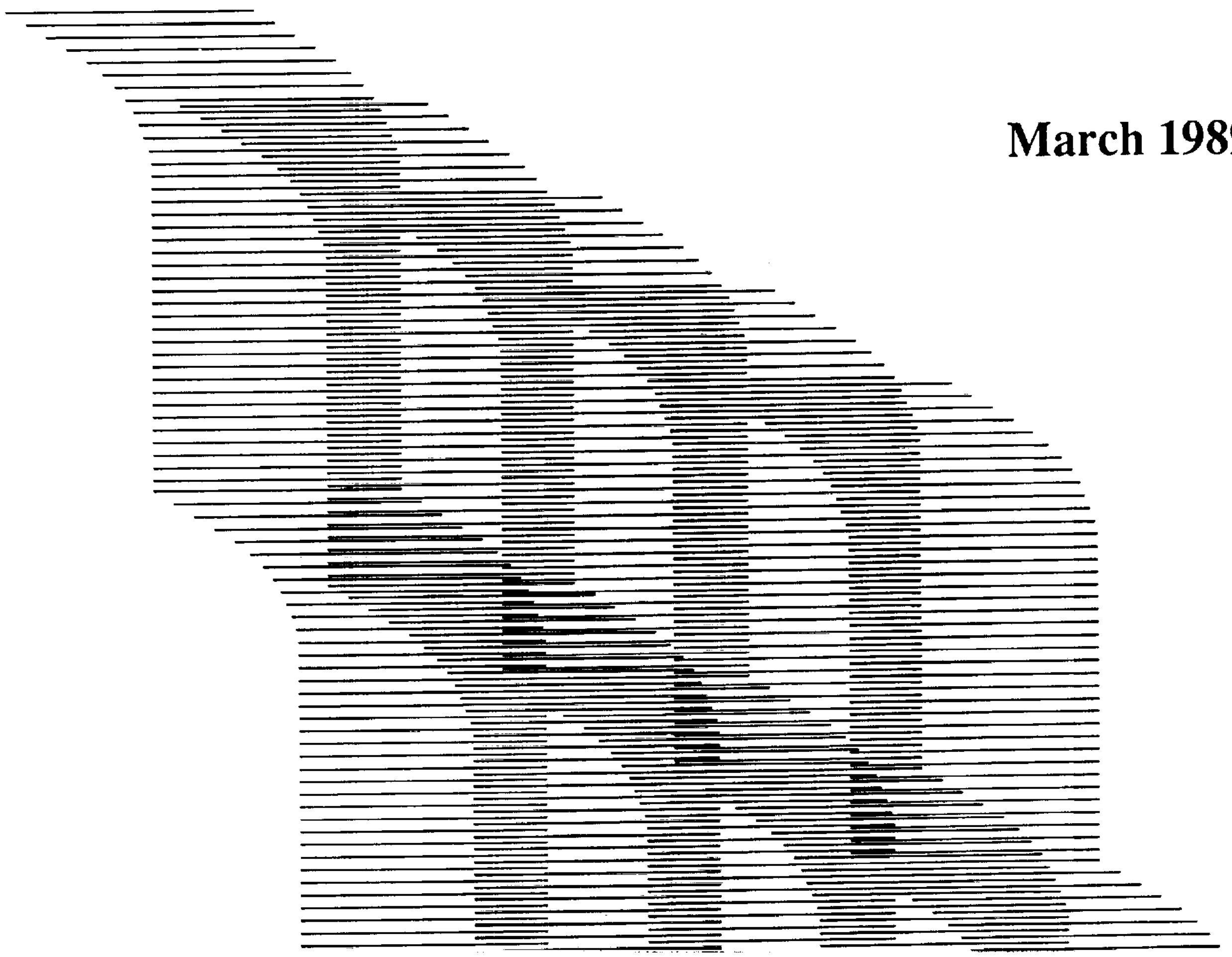
Ada and Software Management in NASA:

Assessment and Recommendations

March 1989

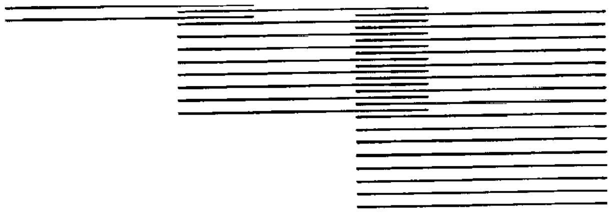


(NASA-TM-103298) Ada AND SOFTWARE MANAGEMENT IN NASA: ASSESSMENT AND RECOMMENDATIONS (NASA) 70 p CSCL 09B

N92-14604

Unclas

G3/61 0276903



National Aeronautics and
Space Administration

Goddard Space Flight Center
Greenbelt, Maryland 20771

ADA AND SOFTWARE MANAGEMENT IN NASA:

ASSESSMENT AND RECOMMENDATIONS

A Report to the Information Resources Management Council

by the

Ada and Software Management Assessment Working Group

March 1989

National Aeronautics and Space Administration

Goddard Space Flight Center

Greenbelt, MD 20771

WORKING GROUP MEMBERS

Frank E. McGarry, Chair
Goddard Space Flight Center

Robert A. Carlson
Ames Research Center

Edward S. Chevers
Johnson Space Center

John L. Feagan
Lewis Research Center

Donald W. Sova
NASA Headquarters, Code QR

John W. Wolfsberger
Marshall Space Flight Center

Arthur I. Zygielbaum
Jet Propulsion Laboratory

Additionally, the following advisors made significant contributions:

Michael R. Gardner[1]
Computer Sciences Corporation

Jody M. Steinbacher
Jet Propulsion Laboratory

Keiji Tasaki
Goddard Space Flight Center

Susan Voigt
Langley Research Center

[1]Michael R. Gardner was also responsible for the full editing and integration of all portions of the report generated by other working group members.

FOREWORD

The National Aeronautics and Space Administration (NASA) Information Resources Management (IRM) Council was established (1) to coordinate agency-wide programmatic, operational, and institutional requirements through the development of IRM policies and initiatives and (2) to respond on an agency basis to relevant policies, regulations, and statutes issued by the executive and legislative branches of the federal government.

At its March 1988 meeting, members of the IRM Council expressed concern that NASA may not have the infrastructure (standards, policies, and internal organizations) necessary to support the use of Ada for major NASA software projects. Members also observed that the agency has no coordinated strategy for applying its experiences with Ada to subsequent projects.

The Council, therefore, recommended that its chairman appoint a group "to assess the agency's ongoing and planned Ada activities and the infrastructure supporting software management and the Ada activities (present and projected)." Accordingly, Chairman Noel Hinners established an intercenter Ada and Software Management Assessment Working Group (ASMAWG) and directed it to "review the agency's software management programs and present an Ada implementation and use strategy appropriate for NASA over the next 5 years" (Hinners, 27 June 1988).

The ASMAWG consists of seven members and four advisors, representing a broad spectrum of agency organizations and activities. To assess NASA practices and plans, the working group reviewed many programs; studied previous reports on these topics; interviewed many software engineers employed by NASA, support contractors, and other organizations; and analyzed the current NASA infrastructure supporting the software development process. The products of the ASMAWG activities consist of this report and a 5-year plan for NASA (McGarry et al., April 1989). The plan describes steps toward the adoption of effective software engineering practices and technologies.

F. E. McGarry

Frank E. McGarry
Chair, ASMAWG

EXECUTIVE SUMMARY

INTRODUCTION

Recent NASA missions have required software systems that are larger, more complex, and more critical than NASA software systems of the past. The Ada programming language and the software methods and support environments associated with it are seen as potential breakthroughs in meeting NASA's software requirements.

OBJECTIVES AND SCOPE

This report presents the findings of a study by the Ada and Software Management Assessment Working Group (ASMAWG). The study was chartered to perform three tasks:

- "Assess the agency's ongoing and planned Ada activities"
- Assess "the infrastructure [standards, policies, and internal organizations] supporting software management and the Ada activities"
- "Present an Ada implementation and use strategy appropriate for NASA over the next 5 years" (Hinners, 27 June 1988)

FINDINGS

The following statements summarize this report's findings concerning NASA's current use of Ada and software engineering technology:

- Ada is an appropriate vehicle to support the evolution to improved software practices in NASA.
- Although NASA offers courses in Ada and software engineering, the training programs are not adequate to accomplish a transition to Ada.
- NASA has very little experience with Ada, and current plans do not provide for building an experience base adequate to meet existing commitments for the use of Ada.
- NASA Ada projects have often selected or developed sophisticated tools and methods; but software reuse is still in a rudimentary state.

- NASA software management has been weak in two areas pertinent to this report: agency-level planning for the use of state-of-the-art software engineering practices (e.g., Ada) and project-level management of software risks, including those associated with Ada.
- NASA does not have an adequate set of agency-level standards and internal organizations to provide direction in software engineering and to support the evolution to new software technologies such as Ada.
- Although NASA has sponsored extensive research in software engineering, some of which has dealt with Ada-related technology, the agency has not had a broad, coordinated program of research, experimentation, and pilot projects.
- NASA does not have an adequate program of collection and use of metric data about the software development process.

RECOMMENDATIONS

Based on the findings of this report, the following statements summarize the recommendations for a transition to improved software engineering in NASA:

- NASA should adopt Ada as its standard programming language.
- NASA should establish a Software Engineering and Ada Implementation Task Force.
- NASA should develop and adopt tailorable standards for software development, management, and assurance.
- NASA should evolve toward a common software support environment.
- Each center should develop a plan for evolving to Ada.
- The Software Engineering and Ada Implementation Task Force should ensure the development and implementation of an agency-wide core curriculum in software engineering and Ada. Each center should adapt the core curriculum to its specific needs.
- For any "critical" project, management should be required to develop and implement a written risk management plan.
- NASA should establish an Ada incentive program for its software contractors.

- The Office of Aeronautics and Space Technology (Code R) should plan and coordinate agency-wide software research and development, more of which should support Ada.
- NASA should establish an agency-wide program to collect and use software metrics.
- NASA should establish a Software Process Engineering Task Force to support the evaluation and improvement of the agency's software acquisition and in-house development processes.

Table of Contents

Table of Contents (Cont'd)

Table of Contents (Cont'd)

List of Illustrations

List of Tables

1 — Introduction

Software has become a pervasive part of the modern world. Most organizations, whether corporate or governmental, are highly dependent upon information systems that are orchestrated through complex software systems.

In the National Aeronautics and Space Administration (NASA), software has been a key element in nearly every project. However, in recent years, both the criticality and complexity of NASA software have been increasing at rates that demand new, more effective software development approaches; otherwise, the software element of a project may cause its failure.

NASA's success in accomplishing the first steps in the exploration of space is testimony to the agency's ability to produce large quantities of complex, reliable, real-time software. But new initiatives involved in the Space Station Freedom Program (SSFP), the exploration of Mars and the Moon, and the exploration of the Earth from space require software whose size and complexity are orders of magnitude greater than existing NASA software. The agency's continued success hinges to a large extent on meeting the challenge of developing that software.

One major software technology that has evolved over the past few years is the Ada programming language. This language and the software methods and support environments associated with it are seen as potential breakthroughs in NASA's approach to software.

1.1 Objectives and Scope

This report presents the findings of a study that the Ada and Software Management Assessment Working Group (ASMAWG) undertook at the request of the NASA Information Resources Management (IRM) Council. The study was chartered to perform three tasks:

- "Assess the agency's ongoing and planned Ada activities"
- Assess "the infrastructure [standards, policies, and internal organizations] supporting software management and the Ada activities"
- "Present an Ada implementation and use strategy appropriate for NASA over the next 5 years" (Hinners, 27 June 1988)

The report summarizes the results of the working group's effort and presents recommendations about the incorporation of effective software engineering technologies into NASA's software development processes. In particular, the report describes the implications of Ada and then presents recommendations about the extent to which the language is appropriate for NASA and about how the agency should use it.

The working group also produced a 5-year plan for NASA (McGarry et al., April 1989), which defines specific steps for implementing the recommendations presented in Section 7 of this report.

