted software to form ISSS.

During the final transitional step, new computer software and new processors to augment the host computers will be introduced. Software functions that are now unique to the terminal ATC systems will be incorporated into the new software to support the area control facilities. All remaining elements of the current en route automation hardware and software will be replaced.

AAS will be a distributed system: operations requiring centralized processing will be accomplished in the centralized computers, with all remaining functions performed within the individual Sector Suites. The reduced capability and emergency modes of AAS will ensure that surveillance, flight, and weather data are provided with near 100 percent functional reliability. Additional safety and productivity functions will be included in the new software.

In August 1984, competitive contracts were awarded to two teams led by IBM and Hughes for the design of AAS. These contracts provide for the total system design of ISSS and AAS, including all hardware and software, and development and demonstration of the common consoles. Contract award for the production phase of AAS is currently scheduled for 1988. Transition to the full AAS will begin with ISSS, currently scheduled for installation in the early 1990s. The final step will prepare the way for additional integration of terminal and en route operations within area control facilities and for higher levels of automation when the full AAS is implemented in the mid-1900s. Not only will AAS provide uniform processing capabilities for en route and terminal ATC, but it will also increase the tower automation capabilities; tower console installations are currently scheduled for 1994 to 1999.

A.3 AAS System Description

A.3.1 Overview

An Area Control Computer Complex (ACCC) is the equipment and software that provides automation support for the control of aircraft in

a volume of airspace under the air traffic jurisdiction of an ACF. The equipment and software of all ACCCs will be identical, varying only in installation quantities and in software adaptation to the operational configuration of the sites. An ACCC includes computers, computer software and related documentation, displays, storage devices, input devices, output devices, controller workstations, operator workstations, interconnecting communications, a supporting maintenance subsystem, a training subsystem, and interfaces with other FAA systems.

A Tower Control Computer Complex (TCCC) is the equipment and software that provides automation support for the control of aircraft in a volume of airspace including the airport surface under the air traffic jurisdiction of an Air Traffic Control Tower (ATCT). This includes the control of those airport systems that are related to ATC. TCCCs will be located at several hundred ATCTs. To the extent practical, ACCCs and TCCCs will have common equipment and software.

The major functions of AAS are as follows:

- Primary Processing, including radar target processing, tracking, separation assurance, flight data processing, automation processing, and weather processing.
- Data Entry and Display, including display presentation and controller input message make-up to support seven different types of operational positions.
- A Monitor and Control Capability to provide the maintenance supervisor workstation and associated processing.
- Emergency Processing, which provides a subset of ACCC functions sufficient to ensure the safety of aircraft during ACCC equipment failures.
- Stand-Alone Processing to enable TCCC processing to continue in the absence of an ACCC interface.

It is anticipated that over one million source lines of high order language (HOL) code will be developed for AAS.

A.3.2 Workload Characteristics

An ACCC is to be capable of supporting up to 280 common consoles configured as up to 140 Sector Suites; a Sector Suite can consist of from 1 to 4 common consoles. An ACCC is to be capable of processing data from up to 27 long-range and up to 53 short-range surveillance sites, with a total surveillance coverage area of up to 2500 x 2500 nautical miles, resulting in up to approximately 4000 aircraft target reports per second.

The system is to be capable of maintaining up to 5500 tracks and over 6000 flight plans.

A.3.3 Response Time Characteristics

AAS response time requirements vary widely depending on the nature of the function being performed. For example, lexical feedback for controller keyboard entries and display item selections is to be provided within 50 milliseconds (mean); response time from time of receipt of surveillance inputs is not to exceed one second for target position updating or 1.5 seconds for track data updating; and changes to flight data requiring reprocessing of the route of flight are to be provided in 1.5 seconds (mean).

A.3.4 Availability Requirements

AAS availability is defined in terms of the ability of the AAS system to provide required services within the required response times. The availability of an ACCC providing all ATC operational functions (full-service mode) is to be at least 0.999995 (down 2.6 min/yr). The availability of an ACCC providing emergency mode functions is to be at least 0.9999999 (down 3 sec/yr). Emergency mode functions include tracking and display data maintenance, but do not include flight plan processing or separation assurance.

A.3.5 Design and Construction Requirements

Software developed for AAS is to be written in a single high order language. Any exceptions must be justified by the contractor and approved by the FAA. Identical computer programs, adapted through site-specific data to local resources, environment, and workload, will be installed at each site.

All common consoles within an ACCC Sector Suite will be identical and must be capable of performing the functions of any other console within the suite.

An ACCC local communications network using standardized local area network protocols is to be provided for communications within the ACCC, and between the ACCC and certain external systems.

REFERENCES

Boehm, Barry W (1981), Software Engineering Economics, Englewood Cliffs, NJ: Prentice-Hall.

Booch, Grady (1987), Software Engineering with Ada, Menlo Park, CA: The Benjamin/Cummings Publishing Company.

Bryan, Doug (January, February 1987), "Dear Ada", Ada Letters, VII:1, pp. 25-28.

Buhr, R.J. (1984), System Design with Ada, Englewood Cliffs, NJ: Prentice-Hall, Inc.

Castor, Virginia L. (1985), Issues To Be Considered in The Evaluation of Technical Proposals from The Ada Language Perspective, AFWAL-TR-85-1100, Wright-Patterson Air Force Base, OH.

Humphrey, W.S. et al. (1987), Assessing the Software Engineering Capability of Contractors, Pittsburgh, PA: Software Engineering Institute.

The Institute of Electrical and Electronics Engineers, Inc, IEEE Recommended Practice for Ada As a Program Design Language, IEEE Std 990-1987, New York, NY.

U.S. Air Force (1984), Software Development Capability/Capacity Review, Wright-Patterson Air Force Base, OH.

U.S. Army (1984), Ada Training Curriculum, Fort Monmouth, NJ: Center For Tactical Computer Systems.

Weiderman, Nelson et al. (1987), Evaluation of Ada Environments, Pittsburgh, PA: Software Engineering Institute.

U.S. Department of Defense (17 February 1983), Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A-1983.

GLOSSARY OF ACRONYMS

AAS	Advanced Automation System
ACCC	Area Control Computer Complex
ACF	Area Control Facility
ARTCC	Air Route Traffic Control Center
ATC	Air Traffic Control
CDR	Critical Design Review
COCOMO	Constructive Cost Model
CPU	Central Processing Unit
DCP	Design Competition Phase
FAA	Federal Aviation Administration
HOL	High Order Language
ISSS	Initial Sector Suite System
NAS	National Airspace System
NASA	National Aeronautics and Space Administration
PDL	Program Design Language
PDR	Preliminary Design Review
RFP	Request for Proposals
SOW	Statement of Work
TCCC	Tower Control Computer Complex
TRACON	Terminal Radar Approach Control

END

9-87

DTIC