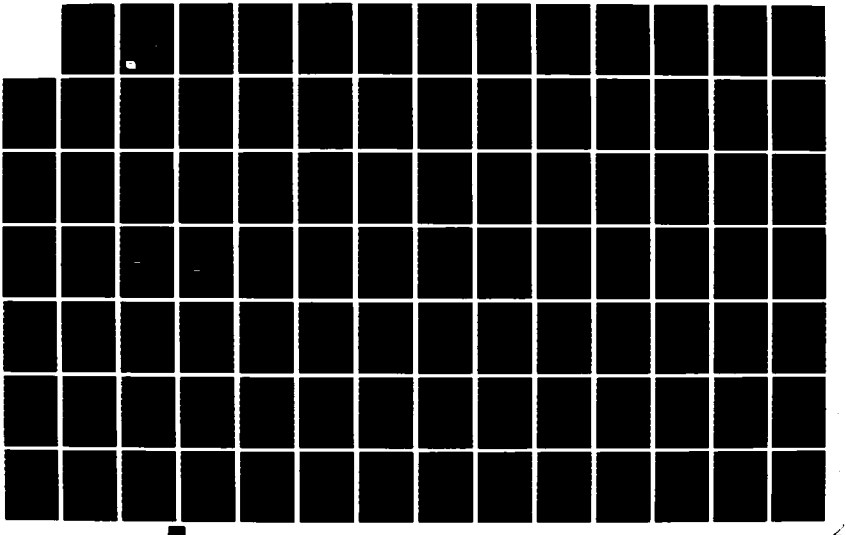
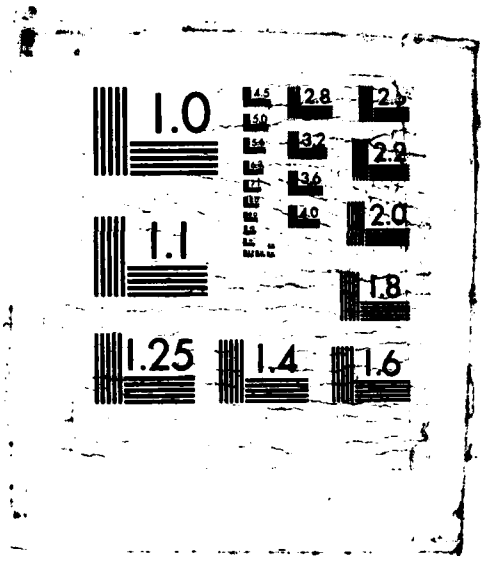
AD-A175 352 COST EFFECTIVENESS TRADEOFFS IN COMPUTER STANDARDIZATION AND TECHNOLOGY I..(U) INSTITUTE FOR DEFENSE ANALYSES ALEXANDRIA VA A A HOOK ET AL. JUN 86 1/4

UNCLASSIFIED IDA-P-1931 IDA/HQ-86-31052 MDA903-84-C-0031 F/G 9/2 NL




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0

(2)

IDA PAPER P-1931

COST EFFECTIVENESS TRADEOFFS IN COMPUTER STANDARDIZATION AND TECHNOLOGY INSERTION

Audrey A. Hook
Terry Mayfield
Thomas Frazier
Alan K. Graham
David Kreutzer

June 1986

Prepared for
Office of the Under Secretary of Defense for Research and Development
(OUSDRE)

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

IDA Log No. HQ 86-31052
Series B

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

ADA 175 352

1a REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b DECLASSIFICATION/DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S) P-1931	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION Institute for Defense Analyses 6b OFFICE SYMBOL IDA	7a NAME OF MONITORING ORGANIZATION
6c ADDRESS (City, State, and Zip Code) 1801 N. Beauregard Street Alexandria, VA 22311	7b ADDRESS (City, State, and Zip Code)
8a NAME OF FUNDING/SPONSORING ORGANIZATION Ada Joint Program Office 8b OFFICE SYMBOL (if applicable) AJPO	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER MDA 903 84 C 0031
8c ADDRESS (City, State, and Zip Code) The Pentagon, Room 3E139 (1211 Fern St., C107) Washington, DC 20301	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. T-D5-215 / WORK UNIT ACCESSION NO.
11 TITLE (Include Security Classification) Cost Effectiveness Tradeoffs in Computer Standardization and Technology Insertion	
12 PERSONAL AUTHOR(S) Audrey A. Hook, Terry Mayfield, Thomas Frazier, Alan K. Graham, David Kreutzer	
13a TYPE OF REPORT 13b TIME COVERED FROM _____ TO _____ 14 DATE OF REPORT (Year, Month, Day) 1986 June 30 15 PAGE COUNT 332	
16 SUPPLEMENTARY NOTATION	

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Ada, technology insertion, modeling, cost effectiveness, APSE, MCCR acquisition, decision support tools, software engineering environment, standards

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

This document reports on the feasibility of developing a decision support tool that could aid decision makers in formulating policies for the use of software standards and strategies for technology insertion. During the first phase (1983-1984) a "white paper" was completed which provided a conceptual framework for examining the role of standards in the MCCR acquisition process. During the second phase (1985-1986), this conceptual framework was translated into a prototype decision support tool. This tool provided the ability to simulate the effect of selected standardization policies on related technology and Mission Critical Computer resources (MCCR) costs, thus demonstrating the feasibility of modeling the linkages among DoD, standards policies, industry, technology, amd MCCR costs.

The first scenario simulated the role of the current policy for the use of Ada as the only higher order language for the development of MCCR software. The other two scenarios examined cost effective strategies for inserting Ada and its related software engineering environments into the MCCR software production and maintenance process. The findings from these simulations indicate that certain strategies/policies concerning the utilization of Ada have a powerful influence on MCCR software costs.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION Unclassified
22a NAME OF RESPONSIBLE INDIVIDUAL	22b TELEPHONE (Include Area Code) / 22c OFFICE SYMBOL

DD FORM 1473, 84 MAR

83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

IDA PAPER P-1931

COST EFFECTIVENESS TRADEOFFS IN COMPUTER STANDARDIZATION AND TECHNOLOGY INSERTION

Audrey A. Hook
Terry Mayfield
Thomas Frazier
Alan K. Graham
David Kreutzer

June 1986

INSTITUTE FOR DEFENSE ANALYSES

Contract MDA 903 84 C 0031
Task T-D5-215

Acknowledgments

During this project a number of people contributed ideas and effort that were very beneficial to the outcome of the project. Special thanks to Ms. Janet M. Gould (MIT) for editing the appendices and to Mr. Michael Saylor (MIT) for editorial work on the finished report. Others who deserve credit for helping during the construction of the prototype model were Dr. Thomas Probert, Col. Ken Nidiffer, Mr. Burt Newlin, Dr. Edward Lieblein, Dr. Jack Kramer, and Dr. John Salasin.

TABLE OF CONTENTS

LIST OF FIGURE

COST EFFECTIVENESS TRADEOFFS IN COMPUTER STANDARDIZATION AND TECHNOLOGY INSERTION

1.0 INTRODUCTION

This paper fulfills requirements for IDA Task T-4-215. This task was initiated in November, 1983 by the Director, Computer Software Systems, Deputy Under Secretary (Research and Advanced Technology) to be completed in several phases, each phase being dependant upon the availability of funds. The purpose of the task was to determine the feasibility of developing a decision support tool that could aid decision makers in formulating policies for the use of software standards and strategies for technology insertion. During the first phase (1983-1984) , a "white paper" was completed which provided a conceptual framework for examining the role of standards in the MCCR acquisition process. During the second phase (1985-1986), this conceptual framework was translated into a prototype decision support tool. This tool provided the ability to simulate the effect of selected standardization policies on related technology and Mission Critical Computer Resources (MCCR) costs, thus demonstrating the feasibility of modeling the linkages among DoD, standards policies, industry, technology, and MCCR costs. Section 4.0 of this paper provides an overview of the model while Appendices A and B provide detailed documentation.

Section 5.0 describes the results of scenario simulations. The first scenario simulated the role of the current policy for the use of Ada[1] as the only higher order language for development of MCCR software. The other two scenarios examined cost effective strategies for inserting Ada and its related software engineering environments into the MCCR software production and maintenance process. The findings from these simulations indicate that certain strategies/policies concerning the utilization of Ada have a powerful influence on MCCR software costs.

[1]Ada is a registered trademark of the U.S. Government, Ada Joint Program Office.

2.0 SCOPE

Translation of the conceptual framework presented in the "white paper" required a modeling technique that can make use of aggregate or anecdotal data to simulate the interaction of multiple variables over an extended period of time. The lack of detailed data which could be used to derive mathematical relationships among these variables was a constraint to be minimized by proper tool selection. We selected the system dynamics modeling technique as implemented by a commercially available software package.[2] The resulting simulations were based upon a model that was constructed in an iterative fashion as information was obtained from interviews and a literature search.[3] The scenarios were constructed to provide plausible answers to the Sponsor's questions concerning the cost-effectiveness of the Ada language standard and the need for additional standards for Ada Programming Support Environments (APSE's). The Sponsor's questions were:

- What will be the long term effect of the current Ada policy on MCCR software costs?
- Should the Government develop a standard for an APSE? If so, what should be standardized - operating system, tool sets, interfaces?
- Is there a strategy/policy that will accelerate insertion of Ada technology in the MCCR software acquisition process? If so, what will be the effect on MCCR costs?

The intended users of the results of these simulations are the Sponsor and those concerned with the use of standards in the Ada Program. Although the software is easy to learn and to use, we assume that operators of the model will primarily be analysts who support the process of policy formulation. The model enhances, rather than replaces, expert judgement about MCCR software by quantifying many of the considerations surrounding standardization issues.

Collection of data from the primary sources (i.e., DoD software developers/maintainers) and calibration of the model was outside the scope of the task.

---

[2] STELLA V1.1: High Performance Systems Inc., Lynn, NH.

[3] See Appendix F for listing of references.

3.0 BACKGROUND

DoD decision makers must have both a clear understanding of the general effects of standards in the MCCR acquisition process and the ability to analyze the potential impact of a particular policy for their use. This impact could be on overall MCCR costs or on the technology that is available for use in the development and maintenance of MCCR software. The general problem, facing DoD decision makers, is how to formulate standardization policies that are goal oriented and cost-effective for each of the life cycle phases of multiple MCCR programs.

The Ada Language was mandated for use in new MCCR projects by the DeLauer memorandum (DeLauer 1983). The life-cycle cost of these projects is expected to be lower than for those MCCR projects that have used older languages such as JOVIAL and CMS2, but the visibility of these lower costs could be obscured by the larger number of non-Ada projects that must be maintained for 20-30 years at higher costs per project. This underlying limitation on benefits from the Ada policy is the result of decades of non-Ada programming that is in the pipeline for development and maintenance. Even with the introduction of Ada for new projects, continuing development and enhancements of non-Ada applications will make savings from Ada smaller and more distant in time.