1.2 Sources of Information

This report is based on information from four sources:

1. *Discussions and interviews.* The ASMAWG held personal discussions with software engineers and Ada experts employed by many NASA centers, support contractors, and other organizations. The persons consulted are listed in the Appendix.
2. *Review of completed studies and reports.* NASA and other agencies have carried out Ada studies over the past 3 years. The ASMAWG reviewed key reports to determine their relevance to the present study. These studies and reports are included in the list of references.
3. *Analysis of results of Ada development efforts.* The ASMAWG identified and analyzed the results of ongoing and completed projects that used Ada as the primary programming language. The group collected and analyzed information on cost, reliability, quality, other software characteristics, and lessons learned.
4. *Review of the current NASA infrastructure.* The ASMAWG studied current NASA policies, standards, and internal organizations supporting software development.

1.3 Organization of This Report

The remainder of this report is organized as follows:

- **Section 2** describes the growing size, complexity, and criticality of software systems that NASA develops or acquires.
- **Section 3** discusses the Ada programming language and its potential benefits to an organization that requires large, reliable software systems.

- **Section 4** presents a model of how an organization that acquires or develops software can evolve to the use of Ada and state-of-the-art software engineering.
- **Section 5** describes NASA's current use of Ada and current practice in software engineering.
- **Section 6** presents findings on the extent to which NASA conforms to and falls short of Section 4's model for transition to Ada and state-of-the-art software engineering.
- **Section 7** contains recommendations for NASA in light of the findings.

2 — Software Development in NASA Today

NASA computer software, which has always been a vital element of the agency's programs, is now growing in criticality, size, and complexity at an accelerated rate.

2.1 Size and Cost of NASA Software

The development, purchase, and maintenance of software for NASA require more than 20 percent of the total NASA budget.

Our estimate is that NASA and its contractors probably develop between 15 and 20 million new source lines of code (SLOC) per year, at an annual cost of about $1 billion. Across the agency, the annual cost of building new software exceeds 10 percent of the total NASA budget; and the cost of routine maintenance, together with the purchase of commercial off-the-shelf software and the building of special-purpose software (such as personal computer software, special scientific computing software, and institutional support software), probably exceeds another 10 percent of the total NASA budget.[1]

Software is a critical component and a significant cost element of nearly all NASA projects. The software required to support the first shuttle launch exceeded 20 million SLOC and cost more than $1.2 billion (see Table 2-1). The additional recurring cost for the shuttle flight software alone exceeds $30 million per year. The recurring cost is driven by the many changes required for each new shuttle launch and by the frequent recertification required for the software.

NASA requires both flight and ground software to carry out its missions. The flight software is the most expensive (per SLOC) to build and to maintain because of the extremely high reliability required. For typical mission support in NASA, approximately 2 to 3 million new SLOC are required. Some large projects, such as the shuttle, require up to 25 million SLOC. Typically, the flight code is between 5 and 10 thousand SLOC, and

[1] These estimates were derived from a detailed analysis of several major components at Goddard Space Flight Center (GSFC) and Lewis Research Center (LeRC) and by review of the NASA *Information Technology Systems Plan*.

the rest is nonembedded ground software[1] (see Table 2-2). For the shuttle, the flight software was approximately 580 thousand SLOC, and the ground software exceeded 23 million SLOC (see Table 2-1).

Table 2-1. Shuttle Software

Function	Flight/Ground F/G	Size (KSLOC)*	Languages
Flight Planning			
- Design	G	4,700	FORTRAN
- Crew Activity	G	57	FORTRAN
- Remote Manipulation System	G	22	FORTRAN
Flight Readiness			
- S/W** Development Lab	G	1,300	PL/1, HAL, ALC***
- S/W Production Facility	G	2,300	PL/1, C
- Mission Simulation	G	1,800	FORTRAN, ALC
Flight Control			
- Launch Processing	G	13,000	GOAL
- Mission Control	G	1,800	FORTRAN, PL/1
Flight - Onboard	F	580	HAL/S, ALC

*	KSLOC	= thousands of source lines of code
**	S/W	= software
***	ALC	= assembly language code

In developing this very large amount of software, NASA usually contracts for over 90 percent of the development and does the remainder in-house. Therefore, the principal role of NASA in the production of new software is that of acquisition manager. Contractors are responsible for most of the implementation and testing. Each of the NASA centers has some level of software responsibility, whether it is mission support and

[1] These estimates are based on a review of NASA projects including Gamma Ray Observatory (GRO), Galileo, Cosmic Background Explorer (COBE), Solar Maximum Mission (SMM), and several others.

operations, testing or research support; therefore, all centers must have skilled software engineers and managers.

Table 2-2. Mission Support Software

Project	New Software (KSLOC)	Percent Flight	Primary Languages
Gamma Ray Observatory (GRO)	850	5	FORTRAN, ALC
Galileo	600	1	FORTRAN, HAL
Spacelab	400	18	FORTRAN, ALC
Hubble Space Telescope (HST)	2,600	3	FORTRAN, ALC
Cosmic Background Explorer (COBE)	925	1	FORTRAN, ALC
Solar Maximum Mission (SMM)	114	4	FORTRAN, ALC

2.2 The Changing Role of Software

Computers and their software are essential to increasingly sophisticated technical activities in NASA.

NASA's first use of software was to solve small, specific problems. The programming was usually done by engineers or scientists working alone or in small teams. The programs tended to be simple, since the problems the developers were addressing were themselves simple. Such software development involved very little use of engineering disciplines. Indeed, in the early days of NASA, software technology was so young that very few engineering or management principles applicable to software were even known.

Today, the role of software is much greater than in NASA's early days. Functions once performed manually or in hardware are now done under software control. Coping with the sheer complexity of some operations and processes, such as launch processing for the space shuttle, has required the use of computers and, therefore, software. In addition, more functions are being put into software because of its flexibility, an attribute that is especially important in the case of programs like the SSFP that are expected to span decades.

Many current NASA missions are large, complex, multicenter development efforts. Moreover, they are intended to interact with other existing and planned systems. The Space Station Freedom, for example, will depend on millions of lines of code developed throughout the agency. The software for such systems is critical and pervasive. It is expected not only to perform the mission for which it was developed but also to be adaptable and flexible over a long life cycle (up to 30 years for Space Station Freedom).

The use of computer simulation and modeling has accelerated the process of creating, evaluating, and testing innovative designs for aerospace vehicles and structures like the aerospace plane and the Space Station Freedom. Supercomputer simulations are now used to supplement or replace wind-tunnel testing. Such computer-aided engineering requires sophisticated software.

Analysis of scientific and engineering findings is becoming more complex and hence more dependent on capable software and on the cost-effective development or acquisition of that software. NASA will receive billions of bits of information per day from the Hubble Space Telescope, for example, and will receive trillions of bits per day from the Space Station Freedom. Sophisticated software is needed to manage and analyze this avalanche of data (Kneale, 12 January 1988).

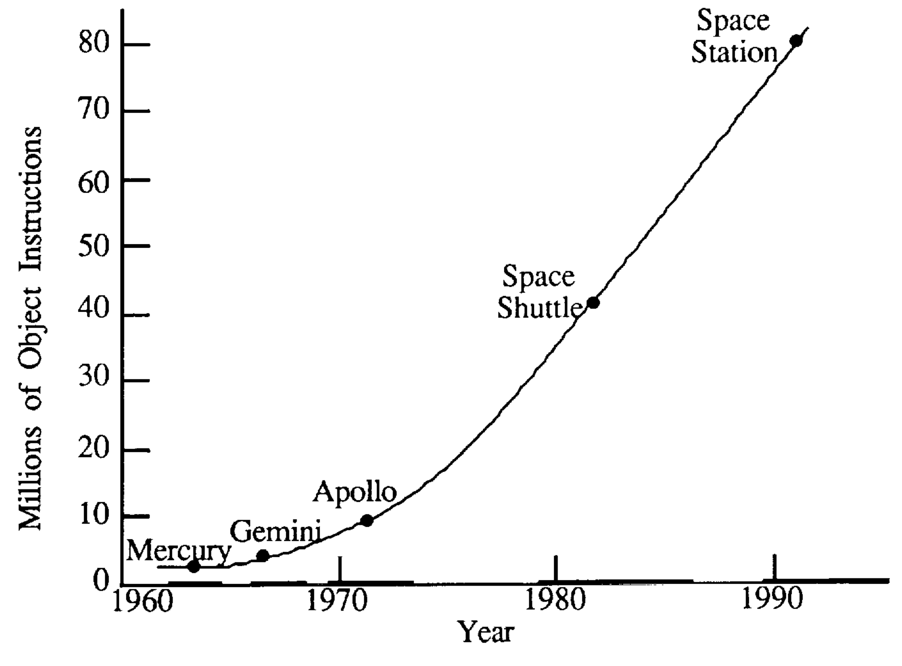
In the future, software will continue to grow in complexity, size, functionality, life expectancy, and required reliability. Software for such missions as the Space Station Freedom and the Earth Observational System will need to have high quality and reliability and yet be adaptable and flexible over its long life cycle.

To address this growing complexity, approaches to the software process must evolve to incorporate state-of-the-art principles of software engineering. These principles provide new design techniques; strategies for decomposing extremely complex problems; concepts for software reuse; techniques for attaining extremely high reliability; methods of testing and verification; and concepts for managing, planning, and controlling high-quality software. The success of such missions as the space shuttle is due in part to the use of new software engineering concepts and techniques (Kolkhorst, September 1988).

2.3 The Growth of NASA Software

Software's size, influence, and criticality are growing dramatically in NASA.

Since there is no universally accepted measure for software size, it is difficult to compare data drawn from different projects. Nonetheless, it is clear that the expanding role of NASA software has resulted in a long-term upward trend in the size of software systems. Figure 2-1 shows that flight software for major NASA programs has quadrupled in size over a 20-year period. (Boehm's estimate of Space Station Freedom software size may not be correct.) There is every indication that this growth in demand will continue and, in fact, that the rate of growth will continue to increase.



Source: Boehm (September 1987).

Figure 2-1. Estimated Growth in Software Demand: Manned Spaceflight Program

If the demand for software continues to increase at the historic rate and unless more efficient approaches to software development are found, NASA will be unable to supply the number of software engineers required to meet this increased demand. The continued demand for increased software functionality, coupled with a limited supply of talented personnel, is an issue of growing concern for all parts of the agency.

One technology that was developed to support new software engineering principles, and thus to reduce the gap between software supply and demand, is the Ada programming language and its associated principles, methods, and tools. The next section discusses the potential value of this technology in meeting the challenges posed by NASA's software needs.

3 — The Promises of Ada

Ada has promise for more efficient production of higher quality software through its support for the principles and goals of software engineering.

3.1 Software Engineering and the Software Crisis

The discipline of software engineering has the potential to alleviate the pervasive problems that constitute the "software crisis."

In 1968, a North Atlantic Treaty Organization (NATO) conference was held to discuss a perceived condition known as the *software crisis*. This crisis, still unresolved today, consists of deficiencies in the following properties of much of the software that is produced: correspondence with user needs, reliability, cost, modifiability, timeliness, portability, and efficiency (Booch, 1987b, pp. 7-10). The term *software engineering* was coined at that NATO conference. Software engineering is the disciplined use of software development principles, methods, and tools in the production of software systems. The discipline of software engineering addresses all facets of software specification, development, and maintenance. Its main goal is the cost-effective development of complex software systems that are reliable, efficient, and maintainable. It concerns itself with techniques for handling complex software problems; planning and managing software projects; development methods; estimation of costs and schedules; software architecture; reliability; assurance; and reuse of software components. The field of software engineering has the potential to remove or at least alleviate the software deficiencies cited above.

3.2 The Inception of Ada

The Department of Defense established a program to create and use a new high-order language to improve software productivity and quality.

In 1974, a report by the Department of Defense (DoD) estimated that annual DoD software costs would soon exceed $3 billion. More recent estimates indicate that total DoD software costs will reach $30 billion by the early 1990s. During this period, the DoD's demand for software is expected to grow about 12 percent per year, the number of software personnel to grow about 4 percent per year, and their productivity to grow about 4 percent per year. Projecting these three growth rates into the future yields an expected shortage of software (Booch, 1987b, p. 10). Other studies noted that hundreds of programming languages or dialects were being used by the DoD and its contractors. This situation made it difficult to interchange programs or programmers within the DoD and increased the department's software maintenance costs. To address these problems, the DoD set in motion a process to design a new high-order language, which was eventually named *Ada* (Sammet, August 1986, pp. 723-725).