Some previous language standardization studies have projected modest savings (Clapp 1977) (Jensen 1984), when compared to the projected total cost for DoD MCCR software (George 1985). Other studies of Ada (Cormier and Alberts 1985) (Foreman 1985a and Foreman 1985b) (Hook and Fischer 1986) indicate that savings per project can be significant, with full accounting for the impact of Ada and its related technology on DoD MCCR software expenditure. However, this projected impact depends upon how quickly and how widely Ada and its related technology is used for the development and maintenance of MCCR projects.

DoD Sponsorship of additional standards has been considered as a possible risk reduction strategy that may accelerate the use of Ada. These potential DoD Sponsored standards include the Common Ada Interface Set (CAIS), an Ada Programming Support Environment (APSE) developed and furnished by the government, e.g., Army Language System (ALS), Ada Compiler System (ACS), Ada Language System (ALS-N).

Even with the DeLauer memorandum that set the policy for the use of Ada in DoD, there are questions raised by proponents of other languages about how acceptable Ada will be in meeting percieved MCCR requirements for real-time, secure, and artificial intelligence applications. There are, different views about what additional standards are needed to increase the rate at which Ada will be effectively used for MCCR applications. There are also different perceptions about how DoD should present Ada-related standards. Some believe that the government should develop and implement a prototype standard product for industry to use and improve through use on MCCR projects. Others believe that this approach would limit industry innovation to minor process improvements by freezing technological development prematurely to the standard.

If a DoD sponsored standard for a particular technology has been standardized at a level below the market place technology, DoD project managers may require waivers from that standard (or portions of it) so as to use a product they perceive as more capable, lower cost technology available to their contractor. Further, incompatibility among implementations will almost certainly result from the insertion of technology on a contractor by contractor basis. Eventually, incentives may disappear for DoD project manager's and contractor's to use a standard that does not reflect the current state of a technology. It is recognized that the maturity of the technology and market place incentives play a major role in

standardization decision. Finally, if additional standards are selected, should the policy for their use effect only the development life cycle or should it effect the maintenance life cycle phase which represents the larger portion (60-70%) of the MCCR expenditure.

Bearing in mind these complexities, we set out to develop a model that would allow decision makers interested in Ada and its related technology to test the impact of various assumptions on the total MCCR expenditure over a thirty year period. In essence, the prime focus of this study is to answer two questions:

- How does standardization affect innovation?
- How can standards best be implemented to obtain both effectiveness and economy?

4.0 APPROACH

The first step in development of the decision support tool was formation of a cross-disciplinary team who developed the approach. The team members provided expertise in economics, computer science, Ada related technology, and simulation techniques. *Ad hoc* team members were used, as needed, to provide insight on DoD software development practices, standardization policies, and industry perceptions of DoD acquisition practices. The team determined that its efforts should be focused on analyzing and understanding the issues and on developing a logical model that could be implemented and refined quickly.

4.1 Modeling Method

The problem addressed in this effort can be characterized as determining the variation of output variables (e.g., cost of DoD MCCR software) over time, based on the dynamic interaction of multiple input and process variables. Such problems can best be expressed using a continuous system simulation approach, particularly when many of the variables are most easily estimated in terms of rates of change.

The system dynamics method was selected as an appropriate technique for defining the linkages among the variables associated with the MCCR acquisition process, technology, standards policies, and cost. The team felt that the model should permit a user to examine the underlying structure, the assumptions, and numeric values used during previous simulations and to change any part of this structure in order to test a new set of assumptions. The use of rapid prototyping allowed continuous evaluation and refinement of the model as the simulation was analyzed and new information was collected. The software of the model provides documentation of the equations used for each variable while the documentation provided in the appendices includes the rationale and references as to why particular linkages were established with their associated functions and equations.[4]

4.2 The Model

The model has been organized as six sectors: LANGUAGE CHOICE; ADA INFRASTRUCTURE; ADA PROJECTS; NON-ADA PROJECTS; NON-ADA INFRASTRUCTURE; and COST. Figure 1 illustrates this organization. The principal variables in each of these sectors are described in the following sections.

4.3 Cost Sector

The cost sector of the model evaluates the yearly and accumulated costs of MCCR computer programming under whatever scenario is being simulated. The cost sector computes the costs on the basis of the programming workload represented by the number of projects, the computer language being used, and the amount of infrastructure (the summation of resources a programmer can draw upon when working in a particular higher order language) available. The cost sector allows for selection of constant dollars or discounted dollars to be calculated for any given scenario. The degree of discounting of future expenditures is a parameter whose value can be varied and experimented with.

Although the cost sector does not affect the scenarios, it is crucial in illustrating the impact of infrastructure on cost effectiveness. The model determines total costs by multiplying the number of projects by the cost per project. As infrastructure intensity and coverage

[4] Appendices A and B provide complete documentation of the model.

Organization of Prototype Model

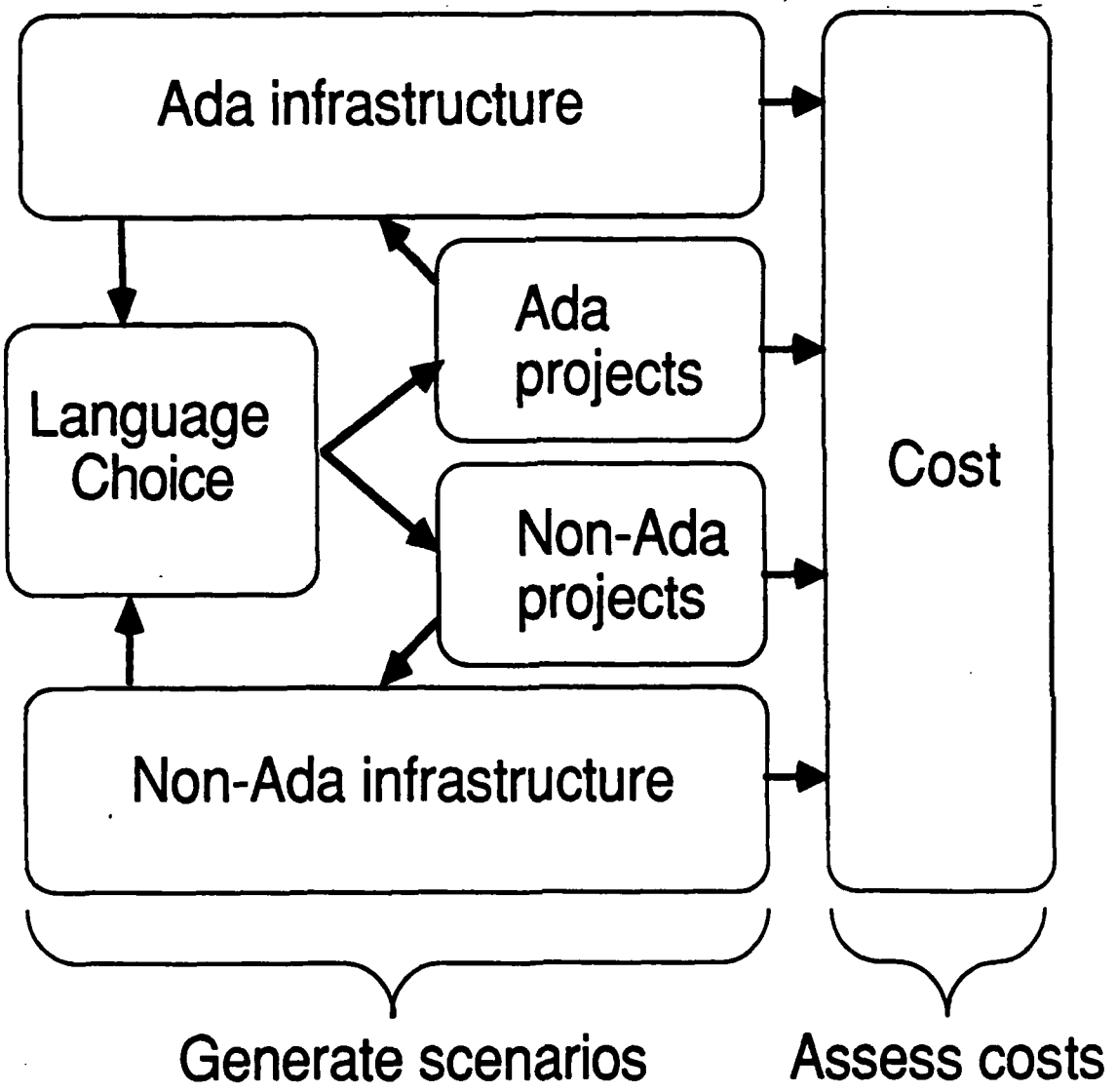


Figure 1

increase, the cost per project decreases.[5] Because the model sets the size of the standard project (amount of programming to be done) constant, this means that innovation is taking place. Not only is a given amount of work being done faster and cheaper, but for a constant expenditure, better (or more) work is being done.[6]

4.4 Infrastructure

Infrastructure denotes the programming support environment which includes not only the programming environment proper (operating system, tools, and program libraries) but also programmer experience and availability, programmer managers, and courses of instruction (in Ada and non-Ada languages). As the amount of infrastructure for a given language increases, so does the effectiveness of those projects making use of that language. This results in better performance as well as lower costs. It is thus crucial that the model represent the level of infrastructure available for both Ada and non-Ada programming separately.

An important input to infrastucture is the number of projects (either Ada or non-Ada) currently in progress (see Figure 1). Programmers gain valuable experience as they do more of their work in a certain language. In addition, there is an economic incentive for private companies to develop auxilary tools and environments. An increasing number of projects creates a market incentive, encouraging business to invest in a language with a secure future.

Because thorough comprehension of infrastructure is crucial in understanding how MCCR costs are affected by standardization policies, it has been disaggregated into three separate variables representing the characteristics of intensity, coverage, and incompatibility.[7]

4.4.1 Intensity

The intensity of infrastructure denotes how much infrastructure is available to a contractor or DoD programming manager. Intensity in the model is a selectable index number used to characterize whether or not good, efficient compilers are available, whether it is possible to hire (or train) programmers, the size and quality of the tool sets and the program libraries available to programmers, etc. Intensity is the primary determinate of programming cost. Higher intensity substantially raises programmer productivity.

Within the model, a low intensity index was selected to represent a "bare-bones" infrastructure which consists of a compiler and a link-loader, along with a few experienced systems analysts and managers. The highest intensity level of interest was selected to represent the conditions envisioned in the Stoneman specification (DoD 1980), assuming that the non-tool parts of the infrastructure (tool using, management experience, training, etc.) are equally well-developed.

---

[5] Consider the automobile analogy: the cost as well as the utility of the vehicle over its lifetime is a strong function of the availability of good roads, spare parts, qualified mechanics, and plentiful oil/gasoline.