The Ada language is the central element of a broader entity sometimes called the "Ada culture" (Booch, 1987b, p. 4). This culture includes the following elements:

- The Ada language itself
- Principles of software engineering that Ada supports
- Methods for effective use of the language
- Ada programming support environments (DoD, February 1980)
- The institutions, such as the DoD's Ada Joint Program Office (AJPO), that support the use of Ada (Sammet, August 1986)

The AJPO was established in 1980 to coordinate the introduction of Ada into the DoD by assuming such responsibilities as the validation of compilers and the dissemination of information about Ada. The existence of an official validation process is an extraordinary aspect of the Ada language. It prevents the proliferation of subsets, supersets, and dialects. The use of Ada became a matter of broad DoD policy with Department of Defense Directive 3405.1 (Dept. of Defense, 2 April 1987).

Because of Ada's merits and the breadth of its application domain, the use of the language has spread far beyond the DoD. The Federal Aviation Administration selected Ada for its new air traffic control system, the Advanced Automation System. The Aeronautical Radio, Inc., Airlines Electronic Engineering Committee (October 1987) has adopted Ada for avionic systems. Ada is also widely used in Europe (e.g., by the European Space Agency). Several NATO countries are collaborating to produce an Ada programming support environment. NASA has also selected Ada for a number of projects (see Section 5 of this report).

3.3 Support for Software Engineering

Ada is not an end in itself but rather a means to achieve the goals of software engineering.

Those who participated in the specification and design of Ada (e.g., Fisher, March 1978, p. 30) intended that use of the language would reduce the costs and increase the reliability and quality of software. The main promises of the Ada language are that it supports the principles of software engineering and, thereby, achieves the goal of cost-effective production of efficient, reliable, and maintainable software. For example, the language design emphasizes readability in order to promote reliability and facilitate maintenance. Ada features, such as the package construct, enable one to decompose a program into small parts that are easily understood and tested. The language design promotes reliability by facilitating automatic detection of errors. Language features enable one to write software designs in a compilable Ada-based design language. An Ada compiler can then check the consistency of the design.

Ada provides much more support for the principles of software engineering than other languages, such as FORTRAN and C. It is possible, however, to develop Ada programs in the style of such earlier languages, thereby losing the potential benefits of using Ada. Education in the principles of software engineering and in the use of Ada to support those principles is required to use the language's features most effectively.

Automated tools are key elements of state-of-the-art software engineering practices. Use of Ada is supported by a suite of tools that compose an Ada programming support environment (APSE). A number of APSEs with varying sets of capabilities have been produced by private and governmental organizations. An APSE always includes a compiler, text editor, and linker. Some APSEs contain tools, such as Ada-specific text editors and symbolic debuggers, that have indepth "knowledge" of Ada. Such tools have a high potential to increase software development productivity and reduce the attendant risks.

Boehm (September 1987, p. 52) states that a number of studies have predicted "a long-range savings of 40-50 percent for a fully mature Ada support environment and development staff." The cost of design may increase, but the expanded design efforts are expected to result in a net reduction in total development costs. For example, Ada's support for software reliability is expected to reduce the cost of testing.

Proper use of Ada is also expected to produce significant reductions in the cost of the maintenance phase. Industry spends between 40 and 70 percent of its total hardware and software budget on maintenance (Booch, 1987b, p. 8). The expected benefits in the maintenance phase are attributable to Ada's support for readability and isolation of code that is likely to change; to increased portability due to the standardization of the language (Fisher, March 1978, p. 30); and to reduced costs of training maintenance personnel in languages and the use of tool sets (Sammet, August 1986, p. 723).

3.4 Support for Reuse

Ada may produce its greatest reductions in life-cycle costs by increasing software reuse, that is, the use of components in more than one program.

Reuse has been a major topic of study and discussion for several years, but existing methods and tools have produced only a small increase in the amount of reuse. Through software engineering using Ada and related technologies, software components can become standardized in format and style, portable, and reflective of the entities in the problem space, as in *object-oriented design* (Booch, 1987b, ch. 5).

Indeed, "the ability to assemble a program from independently produced software components has been a central idea in [the] design" of the language (Dept. of Defense, 1983, sec. 1.3). Since the cost of a software system increases rapidly with its size, any technique that significantly decreases the amount of new software that must be specified, designed, coded, and tested has a potential for large cost reductions. Components can also be used to build prototypes of some parts of a system (Booch, 1987a, sec. 18.2) to validate requirements and thereby avoid costly requirements changes at later stages.

To realize the potential of software reuse, additional research and development will be required to understand optimum design characteristics; the problem domains in which reuse is feasible; retrieval-and-use techniques; portability; and other pertinent issues. Software reuse is an integral part of the Ada culture and has unmatched potential to reduce software costs, but the best ways to approach it are not yet completely understood.

3.5 Advantages of a Common Language

Language commonality has advantages that are largely independent of the chosen language.

Commonality allows an entire software development organization to acquire and use a single set of language-specific tools. Personnel can be transferred among projects without retraining either in a language or the use of tool sets. Since the organization's training program can focus upon the common language and associated methods and tools, it can be simplified. Similarly, the organization can have a single set of software standards, policies, and procedures tailored for the common language (Fisher, March 1978, p. 26). The use of a common language that is supported on many processors makes it possible to port programs from one environment to another with relatively little change. For all these reasons, language commonality should lead to a reduction in total software costs.

To realize the promises of Ada, an organization must be willing to make changes to incorporate both Ada technologies and software engineering principles into its software development process. Changes to the training program, software tool suite, and software standards and practices are necessary parts of the adoption process. Section 4 presents a model of the essential aspects of Ada adoption.

4 — A Model for Transition to Ada

A transition to Ada and state-of-the-art software engineering requires changes in an organization's approach to software development and management.

Section 3 has presented the improvements in the software development process and its products that are expected to result from the effective use of Ada and software engineering. However, to attain these benefits, an organization may have to modify older approaches and incorporate new software technologies. These technologies, in turn, may require changes in training programs, management practices, support organizations, standards, acquisition policies, programming support environments, research programs, and methods of measuring success. Few organizations have yet had time to make such changes, since Ada and associated technologies have evolved so recently. These changes require careful planning, take considerable time, and have certain costs that must be weighed against their benefits.

Section 4 is not intended as a set of recommendations for NASA. Rather, it is a model of how any organization that acquires or develops software could evolve to the use of Ada and update its software engineering practices. The question of whether such evolution would be beneficial for NASA is considered in Sections 6 and 7.

The transition model consists of changes in training, experience, tools and methods, management, infrastructure, research, and measurement. The model is based on reports by Basili et al. (April 1987), Softech (15 November 1987), Foreman and Goodenough (May 1987), Humphrey et al. (September 1987); on interviews with personnel at four NASA contractors now undergoing a transition to Ada (TRW, Hughes Aircraft, McDonnell Douglas Astronautics, and General Electric); and on conversations with NASA personnel.

The transition model includes not only Ada but also software engineering technologies because effective use of Ada is predicated on the use of sound software engineering principles. As was stated in Section 3.3, Ada is a means to an end; it is not an end in itself. It would be more effective to adopt sound software engineering principles without Ada than to adopt Ada without sound software engineering principles.

Section 5 describes current NASA activities in Ada and software engineering. Section 6 then compares the current NASA activities with the model for transition presented in this section. Section 7 presents recommendations for changes in NASA's current practices and plans in light of the findings in Section 6.

4.1 Building a Knowledge Base

To realize the potential of software engineering and the Ada programming language, an organization must make major changes in its training program.

Training programs dealing with Ada, software engineering, and management of Ada projects must be brought into an organization. Courses and curricula could be developed in-house, obtained from the public domain, purchased, or taught by contractors.

4.1.1 Ada Training

Studies have repeatedly emphasized that effective application of Ada requires adequately trained software developers and managers.

"It is of particular importance that both contractor and customer be well trained in Ada technology. . . . Ada software development includes training for all associated personnel, including managers, with a heavy emphasis on software engineering" (Basili, April 1987, pp. xv, 2-7). "Management training [in Ada] is especially critical" (Foreman and Goodenough, May 1987, sec. 9.1). Ada is a complex language and requires a longer learning period than a language such as FORTRAN. Ada courses should emphasize that Ada by itself does not guarantee good software but must be used as a means to implement good software engineering practices: "Ada is not merely a programming language; it is a vehicle for new software practices and methods for specification, program structuring, development and maintenance" (Brooks et al., September 1987, sec. 4).

Both managers and developers must undergo extensive training in Ada as part of the transition to the use of the language and the adoption of improved software engineering practices. Because of the size, complexity, and advanced features of Ada, training must be more extensive than is required for other classical development languages. Ada syntax, advanced Ada features, designing with Ada, real-time programming with Ada, and Ada for managers are typical topics that should be addressed. All training must involve hands-on programming exercises and should be aimed at preparing students for immediate application of the principles they have learned to a software development project.

The duration and schedule of Ada training may vary across organizations (e.g., Murphy and Stark, October 1985). Developers may require more than 3 or 4 full weeks of classwork, while managers would probably require much less Ada classtime to cover material pertinent to their needs.

4.1.2 Software Engineering Training

Training in Ada also requires training in software engineering principles, methods, and tools.

As noted in Section 3.3, the use of sound software engineering principles is more important and potentially more beneficial than the use of the Ada language alone. For this reason, training in effective software engineering techniques is more important to a software development organization than training in Ada. The evolution to state-of-the-art software development practices cannot be successful without extensive training in and use of software engineering principles. Repeatedly, studies have shown and leading software engineers have stated that much heavier emphasis in classroom time should be placed on training in software engineering than on training in the Ada language (e.g., Softech, 15 November 1987, pp. 3-25, 3-26). Organizations must usually provide software engineering training for their own employees, since it is not usually available from universities.

An effective training program in software engineering would have the following characteristics:

- Comprehensive—The training program should not consist of merely one or two courses addressing a few topics. The program should consist of numerous units addressing a wide spectrum of software engineering principles and activities.
- Consistent—The content of a given course should remain nearly constant through repeated offerings.
- Practical—Courses should be developed by senior software engineers who recognize the characteristics and needs of the software development organization and tailor the courses accordingly.
- Applied—Like Ada training, software engineering training must be used in near-term software development applications, such as pilot projects, to be effective.

The key elements that should be part of the software engineering training program include, at a minimum, the following:

- Requirements analysis
- Software design approaches
- Software assurance
- Verification and testing techniques
- Development methods and tools
- Software metrics

4.1.3 Management Training

Management training should prepare managers to deal with the changes in the software development process that result from a transition to Ada.

Aspects of the management process affected by a transition to Ada include

- Process model
- Scheduling
- Software size estimation
- Resource allocation
- Environments
- Acquisition control
- Reviews
- Product characteristics
- Problem indicators
- Risk management

Management training should point out that Ada source code may differ greatly in size from FORTRAN source code with equivalent functionality. In addition, the use of Ada requires significant changes in the allocation of staff and effort across development phases, e.g., design, code, test, integration (Foreman and Goodenough, May 1987, pp. 15-16). The use of Ada may also require additional allocation of resources early in the project to support proper training and experience in the newly adopted technologies. Finally, Ada management training should prepare managers to deal with new forms of information, such as compilable design language, that the developer is required to deliver for particular reviews.

4.2 Building an Experience Base

An experience base of understanding and lessons learned from both pilot projects and production projects in Ada is vital for the effective use of the language.

Reports by the Software Engineering Institute (SEI) (Foreman and Goodenough, May 1987, sec. 9), the MITRE Corporation (Basili et al., April 1987, sec. 2.1.7), and the Software Engineering Laboratory (SEL) at GSFC (Murphy and Stark, October 1985, p. vi) have strongly recommended that all developers and managers who expect to use Ada and effective software engineering techniques must participate in hands-on development or management under the supervision of an experienced software engineer. Some of this experience may be obtained through a practice problem that is part of the training sessions. However, students should apply the technology in a production project within 6 months after the classroom training. Lectures and seminars that are not immediately followed up with practical application of the principles they teach may be wasted efforts.

Pilot projects are not only important for training purposes but also to prepare an organization for the use of Ada on production projects. Pilot projects provide the initial data for an organization's experience base and allow for the measurement and understanding of the new technology as applied to a production environment (Basili et al., April 1987, sec. 2.1.7; Voigt, 1985, p. 107).

Pilot projects should involve relatively unsophisticated and noncritical systems or parts of systems, e.g., tools, benchmarks, prototypes, simulators, and test drivers. Experience to date shows that, on the average, benefits of Ada are not realized until the third project by a given development organization. Typical initial Ada projects encounter a 10- to 30-percent overhead due to learning curves and other transition costs (Reifer, December 1987). Ada projects developed by organizations with one or two previous Ada projects are expected to break even as compared to projects using other high-order languages. The benefits of using Ada and related development techniques are expected to be realized only in the longer run (Boehm, September 1987,

p. 52). An organization must, therefore, plan for the evolution to an Ada environment by conducting smaller, pilot projects before initiating efforts on large, critical systems.

As Ada is subsequently used for production projects, the experience base should continually evolve, providing lessons learned and guidance for future Ada projects. Information in the experience base should describe the degree of success obtained with given features of the Ada language; methods and tools; and standards, such as life-cycle models.

4.3 Tools, Methods, and Reusable Components

Transition to Ada requires the availability of new software development tools, methods, and reusable components.

Since software development is an expensive, labor-intensive process, it is cost effective to invest considerable sums to automate it. The development of any nontrivial software system requires a minimal set of tools, such as an editor, compiler, and symbolic debugger.

Prospective tools should be evaluated for functionality, efficiency, capacity, compatibility with each other, quality of documentation, and maturity. If results of essential benchmark tests for efficiency and capacity cannot be obtained elsewhere, they should be performed (Foreman and Goodenough, May 1987, sec. 4.2-4.3; Basili et al., April 1987, sec. 2.1.6, 3.4). An organization that evaluates tools should archive results to avoid duplication of the considerable effort required.

Methods for software development that have been used successfully for many years in the development of FORTRAN or COBOL systems are not entirely adequate in the Ada context. Such methods provide no guidance in the use of Ada features such as packages and tasks. Software development organizations adopting Ada must adopt new ways to address these problems, e.g., object-oriented design (Booch, 1987b; Nielsen and Shumate, 1988).

Detailed software development methods, such as a particular version of object-oriented design, should normally be selected at the project level, not at the company or government agency level. Such methods are far less mature than the major programming languages such as Ada. Moreover, their appropriateness varies greatly across kinds of project. This is why TRW, for example, does not prescribe a single company-wide Ada method but teaches a variety of approaches in its in-house courses. TRW does, however, prescribe required properties that must be satisfied by the methods chosen. For example, "Verify consistency and completeness of requirements and design specifications from both a functional hierarchy and a data-flow standpoint."

If a large company should normally not have a single software development method, there is all the more reason why a government agency should not normally have a detailed, official methodology that it imposes upon its contractors (Gardner et al., February 1988, sec. 1.1.3). A government agency can, however, impose both a set of auditable standards on the required products developed by the chosen method and a set of milestones and reviews at which the compliance of the products with the required standards is demonstrated and assessed.[1]

An organization planning to use Ada to develop multiple systems with potential commonality of components should establish, populate, and use a library system containing reusable Ada software. The organization must establish an infrastructure to create the library system, populate it with suitable components developed for projects or specifically for the library, and ensure that the components are used when appropriate.

4.4 Management

Evolution toward the adoption of Ada as an organization's principal programming language requires planning at the organizational level and software risk management planning.

4.4.1 Organizational Planning

A key element to ensure a smooth transition to Ada is an Ada transition plan that includes a training plan.

As with any new technology, the introduction of Ada requires planning. First, an organization must determine the goals that it intends to reach in terms of Ada technology in 5 or 10 years. The transition plan should then specify the rate of approach to the goal, the resources to be applied, and the projects to which the resources should be applied.

Because of Ada's complexity and the foundations in software engineering required for Ada's effective use, a training plan is a critical part of the

[1] Dr. Barry Boehm of TRW assisted us in drafting this paragraph and the one immediately preceding it.

overall transition plan. Since software engineering is not taught in most universities, an organization must make this training available to its employees. The training should include an entire curriculum with different paths depending on the individual's function (e.g., manager, development engineer, or assurance engineer) and on the individual's previous training and experience. Reports are available (e.g., Softech, 15 November 1987) that define sample curricula appropriate for the transition to Ada.

4.4.2 Risk Management

Software projects using new software technology must incorporate risk management as an integral part of the management process.

Managing Ada risks is a special case of software risk management. Requiring projects above a certain size to develop and carry out a formal program to manage software risks, including Ada risks, should be part of a transition to Ada and more effective software engineering (Boehm, April 1988). As Ada technology matures, Ada-specific risks will become less prominent among the overall risks of an Ada software development.

The Language Panel at NASA's Open Forum on Space Station Software Issues listed the following Ada risk factors:

- "The applicability of Ada to distributed, fault-tolerant, and hard [fast-response] real-time systems
- The efficiency of Ada run-time support environments and of code generated by Ada compilers, especially for tasking in real-time and distributed systems
- The development of good Ada implementations for the particular machine architectures that might be used for [a given project]" (Voigt, 1985, p. 107)

Two years later, Basili et al. (April 1987, sec. 3) still sounded the same alarm in recommending that the Federal Aviation Administration use Ada for the Advanced Automation System. In addition to those risks mentioned above, the later report mentioned risks pertaining to the ability of compilers to handle large program size, customer readiness for contract monitoring, availability and maturity of programming support environments, availability of qualified personnel, and the soundness of schedule and cost predictions in view of the paucity of previous Ada projects.

Peschel (13 December 1988) reports a survey of the major Ada risks in eight TRW projects. In descending order, the five greatest risks were found to be the following:

- Compiler performance and maturity
- Staff's experience with compiler and tools
- Ada tool set performance and maturity
- Budgeting and scheduling
- Staff's Ada design skills

4.5 Infrastructure

Transition to Ada and better software engineering requires suitable policies, standards, and internal support organizations.

4.5.1 Policies and Standards

A government agency evolving to Ada should have agency-wide standards that are compatible with Ada and with state-of-the-art software engineering principles.

It is possible for a government agency to eschew agency-wide standards and operate on the basis of standards developed by particular branches, departments, or projects. Such an approach, however, tends to waste both governmental and contractor resources. It means that many different groups within the agency, rather than a single central group, must spend time dealing with many of the same issues. It also means that the agency's contractors who deal with several parts of the agency may need to learn how to work under multiple sets of standards. By contrast, a uniform set of agency standards allows the contractor to economize on training and also may make it cost effective for the contractor to develop automated tools to support the production of a uniform set of deliverable documents and the verification of traceability between them.

Software development standards developed in a pre-Ada era may be incompatible with Ada and also with state-of-the-art software engineering principles. For example, a standard might presuppose functional decomposition rather than object-oriented design, abstraction, and information hiding

(Gardner, et al., February 1988, sec. 3). Again, a standard might presuppose the waterfall life cycle, thereby ruling out such innovative life cycles as the spiral model (Boehm, August 1986).

Revision of a government agency's standards should not be done in a vacuum but with extensive review by outside interested parties. The DoD uses an organization named the Council of Defense and Space Industry Associations as a vehicle for such reviews. This organization consists of various associations of defense and space contractors, such as the Aerospace Industries Association and the Electronic Industries Association. Such a mechanism for involving major corporations in the standards review process is advisable for any government agency.

4.5.2 Internal Organizations

An organization evolving to Ada should establish an internal structure to support the evolution.

An organization's transition to Ada requires gathering technical information from outside the organization and transferring information within it. Employees who are responsible for individual projects may not have sufficient time to keep informed about the latest developments in compilers, other tools, or methods that may be applicable to current or future projects. In a large organization, it is cost effective to establish an internal organization or set of organizations to perform such functions as the following:

- Planning the Ada and software engineering training program
- Gathering and disseminating information on tools and methods
- Planning and supporting software reuse
- Coordinating the collection and dissemination of metric data
- Assessing and revising the software engineering process
- Revising standards
- Coordinating research

4.6 Research and Development

A successful transition to Ada can be assisted by a research program on Ada and associated technologies.

Because Ada is relatively young, additional information about development methods, compilers, environments, and lessons learned is continually being generated. Hence, it is essential for an organization evolving to Ada to investigate the application of Ada and related technologies to that organization's projects. These investigations should not be performed as part of critical production projects, but rather as research and development that promotes the effective use of these technologies on production projects.

Such a research and development program should identify the technological needs of future projects. Critical technology issues should be studied and findings made available to the organization's projects. Ada-related issues might include determining the most effective methods for using the language for real-time or network systems; methods for developing and using reusable components; or life-cycle variations required for effective use of Ada.

4.7 Measurement and Assessment

A high level of software engineering maturity requires the adoption and use of software measurement concepts.

To determine the characteristics of a software development organization, one must have some mechanism for assessing the software development process and its products. Such measurable quantities as resource expenditures, reliability, testing effectiveness, and productivity must be extracted from the software process to determine the effects, strengths, and weaknesses of evolving technology. Without this information, there is no means of determining if, and to what extent, improvements are being made. For example, there is no way of determining whether a particular technology (e.g., Ada) has a positive or negative effect on the software development process or product.

Decisions about the management of an individual project should be made on the basis of metrics that provide such information as testing effectiveness, resource expenditures, key problem areas, or software errors. Metric information from projects should be archived in the corporate memory where it is available to new projects and where it also is used for continuous assessment of the organization's strengths and weaknesses. Such assessments help identify needed training, tools, and methods. The effects on productivity and reliability of process changes, such as the introduction of Ada, can be observed and used to refine the process further (Humphrey et al., September 1987, pp. 26-27, 40).

In the next section, NASA's use of state-of-the-art software engineering technologies, including Ada, will be explored with respect to this model for transition.

5 — Ada and Software Engineering in NASA

NASA has taken some of the steps required for a transition to Ada and to more effective software engineering.

5.1 Building a Knowledge Base

NASA has both agency-level and center-level training in Ada and software engineering; yet the number of NASA personnel who have completed Ada training is small.

An agency-level training program in software engineering topics including Ada has been established by the Software Management and Assurance Program (SMAP) in Code Q. However, the majority of NASA training in Ada and software engineering has been done through courses planned by center education offices or by individual projects. Current information describing courses in Ada and software engineering for NASA indicates that all centers have conducted at least a minimal level of Ada training. SMAP courses have been offered over 75 times with a total attendance of over 1500. (It cannot be determined how many distinct individuals are included in this total.) Johnson Space Center (JSC) indicated that approximately 300 employees have participated in software engineering courses, some of which were Ada courses, at the University of Houston-Clear Lake. GSFC estimated that approximately 60 persons have had some Ada training. A review of all NASA centers indicated that the total number of employees who have attended at least one Ada course is approximately 500, of whom fewer than 200 have subsequently used the training on an Ada project.

A recent survey (Softech, 15 November 1987, sec. 5.5.2) reported the following results:

> Twenty-seven percent of the respondents state that half or more of their technical staff have some form of software engineering training. . . . Fifty percent of the respondents state that less than one-fourth of their technical staff have been exposed to training in Ada in any form.
>
> Over 50% of the respondents [stated] that less than one-fourth of their management staff has received software engineering training and nearly 70% of the respondents said that less than

one-fourth of their management personnel have received Ada training.

On-the-job training, then, is the most commonly used training technique.

Softech developed a comprehensive plan for Ada and software engineering training across NASA. The plan is based on previous work that Softech did in developing Ada training for the DoD. This plan contains two main components (Softech, 15 November 1987, sec. 3.7, 4.5): (1) a core curriculum comprising 18 courses and (2) a procedure by which each center can supplement the core curriculum with a center-specific system of short courses and on-the-job training.

The core curriculum includes courses on such topics as the following:

- Software engineering with Ada
- Ada project management
- NASA software life cycle and standards
- Software requirements analysis
- Software design specification
- Quality assurance
- Configuration management

Softech (15 November 1987, p. 3-30) also recommended the continuation of training with pilot or other project work under the supervision of an experienced software engineer.

We were not able to identify any evidence that this comprehensive training plan has been implemented in any portion of the agency.

5.2 Building an Experience Base

All NASA centers have completed some Ada projects, some of which are small pilot projects; and all centers have indicated that they plan to develop additional projects in Ada.