[6] Appendix A.3 provides a complete description of the cost sector.

[7] See Appendices A.7 and A.8 for details

4.4.2 Coverage

Coverage of infrastructure denotes the fraction of host/target combinations that are available for use with a particular programming language and its related tools. Lack of choice in tools limits creativity, driving up costs and crippling innovation in the programming process. We assume that the critical term of measurement is the fraction of target machines for which a compiler is available because the availability of a particular language on a host is not a significant issue for most MCCR projects.

4.4.3 Incompatibility

Incompatibility of infrastructure denotes the extent to which infrastructure is divided into parts that do not intermix with one another: languages, operating systems, tools, and even managment procedures all fragment the programming meta-environments. Compiler languages usually underly the deepest divisions for programs, program libraries, programmers, and programming managers. We assumed that incompatibility will exist to some extent whenever a language is widely used even though there are policies and mechanisms for controlling the variety of tools and compatibility of tool sets. However, the rate at which incompatibility is introduced into the Ada infrastructure is assumed to be lower because it possesses effective mechanisms for controlling incompatibility. One such effective mechanism is the validation process for Ada compilers which is more rigorous and comprehensive than for other high order languages. Incompatibility is central to the analysis of standards policies, for incompatibility inhibits the accumulation of infrastructure. With markets fragmented among languages and operating systems, for example, neither programmer nor advanced tools develop rapidly.

4.5 Projects

Projects are the model's fundamental measure of programming work that must be done for MCCR systems. In reality, projects vary in size and cost. A very large programming task such as the World-Wide Military Command and Control System (WWMCCS) consists of many project units which are aggregated for budget and management purposes into groups that vary widely in complexity, lines of code, and man-hours. The definition of what constitutes a project unit involved some arbitrary choices in order to create a definition that would remain consistent over simulations of decades of changes in language, programming technology, and mission requirements. We divided projects into five categories and assigned consistent values to their cost-per-year and to the number of years per life cycle phase.[8] The categories of projects are described in the following sections.

4.5.1 Development Projects Using the Ada Language

These are the new development projects started that use the Ada language. Expert judgement was used to set the average duration of the entire development phase at 10 years. Some development projects will be completed sooner than 10 years, and some later.[9] When the infrastructure available for both non-Ada and Ada projects is equal, yearly Ada

---

[8] Appendix A.9 in the cost sector calibration section explains the procedure.

[9] See Appendix A.4, equation #540 for details and references.

development costs were defined to be $5-million . Costs for non-Ada projects were set at $6-million for the reasons described below.

4.5.2 Non-Ada Development Projects

We assumed that some portion of new starts will not use Ada and will initiate a waiver process because of the project manager's perception of risk and lack of available infrastructure for Ada. This non-Ada portion of new starts will be influenced by the free market forces in the Language Choice Sector including the level of both Ada and non-Ada infrastructure. However, since there is a DoD mandate to use Ada for new starts, the level of Ada Infrastructure at the time the project starts will be a stronger incentive for DoD project managers to use it than for a non-DoD project manager. The cost-per-year selected for non-Ada development projects was $6-million at the intensity of today's (primarily non-Ada) infrastructure. Non-Ada development projects have the same average time to complete development (10-years) as that of Ada projects. The slightly higher cost-per-year for non-Ada development projects is based upon our assumption that the expected benefits from Ada software engineering practices justifies a lower cost. We selected equal development time periods for Ada and non-Ada to remove a bias-for-Ada that would have resulted from using currently available Ada productivity studies based upon small projects.

4.5.3 Maintenance Projects

Ada and non-Ada maintenance projects are treated similarly in the model. Maintenance projects denotes all projects that have passed out of the development phase and into the maintenance phase. Any programming done after a project has left the development phase is accounted for in the model as part of maintenance programming. Although not consistent with all service usages, this definition is convenient for the purposes of the model. The meaning of the term maintenance differs among DoD constituents. In the Air Force terminology, large reprogramming projects (usually undertaken because of changed system requirements) are called redevelopment projects and are distinguished from the more routine fixing of bugs and making small additions to capabilities. In Navy terminology, as in the model, both redevelopment and bug-fixing are included in the term maintenance. The time period for a project's maintenance life cycle was estimated at 20 years. Maintenance costs per year for projects defined as above were estimated to be $2-million for Ada and $3-million for non-Ada, under conditions where the Ada programming environment is comparable to todays non-Ada environment. As with development, the cost difference is predicated upon Ada's superior support of software engineering methodologies.

4.5.4 Conversion Projects

These are projects that undergo major upgrades of functionality at planned intervals. It is possible to begin the maintenance life cycle of a project with a compiler and tool set for one language and to end the maintenance life cycle with a compiler and tool set of another language. During the 20-year period of maintenance, language choices can be made at intervals of planned project upgrade when there are significant incentives to do so. These incentives can arise from market forces and from DoD policies. In the model, we have provided a "policy lever" that can be activated for a policy that requires project managers to consider converting to Ada at major upgrade intervals.

The yearly cost for conversion projects includes both the funds for major redevelopment/ conversion as well as those necessary for routine maintenance of the unconverted portion of the project during this period. This has been approximated in the model to be equal to the Ada development project cost-per-year.

We assumed that if a project is undergoing a major redevelopment and conversion to Ada, it is doubtful that cost-conscious managers would also want to simultaneously carry on a major redevelopment in the non-Ada language. Therefore, once conversion work starts on a piece of software, it is reclassified as a conversion project immediately. When a project has completed the conversion period (considered to be two years in the model), it becomes an Ada maintenance project and is treated as such for the remainder of its lifetime.

4.6 Language Choice

We assumed that the world of non-DoD programming is fully functional and operating in parallel to the DoD programming project work. By making this assumption, we have created a sector of the model that operates on free market forces with respect to decisions about programming language and related technology. Implicitly, then programming decisions for DoD and non-DoD projects will be made from much the same inputs and incentives. Even with a mandated use of a particular language in DoD, the planned target for the number of projects using that language could be higher or lower than what actually occurs. With market forces operating to affect language choices, the actual number of projects could be lower than planned because of disincentives such as risk, as well as real or perceived lack of available infrastructure. On the other hand, if there is a perception that a particular language choice reduces risk/cost and will lead to a sustained business base, the number of projects using that particular language could be higher than planned. This is the link represented in Figure 1. The role of the Language Choice sector is to determine the acceptability of a given language standards policy to both DoD project managers and to the producers of software for DoD and other markets.

4.7 Model Construction

Construction of the model implementing the system dynamics symbology and mathematics began in August 1985. This model was refined and expanded through December 1985. The refinement process resulted in thirty-two software versions of the model which incorporated the assumptions and reference data that were collected by the team. The team simulated numerous scenarios to test the "reasonableness" of the model and discussed these simulations with several knowledgeable DoD project managers. From December 1985 through February 1986, detailed documentation was prepared so that the internal structure of the model would be visible for any future user.

Three policy scenarios were simulated by the team to demonstrate the feasibility of using an automated tool to support decision making. These simulations indicated plausable impacts on MCCR expenditure for each of the policy scenarios. However, the team recognized the need for collecting data from primary sources to calibrate the model and to re-run these simulations. A multiple target survey was developed and discussed with Service representatives as an important data source for model calibration.[10] Unfortunately, data collection activity was suspended in January because of a reduction in available funds for the second phase of this project.

[10] This survey is provided in Appendix D.

5.0 FINDINGS

The findings of the team are based upon their research, analysis of available data, and use of the prototype model to generate scenarios. Although the quantitative results of the simulations are open to question and substantial refinement, the qualitative results seem robust and important enough to the Sponsor that a detailed description is given here.

5.1 Baseline Scenario

This scenario simulates the effect of continuation of the current Ada policy. In 1981 there is a moderate initial injection of Ada Infrastructure due to the development of compilers by the government and private investment. The behavior which follows provides both a reference point as well as some insight into how the MCCR process develops in the coming years.[11]

5.1.1 Finding

Although the current Ada policy will eventually result in decreasing costs and greater effectiveness, these results will be painfully (if not needlessly) delayed. Figure 2 depicts this cost curve.

5.1.2 Discussion

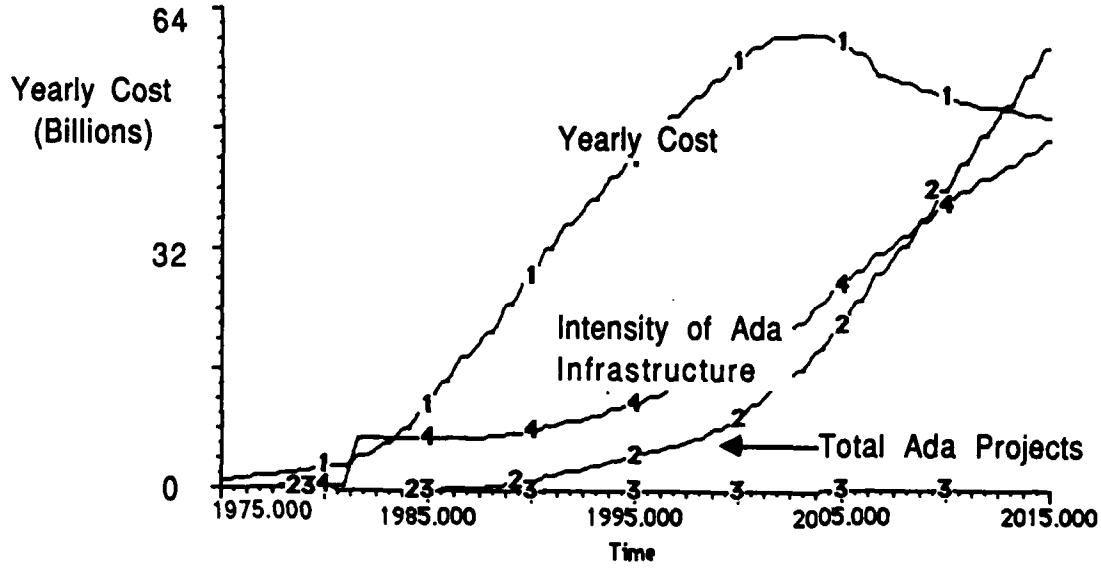
The increase in Ada infrastructure occurs when there are buyer's choices among validated compilers for host/target pairs that are useful for MCCR projects. This initial increase in Ada infrastructure provides increased incentives to use Ada. However, Ada-related savings are stalled by a large number of non-Ada projects already initiated which are progressing slowly through their thirty year lifecycle. Therefore, the inventory of Ada projects is small compared to non-Ada projects until 1996 when the cost-effectiveness of Ada has been demonstrated conclusively. As companies begin to independently develop more tools that add to the infrastructure, the intensity of Ada infrastructure increases slowly but steadily driving down the cost of Ada programming. The overwhelmingly important difference between Ada and non-Ada is that incompatibility is much more difficult to create among projects. Ada shows a correspondingly greater accumulation of intensity -- skills, tools, and reusable software -- that drives down Ada costs. Because of the much higher incompatibility in the non-Ada world, infrastructure cannot accumulate nearly as effectively, and intensity of non-Ada infrastructure never moves substantially beyond today's levels. At approximately 2003, the cost curve (1) indicates that the total yearly cost of programming peaks and begins to decline, despite increasing numbers of projects. At the end of this time period, Ada project costs have become 15 times less than those of Non-Ada projects. Thus, the majority of the cost after the year 2003 is coming from the maintainence of a minority of expensive non-Ada projects. Total MCCR expenditures for this scenario are 1.324 trillion dollars.[12]

---

[11] See Appendix B.3 for details and plots.