The growth of the Ada experience base at NASA can be measured by the number of completed Ada projects developed by NASA personnel or developed for NASA by support contractors. Softech (15 November 1987, pp. 2-1, C-5) identified a total of 20 Ada projects completed by or for NASA as of September 1987 and a total of 150 projects planned before 1992. Most of the completed projects are small training and pilot projects, but the planned projects include some large systems intended for operational use. Excluding the SSFP, these projects will require a total of 1200 staff-years of effort.

The Softech report also summarized the Ada experience level of managers, technical staff, and support personnel. There were 24 survey respondents who were responsible for approximately 1400 personnel. The report states that the "average level of experience in Ada related projects for the sample population of this study was zero for management and support personnel and under six months for technical personnel" (Softech, 15 November 1987, p. 2-1). Since most of the survey's respondents were personnel associated with Ada at the NASA centers, it is unlikely that the rest of the population (those not responding to the survey) would raise the average level of experience.

During the fall of 1988, the ASMAWG attempted to update Softech's findings on Ada experience by contacting key personnel at most centers. The ASMAWG determined that additional Ada projects are planned but that few additional Ada projects were completed between September 1987 and October 1988. The Ada experience base for NASA has grown somewhat but not significantly in the past year. However, NASA's Ada experience base is probably no less than that in most organizations responsible for producing or acquiring large amounts of software. Because of the newness of Ada, a shortage of experienced Ada managers and developers currently exists; that shortage will persist for the foreseeable future.

NASA's most significant commitment to Ada obviously has been that of the SSFP. The program commissioned three studies, to be completed by the summer of 1985, to determine whether Ada should be chosen for the program. All three recommended that Ada should be chosen as the principal language for the Space Station Freedom. TRW Defense Systems Group (11 April 1985, sec. 1.4.1.2) recommended that special-purpose languages could be considered on a case-by-case basis for use in such limited areas as testing, artificial intelligence, and data base management. SSFP project management then mandated that all operational software would be written in Ada unless a closely scrutinized waiver were granted. Other projects associated with the SSFP have taken that program's lead and have selected Ada for their software development.

Since the Softech report was written, additional projects have selected Ada. These projects include the Orbiting Maneuverable Vehicle (over

50 KSLOC), Aero-Assist Flight Experiment (over 200 KSLOC), and Secure Shuttle Data Systems at Marshall Space Flight Center (MSFC). GSFC has developed over 500 KSLOC of Ada code for flight dynamics projects over a 3-year period. In addition, GSFC is using Ada for the Explorer Platform onboard system (5 KSLOC) and the Second TDRSS[1] Ground Terminal (more than 500 KSLOC). At the Jet Propulsion Laboratory (JPL), numerous production projects plan to use Ada, including Real-Time Weather Processor (over 72 KSLOC) and the Network Operations Communications Center Upgrade.

In addition to the projects that have already selected Ada, numerous other projects have indicated a serious interest in it. As NASA's Ada experience base grows, a significant number of additional projects will probably select Ada as the software development language.

5.3 Tools, Methods, and Reusable Components

NASA software is developed using a wide variety of tools and methods.

Many of the Ada projects at GSFC have been developed on a Digital Equipment Corporation (DEC) VAX-11/780 or 8600 using the DEC compiler. However, GSFC's Explorer Platform used the Interact cross-compiler hosted on a DEC MicroVAX and targeted to the Military Standard (MIL-STD) 1750A processor. MSFC is developing the Secure Shuttle Data System using Concurrent Computer's Ada compiler for the Perkin-Elmer 3244 but is developing the Orbiting Maneuverable Vehicle using the TLD Systems Ada compiler for the MIL-STD-1750A processor. Various Ada projects at JPL are being developed on DEC, Rational, and Gould environments.

NASA Ada methods are comparably diverse. Developers in the flight dynamics area at GSFC have formulated a general object-oriented software development method (Seidewitz and Stark, July-August 1987). The Real-Time Weather Processor at JPL used a tailored version of part of this approach for preliminary design but applied Yourdon, Ward, and Mellor techniques for requirements specification. By contrast, the Power Management and Distribution Photovoltaic Testbed at Lewis Research Center (LeRC) used George Cherry's Pictorial Ada Method for Every Large Application (PAMELA 2).

[1] Tracking and Data Relay Satellite System.

The SSFP's Software Support Environment (SSE) will be the first complete computer-aided software engineering environment created to meet NASA specifications. It will consist of the "rules and tools" for developing Ada software for the Space Station Freedom. Some of the tools will be purchased commercially and some will be custom built. The SSFP has developed a well-defined software management, development, and assurance process, which will be automated by means of the SSE. The process will be controlled and traceable from its very beginning. The SSE will support a library of reusable components and will represent an investment of $151 million (in 1987 dollars), not including a 3-year optional extension.

5.4 Management

NASA has not adopted an Ada transition or training plan or an approach for managing the associated risks.

5.4.1 Agency-Level Planning

NASA has not adopted a plan for agency-wide transition to Ada or for the training that such a transition would require.

Neither the agency nor any center has adopted a plan for evolving to Ada. Some projects have selected Ada and have formulated the necessary training and software development plans. Softech (15 November 1987) recommended an Ada and software engineering training plan (Section 4.1, above), but the agency has neither formally adopted it nor established a mechanism to implement it. It is up to individual projects to plan and implement their own training plans in Ada and software engineering.

5.4.2 Risk Management

Currently, no agency-level policy or standard requires the formulation and implementation of a software risk management plan.

No risk management plan appears on the NASA Software Acquisition Life Cycle chart published by SMAP (version 3.0, 15 October 1986). However, version 4.3, due for release in March 1989, does address risk for information systems and software. The new baseline of SMAP's *Information System Life-Cycle and Documentation Standard* includes a risk management plan (see SMAP-DID-M910 Risk Management Plan Data Item Description).

The SMAP baseline is not an agency-wide standard and is not even an agency-wide policy.

Although software risk management plans are rare in NASA projects, a few individual projects have heeded the warnings about risks discussed in Section 4.4.2. For example, Nelson (1988, pp. 121-122) reports that

> Each of the Space Station [Freedom] Program software managers has addressed the issue of risk management in their software management plans. Some of the technical concerns identified . . . include interoperability . . . , operating system performance, . . . portability, Software Support Environment System immaturity and Ada performance. Some of the steps that are being taken to mitigate these concerns include rapid prototyping, engineering test beds and reevaluation of requirements.

The Real-Time Weather Processor Project at JPL (Molko and Loesh, 1 December 1988) has developed and is implementing an Ada risk management plan very similar to the one that Basili et al. (April 1987) have recommended.

5.5 Infrastructure

> **NASA has a small number of agency-level standards and agency-level internal organizations supporting software; all have limited scopes.**

5.5.1 Policies and Standards

> **NASA has only one official agency-level product and a set of draft documents applicable to software development.**

NASA Management Instruction (NMI) 2410.6, issued in 1979, governs the management of flight software. Its major requirement is that the project manager for each flight project will produce a Software Management Plan. The NMI requires that this plan describe the project's management approach (e.g., management responsibilities and mechanisms, configuration management techniques, and quality assurance procedures) and technical approach (e.g., processes and schedules for requirements definition, implementation, and testing). However, the NMI imposes no substantive requirements on either the management or technical approach.

SMAP has baselined a document called *Information Systems Life-Cycle and Documentation Standards*. It defines a life-cycle model for software systems and specifies the format of deliverable documentation. These descriptors have not been adopted as NASA standards through a NASA-wide policy; consequently, they often are not applied in major software projects. SMAP is also preparing a set of guidebooks on several topics in software management and assurance. Finally, SMAP is formulating a software management and assurance policy to be issued as an NMI, replacing NMI 2410.6, and addressing prototyping, flight software security issues, and several other assurance issues.

In the absence of NASA policies for software other than NMI 2410.6, each center, program, and project office is responsible for establishing supplementary software development policies, standards, and guidelines for its own use.

SMAP has adopted the practice of seeking contractor input into the process of standards development by inviting industry representatives to its workshops. SMAP has also asked the centers to solicit contractor comments on draft standards and to incorporate them into the center's comments.

5.5.2 Support Organizations

Three main organizations concern themselves with software at the headquarters level: the Automated Information Management (AIM) Council, the Inter-Center Committee on Automatic Data Processing (ADP), and SMAP.

The AIM Council addresses management and technical issues of the institutional automated systems. Its main purpose is to create a standard environment for the development and support of agency-wide institutional systems, such as payroll, equipment inventory, and personnel.

The Inter-Center Committee on ADP draws its participants from the managers of NASA's large central computer facilities. It deals primarily with the management regulations pertaining to long-range ADP planning and acquisition planning and with acquisition issues raised by external regulatory agencies such as the General Services Administration and the Office of Management and Budget.

SMAP, which is guided by an intercenter steering committee, was chartered in 1983. Its responsibilities include formulation of life-cycle standards, documentation standards, and guidelines for software development. SMAP's training courses (Section 5.1), life-cycle standard, documentation standards, and guidebooks (Section 5.5.1) are parts of a broader plan according to which SMAP will serve as the central agency-level source of policies, procedures, information, consultation, and planning in the area of software management and assessment. However, none of the additional products or services envisioned in the broader plan are available at this time.

Most centers (e.g., GSFC) have an organization separate from software development that is responsible for the assurance of major software projects developed at the center. In recent years, such organizations have made some strides in recognition and in impact on software projects. However, they are typically a relatively small component of a center and do not have adequate staff to carry out an effective program.

Other support functions such as independent verification and validation, configuration management, software technology assessment, software process engineering, and technology information centers are not established parts of the agency but are supported locally by the center or by individual projects.

One exception is the commercially operated Ada Technology Transfer Network, called AdaNet. Located in West Virginia, this network has been partially supported by NASA.

5.6 Research and Development

NASA has supported research in the area of software engineering (e.g., methods, management practices, reliability) for at least the past 12 to 14 years; but little of it concerns Ada.

NASA's major research activities are centered in the Office of Aeronautics and Space Technology (OAST). This office defines various programs of research across a broad spectrum of technologies that are of interest to NASA projects. These technologies include computer science and, in particular, software engineering. OAST delegates responsibility for research in particular areas to various NASA centers on the basis of their interests, capabilities, staffing, and programs. OAST does not restrict itself to working with the three NASA research centers—LeRC, Langley Research Center (LaRC), and Ames Research Center (ARC)—since another center may have the best facilities to perform a particular research program.

One center is responsible for defining, managing, and carrying out each major OAST research program. OAST is responsible for integrating and coordinating the full effort across all centers. Such programs as the NASA Initiative on Software Engineering (NISE) and the High Performance Computing Initiative are examples of research and development (R&D) programs directed by OAST.

Other NASA program offices support R&D efforts that are pertinent to their specific needs. The Office of Space Operations (OSO) may support research in communication technology, while the Office of Space Science and Application (OSSA) may support research in sensor technology. Since software technology is vital to nearly every program office, each office supports at least some software research.

The research supported by OSO, OSSA, and other program offices is managed in nearly the same way as the research supported by OAST. A center, which need not be a research center, proposes research that a program office may consider relevant to its concerns. The program office may then support the R&D at that center. Research at a given center may be supported by more than one program office.

When research is completed at a center, technology transfer is the responsibility of the sponsoring program office. The infusion of advanced technology into specific application areas is not the principal responsibility of any agency-level organization.

Until recently, the agency has not had a coordinated research program in which multiple centers have attempted to cooperate in order to achieve a single well-defined objective. Such cooperation is the aim of the NISE, which began in fiscal year 1988. Its major concern is the management and development of complex, highly reliable software systems. At least four centers (JSC, LaRC, GSFC, and JPL) have active roles in the coordinated effort. NISE supports research in critical areas of software engineering but currently has no plan to address any specific Ada issues.

The following subsections describe NASA research programs in software engineering.

5.6.1 Goddard Space Flight Center

The Software Engineering Laboratory (SEL) at GSFC has been carrying out research in software engineering for over 10 years.

At GSFC, one division has been carrying out experimental studies in the effectiveness of software technologies. An informal organization, the

SEL, was established at GSFC to carry out the research. The primary function of the SEL has been to identify potentially useful software technologies and then to evaluate them as applied to flight dynamics production software projects. The mode of operation is to train development teams in the use of some method, language, or tool and then measure its effects on the software process and products. Although the SEL has been functioning in one particular environment, many of the results are relevant to, and have been applied to, a broad spectrum of software development areas. Ada is one technology that has been under study since 1985; in fact, it has been the SEL's principal subject of study during this time.

5.6.2 Jet Propulsion Laboratory

JPL sponsors the Systems, Software, and Operations Research Center (SSORCE) and the Ada Development Laboratory.

The purpose of the SSORCE is to infuse state-of-the-art software engineering technology into practice. It does so through education, consulting, development of standards, and the introduction of automated software tools. More recently, JPL established the Ada Development Laboratory as an environment for teaching Ada and developing Ada software for specific applications and pilot projects.

5.6.3 Johnson Space Center

JSC has performed extensive research in Ada-related technologies and other areas of software engineering during the past 5 years.

The major efforts at JSC were coordinated through an Ada beta test bed established to evaluate Ada with participants from local industry and a local university. This program's research on Ada-related issues has produced a growing awareness of the complexity and potential of the language. The Software Engineering Research Center at the University of Houston-Clear Lake is also also doing research for JSC.

5.6.4 Langley Research Center

LaRC has been carrying out research in software engineering for over 10 years.

Much of the LaRC research has involved the development and study of techniques and tools intended to improve software reliability. LaRC has also carried out some research related to Ada tasking and has directed and coordinated computer science and software engineering research at several universities.

5.6.5 Other Centers

ARC, MSFC, and LeRC have also conducted some research in software engineering.

The other centers have also carried out software research. For example, ARC has carried out research in parallel processing and in artificial intelligence.

5.7 Measurement and Assessment

NASA has two significant programs of software measurement and assessment.

One of the goals of JPL's SSORCE program, which was established in 1985, is to assess software development methods. In addition, the SEL at GSFC (Section 5.6.2) has been performing software measurement and assessment since 1977. The SEL has collected measurement data from over 65 flight dynamics projects and has used these data to study the effects of software development technologies, including Ada. Such data as resource usage, software error characteristics, product complexity, and software quality are recorded and archived for study and for use in creating new development guidelines and management aids. Ada has been part of this study effort since 1985.

6 — Findings

This section contains the ASMAWG's findings on NASA's use of Ada and its software engineering practices.

6.1 Appropriateness of Ada

Ada is an appropriate vehicle to support the evolution to improved software practices in NASA.

NASA's many past successes in missions that depended heavily on software attest to the strength of its software development approach. However, to meet the challenges posed by the increasingly voluminous and complex software that the agency will require, an evolution to new software technologies, such as Ada, is critical.

The ASMAWG assessment indicates that Ada technology (the language, compilers, other development tools, and methods) has now matured to the point that the claims for Ada discussed in Section 3 are credible for many applications. Specifically, Ada provides a high degree of support for the principles of software engineering and thus for NASA's goal of cost-effective development and maintenance of large, complex software systems that are also efficient and reliable. More than any other major programming language, Ada promotes readability, maintainability, extensive checks during compilation and execution, reuse of general-purpose software components, and compilable designs.

In addition, standardizing on Ada would enable NASA to take advantage of the massive investments that the DoD, its contractors, and their vendors are making in Ada technology as a result of the DoD's Ada mandate. Because of these investments, Ada is "doomed to success," as Michael Deutsch put it in a briefing at Hughes Aircraft. That is, problems that afflicted Ada in its early years, such as immature compilers and other tools, are being overcome because they have to be. NASA can benefit from these efforts without paying much of their costs.

Selecting a principal programming language, regardless of what it is, has great advantages to NASA. The training of personnel; acquisition and development of tools, methods, and reusable components; development of standards and policies; and performance of software research may be much more economical if they concentrate upon a principal programming language. This consideration applies both to work done by NASA and by its contractors. Savings that contractors achieve through the use of a principal language should be passed on to NASA in the form of lower costs for work done under contract.

In addition, the adoption of a principal programming language would assist NASA in supporting the existence of a wide spectrum of potential vendors. Knowing where NASA intends to go, vendors of compilers and other language-specific tools could better anticipate future markets and create the needed products. Potential contractors would also be stimulated to train personnel in the new language and undertake projects to gain experience with it.

Finally, NASA and the DoD, especially the Air Force, have similar requirements for computer processors. For example, NASA makes some use of the Air Force's standard processor specified by MIL-STD-1750A. In the future, NASA will probably continue to want to use newly developed processors that the DoD uses. Ada compilers are more likely to be supported for these future processors than compilers for other languages for which the DoD will have little use. It will, therefore, be easier to port NASA software to these processors or to create new software for them if that software is written in Ada.

Because of the SSPF's Ada mandate, the technical capabilities of the Ada language, and the increasing availability of excellent Ada tools and techniques, the role of Ada in NASA will inevitably grow. The agency could passively allow such a transition to occur through the uncoordinated choices of individual project managers over several years. However, the transition will be faster and more efficient if NASA adopts a firm mandate for Ada. A mandate will both stimulate and channel the energy that is required to bring about such a technological transition.

Because of the newness of Ada, there currently is very little empirical evidence showing the cost-benefit ratio of applying the language. However, some of the SEL data does indicate positive trends in a series of Ada projects carried out in a single environment, the flight dynamics area at GSFC. These projects show an increase in the percentage of code that is reused, with values that are already well above those typical of comparable FORTRAN projects (Figure 6-1). They also show increases in productivity of about 50 percent from the first-time to the third-time Ada projects (Figure 6-2). These studies concluded that the productivity on a third-time Ada project would result in a slightly lower cost than that of an equivalent FORTRAN project. Finally, these projects show significant decreases in error rates (Figure 6-3).

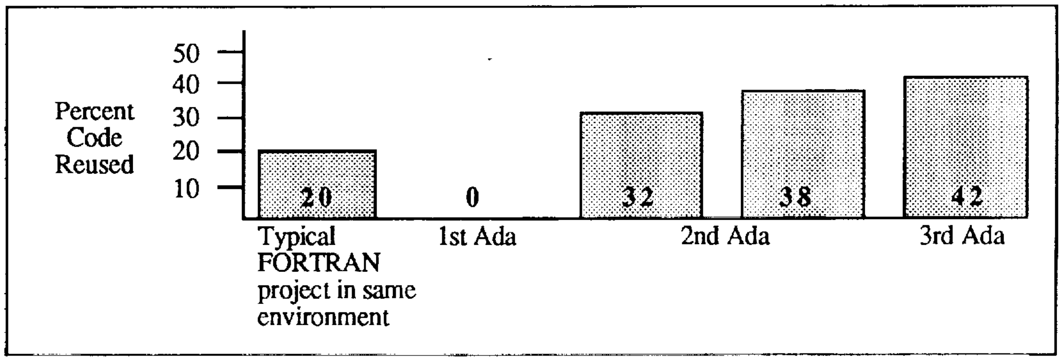


Figure 6-1. Reuse Trends

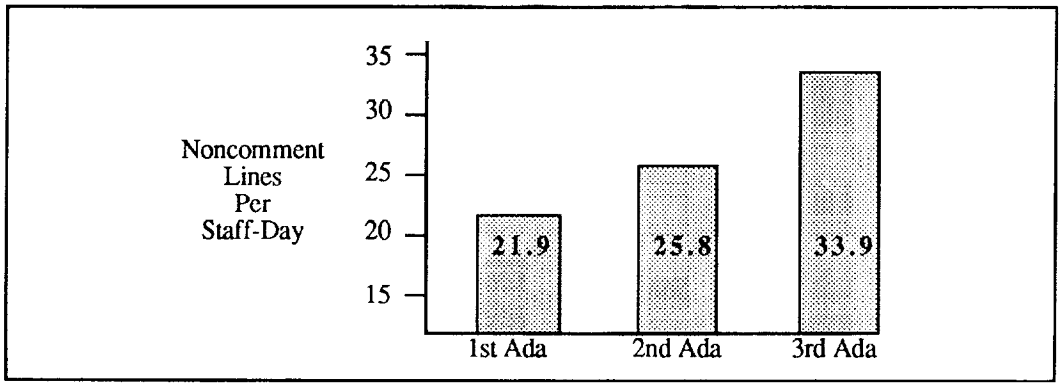


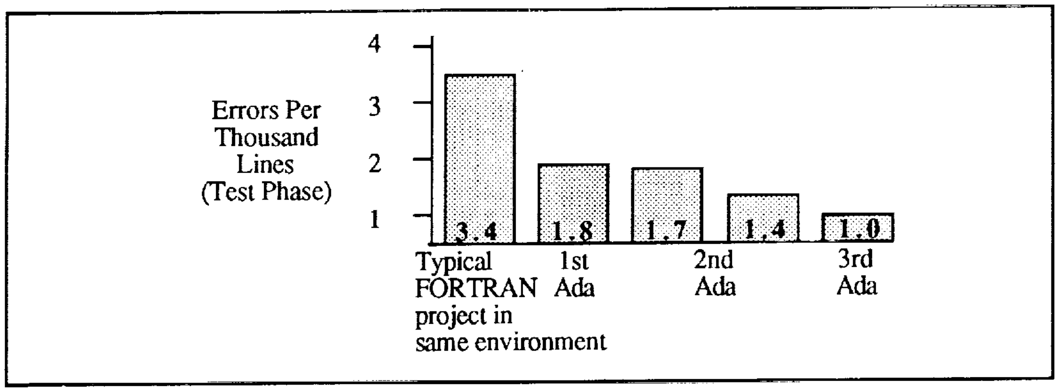
Figure 6-2. Productivity Trends

Many of the remaining findings in this section are based on this fundamental assumption about the desirability of a transition to Ada.

6.2 Knowledge Base

Although NASA offers courses in Ada and software engineering, the training programs are not adequate to accomplish a transition to Ada.

The number and variety of courses in Ada and software engineering offered at the NASA centers reflect the great importance that the agency places on these disciplines. Nonetheless, the Ada knowledge base across the



Data is based on McGarry et al. (December 1988).

Figure 6-3. Error-Rate Trends

agency is small (as it is in industry, too). This situation is partly due to the unique characteristics of Ada and associated technologies, which require more extensive training than is needed for most programming languages. However, inadequacies do exist in NASA's training programs. As noted in Section 5.1, a 1987 survey found that the percentages of NASA management personnel who have received training in software engineering and Ada are low. Often, NASA personnel assume that "long NASA tenure qualifies as software engineering training" (Softech, 15 November 1987, p. 5-11).

SMAP and the center education offices have offered a large number of courses. However, the training programs have not been derived from empirical measurement and assessment of the current capabilities and needs of the agency in software engineering and Ada. In addition, with very few exceptions, the training programs for Ada have not been coordinated with pilot projects or other development efforts in Ada. Moreover, a shortage of adequate funding and time planned for employees' participation exists. Unless reinforced by experience in Ada development, much of the knowledge gained through Ada training will soon be lost.

Additional shortcomings in NASA's current approach to training are the following:

- The agency has not measured the effects of training programs on subsequent projects.
- The center education offices have not implemented the Softech (15 November 1987) recommendations or any other plan for full training in Ada and software engineering.

- The center education offices have not implemented an organized sequence of courses suitable for given classes of employees, as recommended by Softech (15 November 1987, sec. 3.7.1).
- Managers have not required their subordinates to acquire Ada training by taking suitable sequences of courses.

Without significant modification of current approaches to training in Ada and software engineering, the evolution to Ada will require too much time, will involve too many missteps, and may never take place at all.

6.3 Experience Base

NASA has very little experience with Ada, and current plans do not provide for building an experience base adequate to meet existing commitments for the use of Ada.

Although production-quality compilers have been available since 1985, and although NASA made a serious commitment to Ada as early as 1985 (for the SSFP), the agency as a whole has made only small efforts to use Ada on projects in order to build an experience base. Through fiscal year 1988, the total number of Ada projects completed by or for NASA, where NASA directly participated in development or management, is fewer than 20. Most of these projects were located in one of two or three centers. Of the Ada work NASA has done, the percentage of studies, as opposed to production projects, is too high. Our estimate is that fewer than 1 out of 50 NASA software developers or managers has had any practical experience in developing, managing, or acquiring Ada software other than small classroom projects.

NASA cannot establish an adequate Ada experience base solely through training programs and without direct participation in the management and development of Ada systems. Without a concentrated effort to rectify its current lack of Ada experience, the agency will be inadequately prepared to develop, manage, or acquire Ada production systems over the next 5 to 8 years.