[12] By itself, this figure tells us little more than the expected order of magnitude of the of the actual expenditure. However, when used as a comparison point with which to judge the performance of scenarios where quite different assumptions have been made, or where different policies have been enacted, it can be very useful.

Baseline Scenario



Curve 1: Yearly Cost (in Billions) : (0-64)

Curve 2: Total Ada Projects (Projects): (0-24000)

Curve 3: Conversion Projects (Projects): (0-3200)

Curve 4: Intensity of Ada Infrastructure (Composite APSE and other resources): (0-200)

Figure 2

5.1.3 Preliminary Conclusions

Given the slow startup process for Ada use, along with the eventual superiority of Ada infrastucture, two general strategies appear to be particularly beneficial:

(1) Enactment of policies that accelerate the accumulation of Ada infrastructure (tools, program libraries, environments, etc.) promises to even further increase cost-effectiveness of MCCR acquisitions.

(2) Enactment of policies which encourage the conversion of the remaining mass of non-Ada programming to Ada likewise offers considerable potential savings by "piggybacking" off the benefits of increasing Ada Infrastructure with its attendant lower costs.

In the following two scenarios, both of these options are explored in a preliminary manner.

5.2 Commercial APSE Scenario

This scenario represents the case where the government selects one or more industry "standard" operating systems (VMS™/UNIX™[13], for example) for Ada programming, then proceeds to develop and implement Common APSE Interface Set (CAIS). The most important consequence of this policy is borrowing substantial infrastructure intensity from the non-Ada world, but leaving behind many of the incompatibilities caused by numerous other operating systems and tools used for non-Ada programming. The ability to create incompatibility in the Ada environment is permanently reduced by operating system standardization. Inclusion of transportable operating systems in a standard policy increases the speed with which Ada extends its coverage of hosts and targets.[14]

5.2.1 Finding

Ada-related standards policies increase the available infrastructure and reduce overall MCCR software expenditure when these standards are evolved from technology that is current in the market place. Figure 3 provides the curves for cost and infrastructure.

5.2.2 Discussion

A strategy that reduces uncertainty about DoD's preference for implementations of environments will both increase the number of Ada project starts as well as encourage companies to invest in Ada related tools. This is responsible for a positive trend where increasing infrastructure drives down costs.

---

[13] VMS™ is a trademark of Digital Equipment Corporation. UNIX™ is a trademark of AT&T.

[14] Appendix B.4 provides all plots, and tables, for this scenario.

This scenario assumed that these Ada-related environments could be adopted to the CAIS and the results would be widely understood within five years so that there would be an accelerated accumulation of Ada infrastructure within 6-7 years after completion of this initiative. By 1990 the yearly expenditure on MCCR costs is $1.9 billion less than the base scenario and the gap widens steadily. As a result, total costs associated with this scenario turn out to be $201 billion less than those for the baseline case. It is evident from these results that measures which serve to focus the development of Ada infrastucture have the potential to dramatically increase MCCR performance and decrease costs.

5.3 Conversion Scenario

The conversion scenario represents the enactment of government policies to encourage non-Ada projects already in progress to be converted to Ada. In the model, this policy is implemented in 1990 and by 1995 it is evident that a significant number of conversions have taken place.[15]

5.3.1 Finding

Conversion of non-Ada projects to Ada during their maintenance phase is instrumental in reducing total MCCR expenditures and increasing the availability of Ada infrastructure. (Figure 4 provides these curves.)

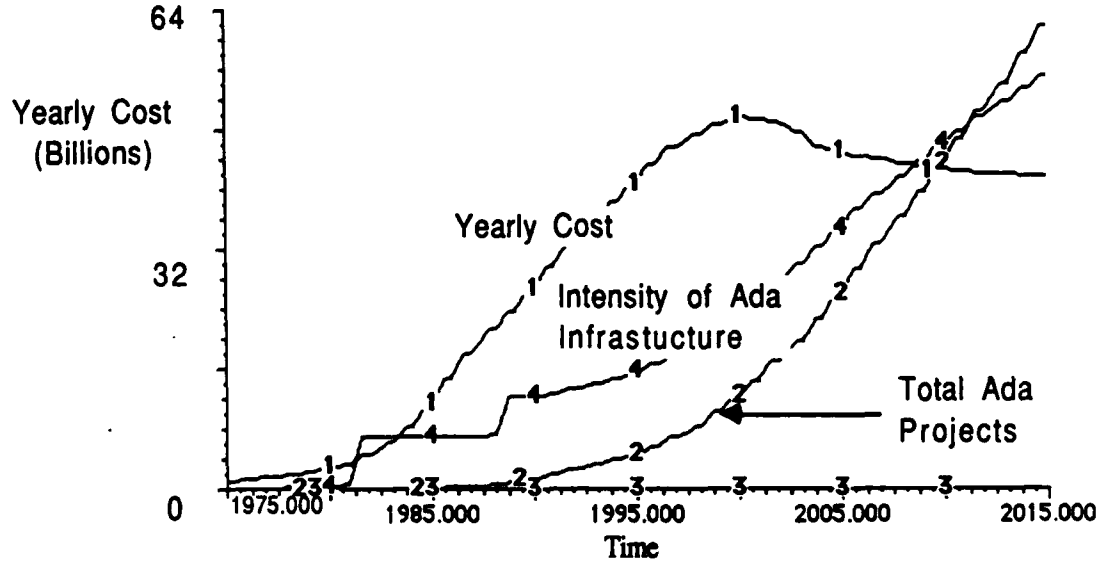
5.3.2 Discussion

Due to the greater proliferation of Ada projects, companies are encouraged to develop auxiliary tools, which results in a significant increase in infrastrucure. The savings in this scenario come from two sources: (1) the greater cost-effectiveness of Ada programming over the non-Ada infrastructure, due to the increase in infrastructure and (2) the added conversion of more expensive non-Ada projects to less expensive Ada projects. Total MCCR expenditures in this scenario are 1.179 trillion dollars. If we compare this figure to that of the base case, we note savings of $145 billion.[16] Thus, even with the additional costs of conversion, this policy results in savings equal to 12% of gross MCCR expenditure. Given that analysis is likely to continue showing high payoff for a general policy of conversions, more comprehensive investigation into specific policies and incentives to encourage non-Ada to Ada conversions is justified.

15. Appendix B.5 provides all output plots and tables for this scenario. Appendix B.6 is an overview of other policies that could be explored using the levers in the model.

16. With comparisons of this type, it is not the absolute numbers which are important, but rather the relative difference in performance between the two policies.

Commercial APSE Scenario



Curve 1: Yearly Cost (in Billions) : (0-64)

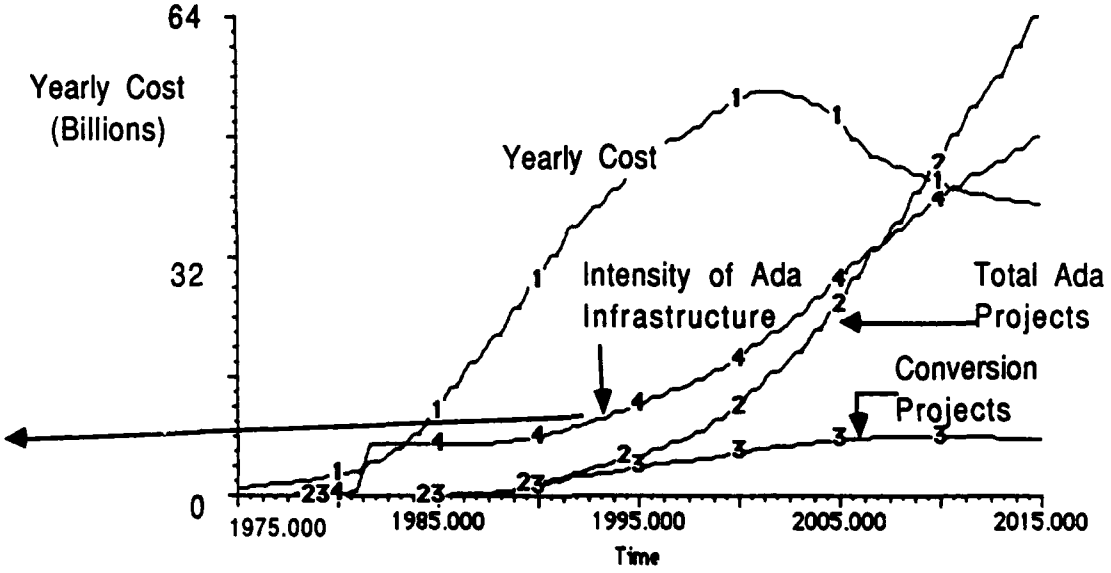
Curve 2: Total Ada Projects (Projects): (0-24000)

Curve 3: Conversion Projects (Projects): (0-3200)

Curve 4: Intensity of Ada Infrastructure (Composite APSE and other resources): (0-200)

Figure 3

Conversion Scenario



Curve 1: Yearly Cost (in Billions) : (0-64)

Curve 2: Total Ada Projects (Projects): (0-24000)

Curve 3: Conversion Projects (Projects): (0-3200)

Curve 4: Intensity of Ada Infrastructure (Composite APSE and other resources): (0-200)

Figure 4

5.4 Utility of the Prototype Model

5.4.1 Finding

The second phase of this study has resulted in the construction of a prototype model which demonstrates the feasibility of developing a decision support tool to assist policy makers in considering various standard related issues. This model not only offers potential solutions to current problems, but also clearly defines areas of study from which further insight into these issues may be gained.[17]

5.4.2 Discussion

The next step in this task involves calibration of the existing model. One value of the system dynamics methodology is that in the process of building the model, the researcher is directed toward the most significant avenues of investigation. Although it was not immediately apparent during the initial conceptualization of the problem, individual variables differ widely in their effect upon MCCR expenditures. For example, there is a substantial difference between the cost-per-year of non-Ada projects and Ada projects.[18] However, when these values were set equal to each other[19] in a run not presented here, it was discovered that the impact on the final results was negligible. In system dynamics terminology, the model is said to be *insensitive* to the cost-per-project, and in further calibration studies, this variable can probably be ignored.