6.4 Tools, Methods, and Reusable Components

NASA Ada projects have often selected or developed sophisticated tools and methods; but software reuse is still in a rudimentary state.

The examples described in Section 5.3 indicate that the tools used in NASA Ada projects vary widely. In some cases, they are mature and reliable. In other cases, projects have been severely limited in their choice of tools because of a requirement to use a particular processor. Consequently, some of these projects have suffered from poor quality or unavailability of tools such as compilers and symbolic debuggers. The SSFP's Software Support Environment is a commendable effort to assemble or create a set of tools and procedures especially suited to the development of large NASA projects.

On the whole, the few completed NASA Ada projects have tended to use state-of-the-art Ada methods and have sometimes contributed to the advancement of the technology (see Section 5.3).

Some NASA projects have achieved high levels of software reuse, e.g., the flight dynamics area at Goddard (Solomon and Agresti, November 1987; and Section 5.7 of this report). However, reuse so far has been achieved without the benefit of systematic planning or automated library support. The Software Support Environment will include reuse plans and library support, but they are not yet available.

6.5 Management

NASA software management has been weak in two areas pertinent to this report: agency-level planning for the use of state-of-the-art software engineering technologies (e.g., Ada) and project-level management of software risks, including those associated with Ada.

6.5.1 Agency-Level Planning

NASA has not adopted an agency-wide plan for a transition to more effective software technologies including Ada.

Although additional Ada projects are planned, no comprehensive agency plan exists to evolve to Ada and associated technologies. Such a transition plan is presented in this report's companion document (McGarry et al., April 1989).

6.5.2 Risk Management

NASA software projects typically neglect the need for a written risk management plan.

Risk management plans provide systematic means to avoid unpleasant surprises that result in costly rework. In recognition of this fact, several DoD standards and directives require the formulation and implementation of a risk management plan. Such plans are also required by the policies of some NASA and DoD contractors, such as TRW (Boehm, April 1988, p. 61). By contrast, NASA's agency-wide policies and the managers of individual projects have given too little attention to this essential management tool. Software risk management was, however, addressed at the agency level in March 1989, with Release 4.3 of the Information Systems Life-Cycle and Documentation Standards.

6.6 Infrastructure

NASA does not have an adequate set of agency-level standards and internal organizations to provide direction in software engineering and to support the evolution to new software technologies such as Ada.

6.6.1 Policies and Standards

NASA's lack of agency-wide software development standards wastes resources.

As noted in Section 5.5.1, NASA's only agency-level products applicable to software development are NMI 2410.6, which requires a Software Management Plan for flight projects, and SMAP's *Information Systems Life-Cycle and Documentation Standards*. The SMAP products are nonbinding on agency organizations. There are no agency-level standards prescribing how to perform any of the following software development activities: management; assurance; configuration management; data base management; interface management; data communication; assurance of safety, security, and integrity; or verification and validation.

With the two exceptions just noted, the establishment of standards is left to the centers and individual projects. As argued in Section 4.5.1, this practice is time consuming and thus expensive for both NASA and its contractors. Many different groups within the agency must deal with the same issues, such as the revision of standards to accommodate the use of Ada. Similarly, contractors who deal with several parts of the agency often must learn how to work under multiple sets of standards and cannot economically develop automated tools to support document production. Moreover, center-level standards are no longer adequate in an era when some projects span multiple centers.

Finally, NASA's procedures for obtaining contractor input into the process of standards revision are not adequate to ensure that the agency's approaches to Ada and software engineering are coordinated with those of its major contractors.

6.6.2 Support Organizations

NASA does not have an adequate agency-level infrastructure to provide direction in software engineering and the evolution to new software technologies such as Ada.

No agency-level organization has general responsibility for NASA software engineering. In particular, no organization is responsible for software technology transfer within the agency. Nor is there any organization responsible for the collection and dissemination of data about NASA software development processes or for the assessment of their effectiveness. Consequently, these tasks are not being performed.

This lack of support organizations sometimes causes good policies to lose their effectiveness for lack of management followup. For example, NMI 2410.6 requires that the software management plan for flight project software undergo peer review before implementation. If lessons are learned from such a peer review, there is neither a requirement to write them down nor an organization responsible for propagating them to other software practitioners in the agency or for using the lessons to refine the review process. In fact, the peer review process has never been formally defined.

Developing agency-wide standards is a major undertaking requiring significant resources. Attempting to do so through ad hoc groups has both advantages and disadvantages. Ad hoc groups are very useful in that their members are committed to solving a problem and have an interest in seeing the effort succeed. However, they normally do not have the time, resources, authority, or visibility to promote their solutions at the agency level. The agency has no resources or internal organizations to perpetuate the good works and transfer the findings to the agency as a whole once the group dissolves.

A lack of agency-level resources also impedes the implementation of project-directed investigations and products. For example, the Softech Ada training plan (15 November 1987), although sponsored by the SSFP, is relevant to the entire agency. But because there has been no agency-level organization to sponsor and implement the plan, it has languished.

6.7 Research and Development

Although NASA has sponsored extensive research in software engineering, some of which has dealt with Ada-related technology, the agency has not had a broad, coordinated program of research, experimentation, and pilot projects.

Very little of the software R&D budget is directed toward issues concerning Ada and related technologies. Considering the number and size of NASA production projects that will use Ada within the next 5 to 10 years, the budget allocated to Ada issues is not adequate.

A number of the NASA centers have conducted small research efforts in Ada. Again, the amount of research is insufficient to support the major projects already planning to use Ada. In addition, no coordination of these research activities exists across the agency; no mechanism is in place to ensure that Ada technology developed through research will be transferred into practice. Therefore, NASA does not receive adequate benefit from the Ada research that is being conducted and has accumulated little evidence about the potential effects of Ada on software development by and for the agency.

6.8 Measurement and Assessment

NASA does not have an adequate program of collection and use of metric data about the software development process.

The SSORCE program at JPL and the SEL data base at Goddard are good examples of such data collection. However, most NASA projects do not gather software metrics. Consequently, such data is not available to serve as the basis for monitoring the development process and estimating the progress of a project. Nor is such data archived and effectively structured to serve as the basis for projections about future projects, for the development of risk models or cost-estimation models, or for research that might lead to evaluations and improvements of software development processes. Finally, there are no agency-level standards requiring the collection of software metrics or defining a standard set of metrics.

7 — Recommendations

This section contains the ASMAWG's recommendations for a transition to Ada and for improved software engineering in NASA.

7.1 Ada Adoption

NASA should adopt Ada as its standard programming language.

By adopting Ada as a standard, the agency can focus its efforts on this technology. This adoption should be approached gradually at each center. All new mission software should be developed in Ada by 1998. Since the Ada language was first defined in 1980, a period of 18 years to develop a full Ada capability in NASA would coincide with the approximately 18-year period that Riddle (April 1984) found was required for new software technologies to mature from initial conception to widespread use.

Each center should evolve to Ada through a three-phase approach in which the scope of Ada use is gradually widened. In the three phases, Ada should be used in the following classes of projects:

1. Laboratory and pilot projects intended to gain familiarity with the language and associated tools and methods
2. Selected production software projects
3. All new mission software projects

Our definition of *mission software* is the following:

> Mission software is all software that is critical to the design, planning, operation, control, or testing of any NASA flight project. It comprises all flight software and all ground software that directly interface with the flight systems or could affect mission planning, control, or operations. Mission software includes, for example, all software used in flight planning, flight dynamics, mission control, and flight readiness. It also includes all software used to simulate, model, or test any of the foregoing software functions.

A waiver process should be instituted to deal with situations where the use of Ada is inappropriate. For example, an entire application might be written in the data manipulation language associated with a commercial data base management system, or a new system might be created almost entirely from existing FORTRAN components. Waivers should not be required for acquiring commercial off-the-shelf software that is to be used without modifications, writing small special-purpose programs, or developing software where adequate Ada support is unavailable.

The goal of transition to Ada as NASA's standard programming language should be reassessed every 2 years on the basis of the data gathered in accordance with the recommendation of Section 7.10.

7.2 Software Engineering and Ada Implementation Task Force

NASA should establish a Software Engineering and Ada Implementation Task Force.

Evolution to Ada and state-of-the-art software engineering practices in NASA should be supported by a Software Engineering and Ada Implementation Task Force. The task force, composed of representatives from center support organizations, should provide information and consulting in software engineering and the use of Ada. A primary function would be to implement the 5-year plan for the transition to Ada (McGarry et al., April 1989).

The task force's duties should include the following tasks:

- Consulting with management on the use of Ada and the correct application of software engineering technology
- Consulting with software engineers on specific technology and application issues and providing background and reference information
- Gathering information on the use of Ada and advanced software engineering technology
- Gathering, analyzing, and disseminating metrics reflecting the effectiveness of Ada and various software engineering tools and methods
- Disseminating technical reports about Ada, software engineering, and the application of specific tools and methods

- Sponsoring and encouraging other NASA organizations to sponsor seminars, workshops, and conferences on Ada-related technologies
- Promoting software reuse
- Providing a NASA liaison with support groups from other government agencies and industry organizations to take advantage of their experiences and research
- Ensuring the development and implementation of the core NASA curriculum in Ada and software engineering

Such a formal support organization is necessary if Ada and improved software engineering practices are to be advanced in NASA. It will enable NASA to leverage limited funds by capitalizing on good practices inside and outside the agency. It will be a highly visible focal point where projects can look for assistance.

7.3 Policies and Standards

NASA should develop and adopt tailorable standards for software development, management, and assurance.

NASA should issue general agency policies and standards for software development, management, and assurance. Agency-level standards should form a framework within which more specific center or project standards can exist.

NASA should request contractor reviews of proposed standards. A mechanism analogous to the Council of Defense and Space Industry Associations should be identified for NASA.

Contractor experiences reported to the ASMAWG indicate that the literal and rigid enforcement of standards can delay a software project and increase its costs by requiring unnecessary actions, such as the production of inapplicable documents. Standards should be tailorable, if necessary, to fit a particular project. NASA project management should be responsive to a contractor's suggestions for tailoring.

7.4 Software Development Environments

NASA should evolve toward a common software support environment.

To evolve toward the common use of Ada without striving toward a common support environment would be inconsistent with the philosophy of Ada. NASA should not impose a single, fixed hardware and software architecture or a single set of precisely specified software support tools on contractors. However, NASA should define a common set of functional capabilities that would become the common environment for NASA software development. NASA should also define standard requirements for deliverables, e.g., language and medium. This policy would not preclude contractors from using their own specialized tools to assist the development process. NASA should also take responsibility for generating the standard set of functional capabilities on common support systems and making the resulting environment available for use on contracted software efforts.

7.5 Transition Planning

Each center should develop a plan for evolving to Ada.

The plan should establish milestones for reaching the goal of all new mission software being written in Ada by 1998. This plan should be submitted to the Software Engineering and Ada Implementation Task Force, which should be responsible for supporting the center in developing and carrying out these plans.

7.6 Training

The Software Engineering and Ada Implementation Task Force should ensure the development and implementation of an agency-wide core curriculum in software engineering and Ada. Each center should adapt the core curriculum to its specific needs.

The core curriculum should address Ada, software management, life cycles, quality assurance, configuration management, computer-aided software engineering, and other pertinent topics relevant to state-of-the-art software engineering. It should factor in the application to pilot or production projects as an integral part of the training process. It should include sequences of courses recommended for given classes of personnel, that is, a common, well-defined track for all managers and developers. Software managers should then require their personnel to take a sequence of courses suited to their backgrounds and responsibilities. Development of the core curriculum should take into account the educational planning that the Software Engineering Institute has done. The training program should be available to the NASA support contractors as well as NASA personnel.

NASA should also enhance its Ada knowledge base by using previous Ada training and experience as a criterion for evaluating potential employees.

7.7 Risk Management

For any "critical" project, management should be required to develop and implement a written risk management plan.

The plan should assess (identify, analyze, and rank) all development risks. It should propose a plan to control the risks through risk management planning, risk monitoring, and risk resolution (Boehm, April 1988). For Ada projects, the plan should cover risk items associated with the use of Ada (Basili et al., April 1987, sec. 3; Peschel, 13 December 1988). The criteria for "critical" projects should be defined in the software management standard referred to in Section 7.3.