Some variables then have a great deal of influence over MCCR expenditures, and they should be examined more closely. For two of these variables, the number of MCCR projects and the measure of the level of infrastructure, there are no authoritative sources of information. Two recent studies have been conducted that provide some aggregate information on MCCR software expenditures and projects. One study (EIA 1985), forecasted expenditures on MCCR software from 1985 to 1995. The other study (IDA 1985) reported investigation of MCCR programs that have a substantial software component but noted that the identification of software within MCCR is not explicit for DoD.

Projects are the model's fundamental measure of the programming work that exists to do for MCCR systems, for both development and maintenance. In the DoD environment projects vary enormously in size from one another. But it is convenient to think (and model) in terms of a standard project unit. A project cannot be defined in terms of some number of lines of code, because shifts between, for example, assembler language and higher-order language would change the amount of apparent programming work to be done. Similarly, a project should not be defined in terms of man-hours or dollar's worth of

---

17 Appendix C discusses potential revisions and investigation to improve the utility of this model.

18 Recall that development costs for non-Ada projects were set at $6-million per year, while maintenance costs for non-Ada projects were set at $3-million per year (assuming infrastructure is held constant at its normal value). These values for Ada projects were set at $5-million and $2-million, respectively.

19 Development costs were set at $6-million, maintenance costs at $2-million.

programming, because improvements in programmer productivity would likewise change the apparent complexity of the programming tasks currently being worked on. The project must be defined as an amount of programming to be done, measured with something akin to the inherent complexity or difficulty of the specification. Of course, the inherent difficulty can be translated into lines of code, man-hours per year, or dollars per year for a given language, programmer productivity, and programmer's wages. But the unit that is created by project starts, exists through the development phase, and endures during the maintenance phase is the less-measurable, but more fundamental and constant mission the software is to perform. The next phase of research should focus on more accurately measuring project units.

There is no source of collected information on the intensity and incompatibility of MCCR programming infrastructure, which we define broadly to include: programmer experience, number and quality of software tools and program libraries, educational programs and materials, and hardware support for programming. In the absence of such information, the model uses aggregated indices to characterize the infrastructure, which can be tied to specific information as it becomes available (see Section 4.0 for discussion and references). For example, consider the software tool component of the infrastructure. (Boehm 1981) defines five levels of tool intensity as it affects productivity:

- Very low: basic microprocessor tools
- Low: basic minicomputer tools
- Nominal: Strong minicomputer or maxicomputer tools
- High: Strong maxicomputer tools, Stoneman MAPSE
- Very high: Advanced maxicomputer tools, Stoneman APSE

Boehm shows econometrically how tools effect costs. Although the model's calibration does not yet explicitly use Boehm's results to show how higher intensity of infrastructure drives down costs, the model curves show the same qualitative effect. In addition, the model asserts that as infrastructure for both Ada and non-Ada programming rises, Ada costs will fall more rapidly, on the assumption that for any given level of infrastructure intensity, Ada's support of modern software engineering methods (which Boehm shows can reduce costs) will cause those methods to be more widely and effectively used.

Very low tool use corresponds to an intensity index of 20, and very high tool use corresponds to an intensity index of 100. The model calibration has not yet used this correspondence and Boehm's cost drivers to estimate the impact of intensity on cost. But such an estimation procedure may be premature, for it is not yet known what a typical intensity of tool use in MCCR programming is currently. (The calibration assumes an intensity index of 40, or roughly Boehm's "nominal" intensity, in 1985; Appendix A.9 gives the details.) Accurate knowledge of the current status must await the survey described in Section 4.7 and Appendix D.

Some amount of "field work," in the form of surveying and case studies, would fill a large gap in knowledge about MCCR projects and programming, and allow much more accurate calibration of the model.

6.0 CONCLUSIONS AND RECOMMENDATIONS

Although the results of this investigation are far from definitive, the weight of evidence suggests that, in general, standardization policies [20] have a payoff two to three orders of magnitude higher than the costs. MCCR software expenditures over the next 30 years can total around one trillion dollars. The policies examined offer savings in the tens or hundreds of billions, and the cost of implementing standards is in the tens of millions. Figure 5 provides a summary comparison of the three scenarios.

Admittedly, with incomplete information, one might be tempted to move cautiously on standards until more certainty can be obtained. However, there is a cost associated with waiting to get all the facts. In one simulation not presented here, delaying the implemention of the "standardize on commercial OSs, then CAIS scenario" for five years costs an extra $5 billion. This cost is by itself much larger than the expenses involved with creating and promoting programming environment standards.

Given the current weight of evidence indicating the effectiveness of standardizing the tool interface/operating system for Ada, and the very high payoff ratio, the most desirable course of action for DoD is to proceed aggressively on several fronts, both advancing the state of knowledge about standardization and continuing along the path of standardization:

Three Scenarios in Comparison

	Total Cost (Billions of dollars)	Fraction of Projects in Ada
Baseline Case	1324	78%
Commercial APSE Scenario	1123	81%
Conversion of non-Ada to Ada	1179	84%

Figure 5

[20] These may be similar to those described in the previous scenarios, or some conglomeration of various policies.

1. Continue toward more definitive research on the cost-effectiveness tradeoffs in standardization.

2. Continue to aggressively develop, prototype, and test CAIS as the eventual standard operating interface.

3. Proceed toward standardization on a selected small number of portable and non-portable operating systems, a) as an interim standard, b) as an aid to migrating programming to CAIS, and c) as a backup if CAIS is delayed or less than satisfactory.

4. Continue research on automatic translation from non-Ada languages to Ada and evolve policies and incentives that would spur conversation to Ada and migration to an APSE.

APPENDIX A

MODEL STRUCTURE

APPENDIX A

LIST OF FIGURES

Appendix A: Model Structure

Appendix A documents the structure of the equations that comprise the model discussed in this report.

Appendix A.1 reviews the system dynamics symbols and terminology used in the remainder of the appendices.

Appendix A.2 gives abbreviations used within variable names; it may be handy keeping track of variables and what they mean.

Appendix A.3 describes the cost sector of the model.

Appendix A.4 describes the Ada projects sector of the model.

Appendix A.5 describes the non-Ada projects sector of the model.

Appendix A.6 describes the language choice sector of the model.

Appendix A.7 describes the Ada infrastructure sector of the model.

Appendix A.8 describes the non-Ada infrastructure sector of the model.

Appendix A.9 describes the multiple-variable parameter estimation procedures used in calibrating the model. Simpler parameter estimations are described with the equation descriptions above.

Appendix A.1: System Dynamics Terminology and Symbols

The model described in this report uses standard system dynamics symbols and terminology to represent diverse types of variables,each having its own dynamic characteristics. For the convenience of the reader a brief description is provided here. For further discussion see (Forrester 1961, Part 2) or (Alfeld and Graham 1976, Ch. 1-2).

Levels

Level variables represent variables the state of the system at any single instant of time, i.e. the stocks, or accumulations. One rule of thumb determines whether a variable is a level variable through a thought experiment: If time were to suddenly stand still, and nothing change, only level variables would be measurable. The amount of water in a bathtub would be observable and measurable. The amount of water is a level. The rate of flow out of the drain is measurable only by observing changes over time. The flow of water out of the drain would not be a level variable, but a rate variable (see the discussion below).

Levels accumulate the changes to the system state caused by the rates of inflow and outflow. The concept is common to many disciplines. Mathematicians would think of levels as the results of *integrating* the inflow and outflow rates. Engineers would recognize levels as *state variables*. Economists would recognize them as *stock variables*, and accountants would recognize *balance sheet items* as levels.

The interconnections among variables in system dynamics models are usually shown graphically on flow diagrams, which differ somewhat from , e.g., FORTRAN flow diagrams in showing not sequential flow of control, but simultaneous flow of information. Figure A.1-1 shows an example of a system dynamics flow diagram. Levels are represented by rectangles. The figure shows a level, Total cost, accumulating the inflow rate, Cost per year discounted. (The meaning and function of this variable are discussed in detail in Appendix A.3) Total cost is measured in units of dollars while its inflow rate is measured in dollars/year. Inflow and outflows to levels are always measured in the units of the level over time units.

The equation below shows how STELLA represents the equation for the level, Total cost. All STELLA equations are printed with a small version of their flow diagram symbol at the beginning of the equation, so here, the equation for computing a level variable, Total cost, begins with a rectangle.

```
□ Total_cost = Total_cost + Cst_yr_dis
  INIT(Total_cost) = 0 {Total cost (dollars)}
```

The actual equation STELLA uses for calculation differs from that above if the simulation proceeds at time intervals other than 1.0 time units (1.0 years). For each time step of the simulation STELLA computes:

Total_cost (present time) = Total_cost (previous time step) +DT * Cst_yr_dis

where DT (delta or difference in time) is the duration of the time step in the simulation. The level at the present time in the simulation equals the value at the previous time step plus the net rate of change in effect over the time step, which equals the yearly rate of flow times the number of years in the time step.

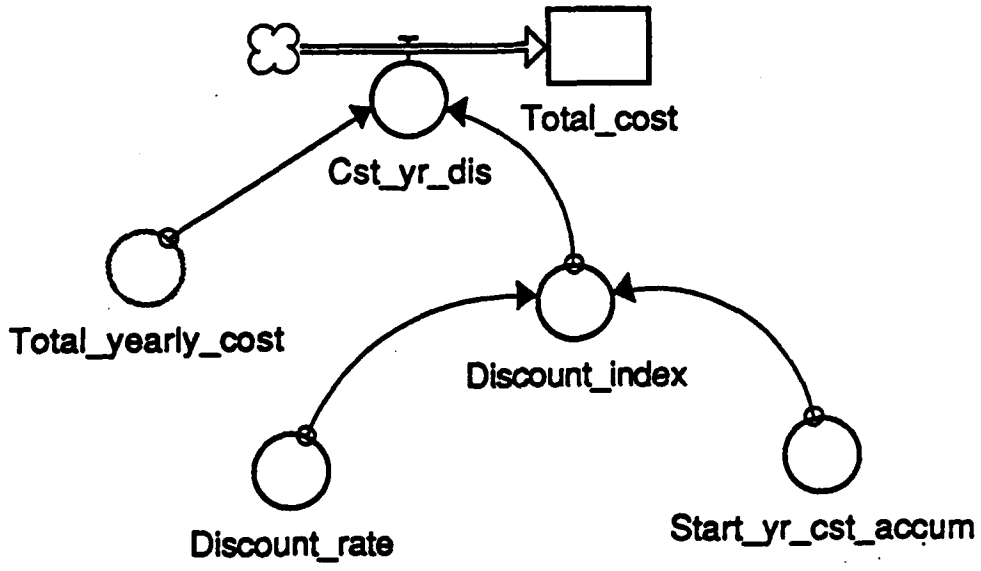


Figure A.1-1. Flow diagram example of inputs to Total cost (Total_cost).

Although STELLA calculates the equation correctly for any value of DT, STELLA shows the equation as if DT were 1.0 so it can be left out. By leaving out the explicit representation of DT, STELLA's designers have denoted the possibility of using mathematical integration methods—in effect, ways of proceeding from one time step to the next—more sophisticated than the simple arithmetic just described. (choosable under STELLA's "Specs" main menu item).

Rates

Rate equations define those variables that change the system levels. Therefore, the rates of change always flow into or out of a level and are always measured in the units of the level over the time units (years, for the model here). For example, in the figure, the Cost per year discounted is the flow measured in dollars per year that is accumulated in the level, Total cost, which is measured in dollars.

The STELLA symbol for a rate, as shown in the figure, is a circle with a symbolic "valve" on top controlling a flow arrow that goes between a symbolic "cloud" (that also resembles a four-leaf clover) and the rectangle which represents the level. If the rate is an inflow, the arrow points to the level and the cloud is called a source. If the rate is an outflow, the arrow points to the cloud which is then called a sink. Sinks or sources just mean that the flows come from or go to destinations outside the system boundary, i.e., places not influencing the system's behavior. For example, for a simple model of water in a bathtub, where the water goes after flowing through the drain would not be modelled by another level, but by a sink.

Converters

Converter variables are used directly or indirectly in a rate equation. Rates ultimately depend only on levels, so it would be possible write very long rate equations as functions only of the levels. This would be confusing, however, so instead, the rates equations are broken up into more meaningful sub-equations called converter equations. Converter variables are then given their own names and are represented with large circles. The small circles on the converter variables represent information takeoff points where the causal links go to other variables. These causal links then terminate with arrowheads pointing to the next variable in the causal chain.

Converter variables take information from levels or other converter variables, does some computation—"converts it"—and then sends that new information to a rate or another converter. All chains of converter variables eventually go into rate variables.

The figure shows several converter variables providing information for the rate Cost per year discounted (Cst_yr_dis). The converter variable called Discount index takes information provided by the converter variables called Discount rate (a percentage per year) and Starting year for cost accumulation (1986) and computes an index that is then used by the rate variable Cost per year discounted. This discount index wieghts or discounts future expenditures, representing the smaller importance of costs in the future as oposed to immediate costs. (For more detail on the meaning and usefulness of the equations just described, see Appendix A.3, "Cost Sector.")

"Ghosts"

As with standard FORTRAN flow diagrams, or any other diagram of a complex system, system dynamics flow diagrams of even medium sized models like the model being discussed here are too large to fit on a normal page. STELLA allows portions of the

flow diagram to be exhibited through the convention of "ghosts"—variables that are shown but aren't really there. (They're defined elsewhere on the flow diagram, usually on another page.) "Ghosted" variable are denoted on the flow diagram by variable symbols drawn with grey, rather than black lines. Total yearly cost (Total_yearly_cost) is shown on the sample as a ghosted input to Cost per year discounted (Cst_yr_dis). Total yearly cost is the raw flow of real (FY'86) dollars spent on MCCR programming; its computation is shown elsewhere.

Graphic functions

Converter variables often use graphic functions because they are a convenient way to specify a variable's output values for any given input values. If the modeller cannot invent a simple, plausible equation that will do the proper conversion, the graphic function can specify the relationship. This is particularly useful for non-linear relationships, where effect is not strictly proportional to cause.

Figure A.1-2, shows a graphic function from the model. The converter variable, Effect of intensity of Ada infrastructure on Ada cost, computes a value for the effect for any value of Intensity of Ada infrastructure. This represents the idea that as Ada infrastructure increases the cost of doing an Ada project goes down. When the user draws the curve, only eleven points need to be specified by drawing the curve accross ten intervals. STELLA uses linear interpolation to calculate the output value for inputs between the specified values.

DYNAMO versus STELLA terms in literature

For the last two and a half decades the DYNAMO family of simulation languages defined the standard in the field of system dynamics for computer simulations from mainframes to microcomputers. Newer software, called STELLA, used in this project breaks with the DYNAMO equation format by organizing the equations around icons. This iconic representation is possible because of the sophisticated graphics capability of the Apple Macintosh personal computer. Although STELLA is a software breakthrough in terms of the style of interaction with the model, the underlying concepts are identical to the traditional DYNAMO simulation languages for which there is a substantial body of literature. There are, however, differences in nomenclature between the STELLA User's Manual (Richmond 1985) and the rest of the DYNAMO-based system dynamics literature. For the benefit of those familar with that literature and nomenclature, the table below shows equivalencies between STELLA and DYNAMO. The asterisks denote the terminology chosen for use in this text.

STELLA Terminology	DYNAMO Terminology
Stock	Level *
Flow and Flow regulator	Rate of flow *
Converter variable *	Auxilliary variable
Input link	Causal link *
Signals	Information flow *

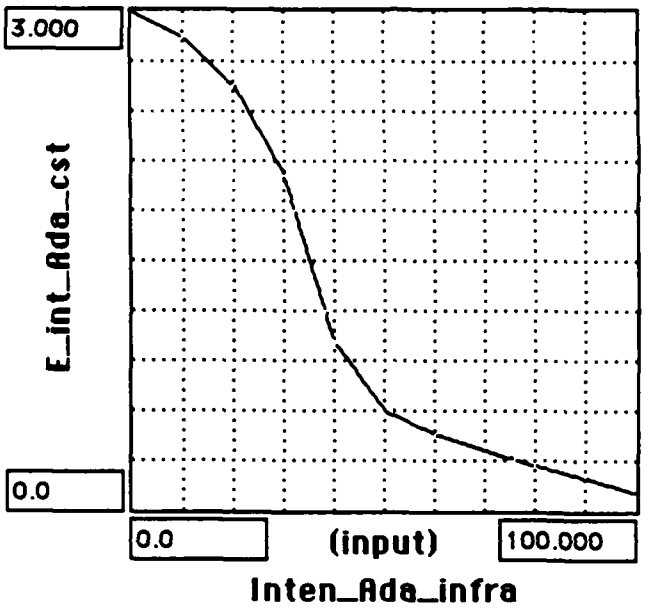


Figure A.1-2. Example of graphic function for a converter variable: Effect of intensity of Ada infrastructure on Ada cost (E_int_Ada_cst).

Appendix A.2: Standard Abbreviations within Variable Names

Variable name length is limited in STELLA, so the names are abbreviated. For example, "Ada maintenance projects" is abbreviated "Ada_maint_prj:" "maint" abbreviates "maintenance, and "prj" abbreviates "projects." Such abbreviations are standardized throughout the model; the following list gives the commonest. Occasionally, a word that appears in more than one variable will have two abbreviations. For example, when there is room, "maintenance is abbreviated "maint." When there is not room, the abbreviation is "mn."

Abbreviation	Stands for
A	Ada
ch	change in
compl	completions
conv	conversion
cov	coverage
crea or cr	creation
cst	cost
dev	development
dur, or du	duration
E	effect of or effects on
fr	fraction
incen, ince, inc	incentive
incom or inco	incompatibility
infra, infr, or inf	infrastructure
inten or int	intensity
maint or mn	maintenance
NA	non-Ada
nat	natural
norm	normal
obsol	obsolescence
perc	perceived
pol	policy
proj, or prj	project
ref	reference
rel	relative
t	time to or time for
targ	target
tot	total
yr	year

Appendix A.3: Cost Sector

Appendix A.3 on the cost sector equations is the first of 6 sections of Appendix A that describe the model equations. The text accompanying each equation describes what the variable or parameter means. If the estimation of a parameter value is straightfoward and relatively simple, the estimation will be described with the equation. For example, all *a priori* estimates are described with the respective equations. More complex estimations are described in Appendix A.9, "Multivariable Model Calibration."

The model is described as it existed in December, 1985. The formulations are still in the midst of review and revision. The model equation descriptions, for clarity, do not describe alternative formulations, questions about standard software practices, inconsistencies in available data, and so on. Such questions and observations are recorded in Appendix C, "Areas for Further Investigation."

The model equations are assigned unique numbers, and they are described in order. To facilitate reference to individual equations, below are listed the ranges for equation numbers in each of the equation description appendices. Also listed are the present numbers of equations in each sector.

Number range	Eqn. Count	Sector
1- 499	31	A.3 Cost sector
500- 819	19	A.4 Ada projects sector
820- 999	9	A.5 Non-Ada projects sector
1000- 1349	14	A.6 Language choice sector
1350- 1999	39	A.7 Ada infrastructure sector
2000- 2399	24	A.8 Non-Ada infrastructure sector
	136 total	

The cost sector of the model evaluates the yearly and accumulated costs of MCCR computer programming under whatever scenario is being simulated. The cost sector computes the costs on the basis of the programming workload represented by the number of projects, what computer language they are using, and how well-developed the infrastructure is. At present, the cost sector has no effect on the scenarios as they are simulated; its function is purely assessment. Of course, costs are a consideration at many points in the system that creates scenarios, but these are represented through different channels, as will be explained in the descriptions of the other model sectors.

Total cost (Equation #10)

Total cost represents the accumulated DoD expenditures on mission-critical software development and maintenance in discounted 1986 dollars. Total cost is one of the most important variables in assessing the outcome of alternative scenario simulations. Given that several policy questions hinge on tradeoffs between short-term expenditures and long-term benefits, it is necessary that one compare not just yearly expenditures on software between different scenarios, but compare some measure of expenditures that spans many years. Total cost is that measure.

Total cost (written Total_cost in the model) is a level variable; it accumulates the rate of flow into it, the Cost per year discounted. Figure A.3-1 shows the flow diagram of the inputs to Total cost; subsequent figures will show the computation of those inputs.