7.8 Contractor Incentives

NASA should establish an Ada incentive program for its software contractors.

First, an offeror's experience with Ada and record of effective software management should be important considerations in the evaluation of proposals.

Second, NASA should recognize that at this early stage in the history of Ada, a contractor may have additional expenses connected with training and hiring Ada software engineers and acquiring a programming support environment. NASA should be willing to share these expenses, since they will affect the size of bids, whether or not they appear explicitly as line items in the cost proposal.

Third, the schedule of activities in a statement of work should be structured so as to give the contractor an incentive to hire or train adequate numbers of Ada personnel by the time they are needed. For example, a project might specify that a prototype be built or other pilot activity be performed at an early stage to give the Ada personnel some experience.

Fourth, a request for proposal or statement of work should be written so as to give the bidder or contractor incentives for software reuse.

7.9 Coordination of Research and Development

The Office of Aeronautics and Space Technology (Code R) should plan and coordinate agency-wide software research and development, more of which should support Ada.

NASA's approach to software-related research requires two significant changes in order to support the agency's adoption of state-of-the-art software engineering practices, including Ada.

First, research related to Ada must be significantly increased. Much of the work (under the Ada beta test bed) that was completed at JSC should have stimulated further NASA Ada studies aimed at learning more about the appropriate uses of Ada. However, the work that has continued is not extensive enough to support the timely evolution to Ada. The agency requires a substantial increase in Ada research, especially experimentation and laboratory studies.

Second, NASA must support a major, coordinated initiative in software engineering research, including Ada-specific research. The initiative that OAST has recently planned, NISE, should become the vehicle to coordinate all NASA-related research in Ada and software engineering. However, NISE must be enhanced to include significant research in Ada. The agency should continue to support Research Technology Objectives and Plans activities in other program offices (e.g., OSSA and OSO); but the relevant work carried out under these programs in the other program offices must be factored into the NISE program in OAST. OAST should not direct, plan, or control the research in other program offices; but it must become the focus for accumulating all relevant results so that the planning and execution of the initiative under the aegis of Code R can be more effective.

7.10 Software Measurement Program

NASA should establish an agency-wide program to collect and use software metrics.

As discussed in Section 4.7, metrics provide a sound basis for decisions about the management of a specific project (e.g., whether a given development phase is on schedule or complete) and about the general effectiveness of given tools or methods. The Software Engineering and Ada Implementation Task Force should specify metrics to be collected by software development projects. Project management should use the metrics for management purposes and make the information available to the task force. NASA researchers can then use these metrics to measure the effectiveness of software engineering practices and to suggest improvements.

The adoption of Ada as the agency's standard language would provide an opportunity to establish a systematic method for collecting and analyzing metric data on a very broad scale. Such data has a wide range of applications, such as improving the development process by monitoring how it is affected by new methods, tools, and techniques. Management has many potential uses of such data. Tools such as risk models and cost-estimation models could be developed. Managers could monitor the development process and estimate the progress of a project on the basis of information generated automatically by the software support environment.

7.11 Software Process Engineering Task Force

NASA should establish a Software Process Engineering Task Force to support the evaluation and improvement of the agency's software acquisition and in-house development processes.

The task force should adopt a method for the internal assessment of NASA's software engineering capabilities using an approach similar to that of the SEI (Humphrey et al., September 1987). This method is based on five postulated levels of process maturity: initial, repeatable, defined, managed, and optimized. It employs a set of questions and investigatory techniques through which an assessment team can determine the maturity level of a given software development organization. The Software Process Engineering Task Force should use such an assessment method to evaluate and improve of NASA's in-house software development capabilities. In addition, the task force should develop an analogous method to evaluate software acquisition processes and should apply the method to acquisition processes in NASA.

Appendix — Persons Consulted

The ASMAWG consulted the following persons in preparing this report:

David Barakat, TRW

Bryce Bardin, Hughes Aircraft Company

Victor Basili, University of Maryland

Frank Belz, TRW

Barry Boehm, TRW

Doris Boyd, Hughes Aircraft Corporation

John Bryant, TRW

George Buchanon, Hughes Aircraft Corporation

Marvin Carr, McDonnell Douglas

Robert Dausch, McDonnell Douglas

Raymond Delaney, General Electric Corporation

Robert Demshki, TRW

Michael Deutsch, Hughes Aircraft Company

Bud Doyle, TRW

John Garman, NASA/JSC

Dana Hall, NASA/HQ

William Halley, McDonnell Douglas

Hal Hart, TRW

James Inscoe, General Electric Corporation

Rhys John, General Electric Corporation

Richard Knackstedt, McDonnell Douglas

Milda Napjus, Hughes Aircraft Company

Jeffrey Neufeld, General Electric Corporation

Rose Pajerski, Goddard Space Flight Center

Albert Peschel, TRW

George Petrovay, Hughes Aircraft Company

Daniel Roy, Ford Aerospace Corporation

Walker Royce, TRW

Edwin Seidewitz, Goddard Space Flight Center

Dwight Shank, Computer Sciences Corporation

David Smith, Hughes Aircraft Company

Phyllis Stevens, General Electric Corporation

Christopher Thompson, Hughes Aircraft Company

Raymond Wolverton, Hughes Aircraft Company

Marvin Zelkowitz, University of Maryland

References

Aeronautical Radio, Inc., Airlines Electronic Engineering Committee (October 1987), "Guidance for Using the Ada Programming Language in Avionic Systems," Project Paper 613. Kansas City.

Basili, V., et al. (April 1987), *Use of Ada for FAA's Advanced Automation System (AAS)*, MTR-87W77. McLean, VA: MITRE Corp.

Boehm, B. (August 1986), "A Spiral Model of Software Development and Enhancement," *Proceedings of the IEEE Second Software Process Workshop, ACM Software Engineering Notes.*

Boehm, B. (September 1987), "Improving Software Productivity," *IEEE Computer,* Vol. 20, No. 9, pp. 43-57.

Boehm, B. (April 1988), *Software Risk Management Tutorial.* Redondo Beach, CA: TRW.

Booch, G. (1987a), *Software Components With Ada.* Menlo Park, CA: Benjamin/Cummings.

Booch, G. (1987b), *Software Engineering With Ada.* Second Edition. Menlo Park, CA: Benjamin/Cummings.

Brooks, F., et al. (September 1987), *Report of the Defense Science Board on Military Software.* Washington, DC: Office of the Under Secretary of Defense for Acquisition.

Century Computing (15 September 1987), *Ada Projects at NASA: Runtime Environment Issues.* Laurel, MD.

Dept. of Defense (February 1980), *Requirements for the Programming Environment for the Common High Order Language, Stoneman.* Washington, DC.

Dept. of Defense (1983), *Reference Manual for the Ada Programming Language,* ANSI/MIL-STD 1815A. Washington, DC.

Dept. of Defense (2 April 1987), *Computer Programming Language Policy,* DoD Directive 3405.1. Washington, DC.

Fisher, D. (March 1978) "DoD's Common Programming Language Effort," *IEEE Computer,* Vol. 11, No. 3, pp. 24-33.

Foreman, J., and J. Goodenough (May 1987), *Ada Adoption Handbook: A Program Manager's Guide*, CMU/SEI-87-TR-9, ESD-TR-87-110. Pittsburgh, PA: Software Engineering Institute.

Gardner, M., et al. (February 1988), *Software Engineering, Ada Development, and Acquisition Streamlining Under DOD-STD-2167*, MTR-88W00006. McLean, VA: The MITRE Corp.

Hinners, N. (27 June 1988), "NASA IRM Council; Ada and Software Management Assessment Working Group," memorandum to NASA IRM Council.

Humphrey, W., et al. (September 1987), *A Method for Assessing the Software Engineering Capability of Contractors* (Preliminary), CMU/SEI-87-TR-23, ESD-TR-87-186. Pittsburgh, PA: Software Engineering Institute.

Kneale, D. (12 January 1988), "Into the Void," *The Wall Street Journal*, p. 1.

Kolkhorst, B. (September 1988), "Shuttle Code Achieves Very Low Error Rate," *IEEE Software*, Vol. 5 , No. 5, pp. 93-95.

McGarry, F., et al. (December 1988), "Evolving Impacts of Ada on a Production Software Environment," briefing to Thirteenth Annual Software Engineering Workshop. Greenbelt, MD: Goddard Space Flight Center.

McGarry, F., et al. (April 1989), *NASA--Evolving to Ada: Five-Year Plan*. Greenbelt, MD: Goddard Space Flight Center.

Molko, P., and R. Loesh (1 December 1988), "Real-Time Weather Processor (RWP) Project: Ada Experience at PDR" (Unpublished briefing). Pasadena, CA: Jet Propulsion Laboratory.

Murphy, B., and M. Stark (October 1985), *Ada Training Evaluation and Recommendations From the Gamma Ray Observatory Ada Development Team*, SEL-85- 002. Greenbelt, MD: Goddard Space Flight Center.

Nelson, R. (1988), "Space Station Software: Progress and Plans," in J. Johnson (ed.), *Proceedings of the Fifth Washington Ada Symposium*. New York: Association for Computing Machinery, pp. 117-122.

Nielsen, K., and K. Shumate (1988), *Designing Large Real-Time Systems with Ada*. New York: McGraw-Hill.

Peschel, A. (13 December 1988), "Ada Risk Management and TRW Project Experience" (Unpublished briefing). Redondo Beach, CA: TRW.

Reifer, D. (December 1987), "Ada's Impact: A Quantitative Assessment," *Proceedings of the 1987 ACM SIGAda International Conference*. New York: ACM.

Riddle, W. (April 1984), "The Magic Number Eighteen Plus or Minus Three: A Study of Software Technology Maturation," *ACM SIGSoft Software Engineering Notes*, Vol. 9, No. 2, pp. 21-37.

Sammet, J. (August 1986), "Why Ada Is Not just Another Programming Language," *Communications of the ACM*, Vol. 29, No. 8, pp. 722-732.

Seidewitz, E., and M. Stark (July-August 1987), "Towards a General Object-Oriented Software Development Methodology," *Ada Letters*, Vol. 7, No. 4, pp. 54-67.

Softech (15 November 1987), *A Report on NASA Software Engineering and Ada Training Requirements*. Houston, TX.

Solomon, D., and W. Agresti (November 1987), *Profile of Software Reuse in the Flight Dynamics Environment* (Preliminary), CSC/TM-87/6062. Silver Spring, MD: Computer Sciences Corporation.

TRW Defense Systems Group (11 April 1985), *Language Task 3 Trade Study*. Redondo Beach, CA.

Voigt, S. (ed.) (1985), *Space Station Software Recommendations*, NASA Conference Publication 2394. Hampton, VA: NASA Langley Research Center.

NMI	NASA Management Instruction
OAST	Office of Aeronautics and Space Technology
OSO	Office of Space Operations
OSSA	Office of Space Science and Application
R&D	research and development
SEI	Software Engineering Institute
SEL	Software Engineering Laboratory
SLOC	source lines of code
SMAP	Software Management and Assurance Program
SMM	Solar Maximum Mission
SSE	Software Support Environment
SSFP	Space Station Freedom Program
SSORCE	Systems, Software, and Operations Research Center
TDRSS	Tracking and Data Relay Satellite System

Glossary of Acronyms

ACM	Association for Computing Machinery
ADP	Automatic Data Processing
AIM	Automated Information Management
AJPO	Ada Joint Program Office
ALC	assembly language code
ANSI	American National Standards Institute
APSE	Ada Programming Support Environment
ARC	Ames Research Center
ASMAWG	Ada and Software Management Assessment Working Group
CMU	Carnegie Mellon University
COBE	Cosmic Background Explorer
DEC	Digital Equipment Corporation
DID	data item description
DoD	Department of Defense
GRO	Gamma Ray Observatory
GSFC	Goddard Space Flight Center
HST	Hubble Space Telescope
IEEE	Institute of Electrical and Electronic Engineers
IRM	Information Resources Management (Council)
JPL	Jet Propulsion Laboratory
JSC	Johnson Space Center
KSLOC	thousand source lines of code
LaRC	Langley Research Center
LeRC	Lewis Research Center
MIL-STD	Military Standard
MSFC	George C. Marshall Space Flight Center
NASA	National Aeronautics and Space Administration
NATO	North Atlantic Treaty Organization
NISE	NASA Initiative on Software Engineering