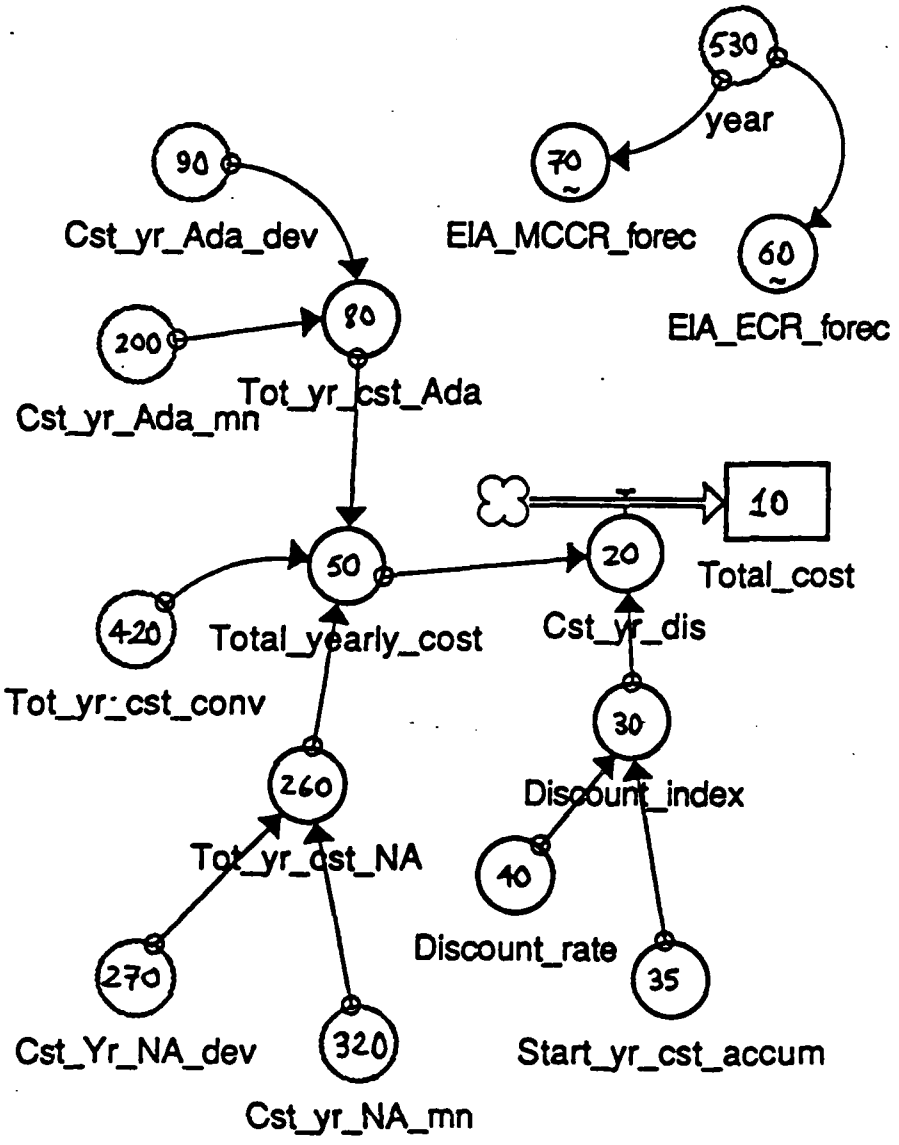


Figure A.3-1. Flow diagram of inputs to Total Cost (Total_cost)

□ Total_Cost = Total_Cost + Cst_yr_dis
INIT(Total_Cost) = 0 {Total cost (dollars)}

Total cost is initialized at zero, since by definition expenses begin to accumulate only at some point during the simulation. (The details of when the accumulation begins are discussed shortly.)

While economic inflation is an important consideration in setting budgets and other activities that call for predictions in current dollars, inflation is not central to evaluating standardization policies. Accordingly, Total cost accumulates costs as measured in 1986 constant dollars.

Resources on hand today are worth more than identical resources deliverable tomorrow. Consequently, dollars with which we can buy resources today are worth more than dollars available tomorrow. Thus, before we can meaningfully add together dollars spent or received in different periods we must "discount" future dollars, for they are worth less than current dollars. Accordingly, Total cost accumulates discounted expenditures, so that expenditures in the future add only a fraction of their current dollar value to the accumulated expenditures represented by Total cost. The degree of discounting of future expenditures is a parameter in the model whose value can be varied and experimented with.

Cost per year discounted (Equation #20)

Total cost accumulates yearly expenditure; Cost per year discounted is that expenditure. It represents the funds actually spent on software activities (not just budgeted), discounted by a factor causes a dollar spent earlier to add more to Total cost than a dollar spent later.

Cost per year discounted (abbreviated Cst_yr_dis in the model) is a converter variable, converting the Total yearly cost to a quantity discounted to weight earlier expenditures more heavily.

○ Cst_yr_dis = Total_yearly_cost*Discount_index
{Cost per year discounted (dollars/year)}

Because the discount index is zero any time before the Starting year for cost accumulation, Cost per year discounted will likewise be zero before that time.

Discount index (Equation #30)

Weighting the importance of present versus future expenditures can be accomplished by multiplying the expenditures by a factor. This should be 1.0 in 1986, representing full importance of present expenditures. The factor should decline slightly for each year thereafter, representing less weight on later expenditures; the later, the lesser. In financial jargon, this is known as discounting cash flows. (The term "discounting" came from a bank's practice of buying mortgages and other financial instruments at a discount from their face value, because the payment of that face value would be happening in the future; the "discount" between face value and purchase price represents the value of money later versus now.) Because the multiplicative factor is dimensionless and passes through 1.0, it is considered an index, the Discount index.

The Discount index (written Discount_index in the model) is a converter variable, converting the year into an index through an algebraic formula.

○ Discount_index = IF TIME ≤ Start_yr_cst_accum THEN 0 ELSE EXP(-Discount_rate*(TIME - Start_yr_cst_accum))

The formula implements in STELLA a standard exponentially-declining weight, starting with a value of 1.0 when TIME equals the Starting year for cost accumulation. The exponential is expressed within an IF...THEN clause that makes the Discount index zero before the starting year, and therefore shuts off accumulation of costs before then.

Starting year for cost accumulation (Equation #35)

Costs which have already been incurred at the time of an analysis are "sunk costs." According to both standard financial analysis and DoD Instruction 7041.3, sunk costs should not be included in the comparison of alternatives. To exclude sunk costs, the model measures accumulated costs of software activities, but only after the present, which is 1986. The present year is specified in the model as the Starting year for cost accumulation.

The Starting year for cost accumulation (abbreviated Start_yr_cst_accum in the model) is a converter variable with no inputs, i.e., a constant

○ Start_yr_cst_accum = 1986
{Starting year for cost accumulation (year)}

The Starting year for cost accumulation is written as a separate symbolic constant, rather than simply a number in various formulas, so that there is just one place in the model where the assumption about what the present time resides. It wouldn't be desirable to review and edit the whole model every time a new year passes.

Discount rate (Equation #40)

Any exponentially-declining Discount index can be characterized completely by a single number, that represents the percentage by which the discount index declines each year. This number is the Discount rate.

The Discount rate (written Discount_rate in the model) is a converter variable, but with no inputs; it is a constant, or equivalently, a parameter.

○ Discount_rate = 0 {Discount rate (fraction/year)}

The discount rate is set at zero, representing the assumption (at least until the value is changed) that all expenditures, from the starting year for the policy analysis onward, are equally important. Setting the Discount rate higher would give less weight to future expenditures. DoD Instruction 7041.3 ("Economic Analysis and Program Evaluation for Resource Management," October 18, 1972) mandates discounting, and suggests a value of 10 percent.

The discount rate is an important parameter in evaluating short-term/long-term tradeoffs in policies. If the discount rate is high, short-term expenditures are weighted very heavily in Total cost. If a policy to minimize short-term spending is being evaluated against policies that have a short-term cost and a long term gain, a higher discount rate will favor the short-term cost minimizing policy. If one believes in policies that invest for the future, one should therefore evaluate policies using a low discount rate, which gives weight to long-term issues as well. At least until better information about DoD decision-making is obtained, the long-term planning orientation is reflected in setting the Discount rate to zero.

Total yearly cost (Equation #50)

The Total yearly cost represents expenditures made per year on mission-critical software development and maintenance, expressed in 1986 dollars. Total yearly cost includes software activities both in Ada and non-Ada languages.

The Total yearly cost (written Total_yearly_cost in the model) is a converter variable, summing the Total yearly cost of Ada projects, the Total yearly cost of non-Ada projects, and the Total yearly cost of conversions (from non-Ada programs to Ada programs)

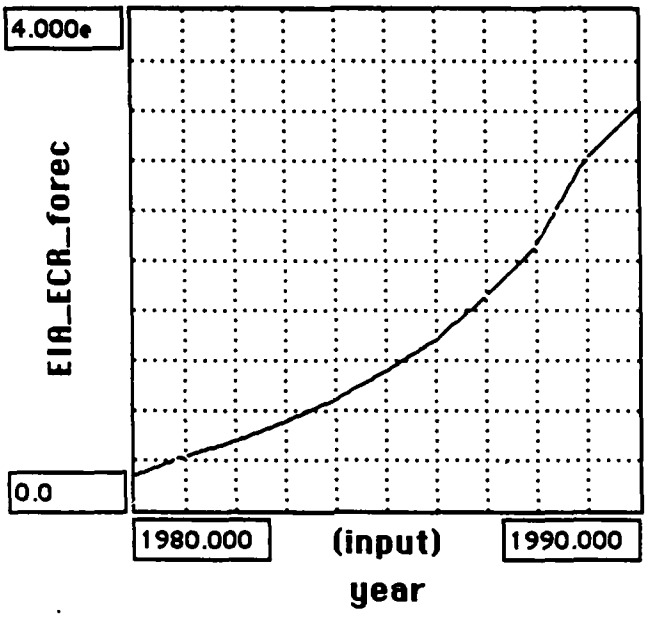
○ Total_yearly_cost = Tot_yr_cst_NA+Tot_yr_cst_conv+Tot_yr_cst_Ada {Total yearly cost (dollars/year)}

Even though the accumulation of yearly expenditures is set to zero prior to a starting year, that setting is accomplished through the Discount index; Total yearly cost is positive for the entirety of the simulation.

EIA 1980-1990 ECR forecast (Equation #60)

For convenience in comparing model output to real data, the model contains time series drawn from the Electronics Industry Association's forecasts (EIA 1980). The first forecast, made around 1980, is of yearly expenditures on embedded computer software from 1980 to 1990. The variable representing this forecast is the EIA embedded computer resources forecast.

The EIA embedded computer resources forecast (abbreviated EIA_ECR_forec in the model) is a converter variable, converting the year into the corresponding forecast value for that year.



Input	Output
1980.000	2.820e+9
1981.000	4.490e+9
1982.000	5.620e+9
1983.000	7.180e+9
1984.000	8.950e+9
1985.000	1.117e+10
1986.000	1.390e+10
1987.000	1.716e+10
1988.000	2.120e+10
1989.000	2.815e+10
1990.000	3.210e+10

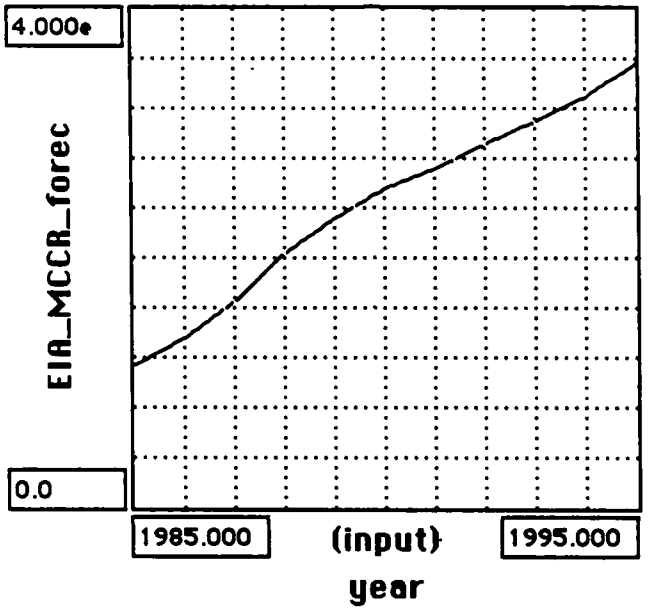
A software bug in the present version of the STELLA simulation program prevents standard display of the equation for the graphic function defining EIA_ECR_forec. There are more decimals, including the floating-point exponents than the modules that display equation listings can handle. The above table of values is drawn from a display by a different part of the software.

Due to limitations of the simulation software, the value of EIA_ECR_forec outside of its defined range of 1980-1990 is just the value of the endpoint: for time before 1980, the 1980 value, and for time after 1990, the 1990 value. The relevant period for comparision to model output, however, is the 1980-1990 range.

EIA 1985-1995 MCCR forecast (Equation #70)

The model contains another time series drawn from the Electronics Industry Association's forecasts. This forecast is of yearly expenditures in current dollars on mission-critical computer software from 1985 to 1995 (EIA 1985). The variable representing this forecast is the EIA mission-critical computer resources forecast.

The EIA mission-critical computer resources forecast (abbreviated EIA_MCCR_forec in the model) is a converter variable, converting the year into the corresponding forecast value for that year.



Input	Output
1985.000	1.141e+10
1986.000	1.354e+10
1987.000	1.662e+10
1988.000	2.038e+10
1989.000	2.311e+10
1990.000	2.559e+10
1991.000	2.729e+10
1992.000	2.911e+10
1993.000	3.104e+10
1994.000	3.297e+10
1995.000	3.566e+10

A software bug in the present version of the STELLA simulation program prevents standard display of the equation for the graphic function defining EIA_ECR_forec. There are more decimals, including the floating-point exponents than the modules that display equation listings can handle. The above table of values is drawn from a display by a different part of the software.

Due to limitations of the simulation software, the value of EIA_MCCR_forec outside of its defined range of 1985-1995 is just the value of the endpoint: for time before 1985, the 1985 value, and for time after 1995, the 1995 value. The relevant period for comparision to model output, however, is the 1985-1995 range.

Total yearly cost of Ada projects (Equation #80)

The total yearly cost of Ada projects represents the amount spent on Ada-based software projects each year, expressed in constant dollars

Total yearly cost of Ada projects (abbreviated Tot_yr_cst_Ada in the model) is a converter variable, which sums the Cost per year for Ada development projects and the Cost per year for Ada maintenance projects.

```
○ Tot_yr_cst_Ada = Cst_yr_Ada_mn+Cst_yr_Ada_dev
    {Total yearly cost of Ada projects (dollars/year)}
```

Cost per year for Ada development projects (Equation #90)

The cost to do computer programming is a fairly predictable function of the size and complexity of the task undertaken. A "project" as used in the model represents a unit of programming complexity, so that a very large programming task in real life would be represented in the model as several projects. (The definition of a "project" as a unit of programming work is detailed in Section 5 of this report.) The cost to do computer programming then becomes just a matter of how much programming work there is to do, measured in projects, and how much the standard project costs per year. The product of these two factors becomes the Cost per year for Ada development projects.

The Cost per year for Ada development projects (abbreviated Cst_yr_Ada_dev in the model) is a converter variable, converting the number of Ada development projects and the Cost per year for Ada development projects into a yearly flow for all Ada development projects. Figure A.3-2 shows the flow diagram for this computation.

```
○ Cst_yr_Ada_dev = Ada_dev_prj*Cst_prj_yr_Ada_dev
    {Cost per year for Ada development projects
     (dollars/year)}
```

Cost per project-year for Ada development projects (Equation #100)

(Boehm 1981) discusses in great detail how much a given piece of software work will cost, given various assumptions about the tools, people, and procedures involved. Moreover, there are reasonably good ways of predicting how much of the cost will be spent in a particular year. Such calculations are for individual projects, but the analogous concept exists for large collections of programming efforts. For the body of Ada development projects, that cost is the Cost per project-year for Ada development projects.

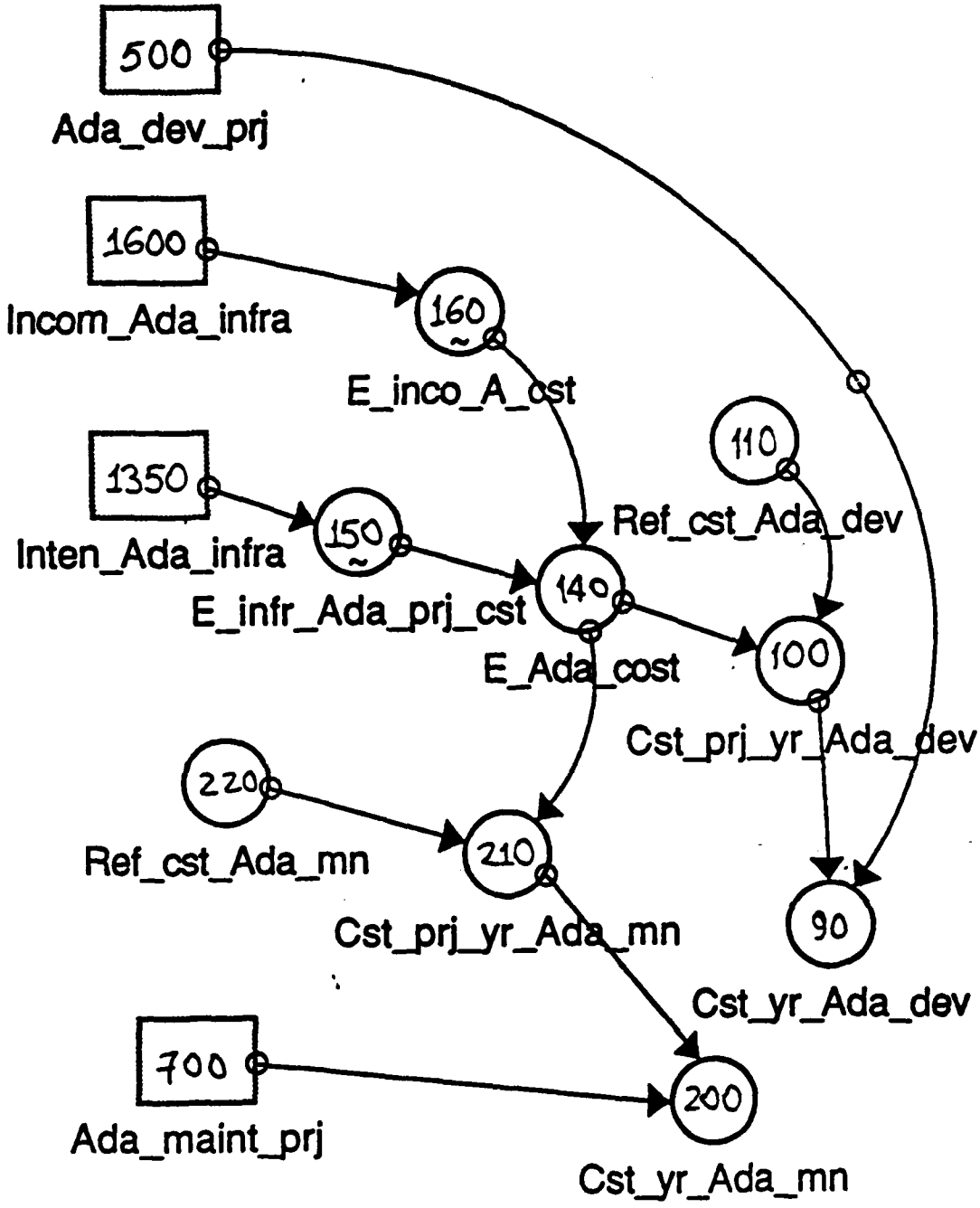


Figure A.3-2. Flow diagram of inputs to Cost per year for Ada development projects (Cst_yr_Ada_dev).

The Cost per project-year for Ada development projects (abbreviated Cst_prj_yr_Ada_dev in the model) is a converter variable.

```
○ Cst_prj_yr_Ada_dev = Ref_cst_Ada_dev*E_Ada_cost
    {Cost per project-year of Ada development projects
    (dollars/project/year)}
```

The cost equation above is written in a form common in system dynamics for expressing a single output as a function of multiple inputs. The Reference cost for Ada development projects defines what the standard unit of software work costs per year to do, under some set of conditions (tools available, experience of programmers, etc.). The Effects on Ada costs gives the effect on costs when the current conditions differ from the reference conditions. This equation form is discussed in (Alfeld and Graham 1976, Section 5.3) and (Richardson and Pugh 1981, pp. 152-56).

Two sets of equations may be defined with respect to to different sets of reference conditions, corresponding to two different sets of hypothetical conditions. This usually happens because the modeller doesn't have good information for all equations under one consistent set of conditions. When reading the model equations, therefore, it will be important to notice the reference conditions for each reference constant as a separate entity.

Reference cost for Ada development projects (Equation #110)

The reference cost for Ada development projects is the cornerstone for determining Ada development costs: it specifies the cost per year for a standard software project under standard conditions. The standard conditions are defined as the current average intensity and incompatibility of (primarily non-Ada) infrastructure. (Details of how the current characteristics of programming infrastructure are measured in the model were discussed in Section 5 of this report.)

The Reference cost for Ada development projects (abbreviated Ref_cst_Ada_dev in the model) is a converter variable with no inputs, i.e., a constant.

```
○ Ref_cst_Ada_dev = 5E6
      {Reference cost for Ada development projects
      (dollars/project/year)}
```

It should be reemphasized that all of the costs in the model deal with yearly expenditures, which are quite different from life cycle costs. For example, it may be that projects have a life cycle maintenance cost higher than the development cost. But the maintenance expenditures will usually be spread out over more years than the development expenditures. In that case, the reference cost (measured in dollars per year) for development may well be higher than the reference cost for maintenance. The spending per year is presumably more intense.

The numerical value for the Reference cost for Ada development projects is central to the policy evaluation process, yet at the current time, highly uncertain. The issue of how the values are selected will be treated in Appendix A.9.

Effects on Ada costs (Equation #140)

A number of variables influence how much a given amount of programming work costs. The composite of such influences is represented by the Effects on Ada costs.

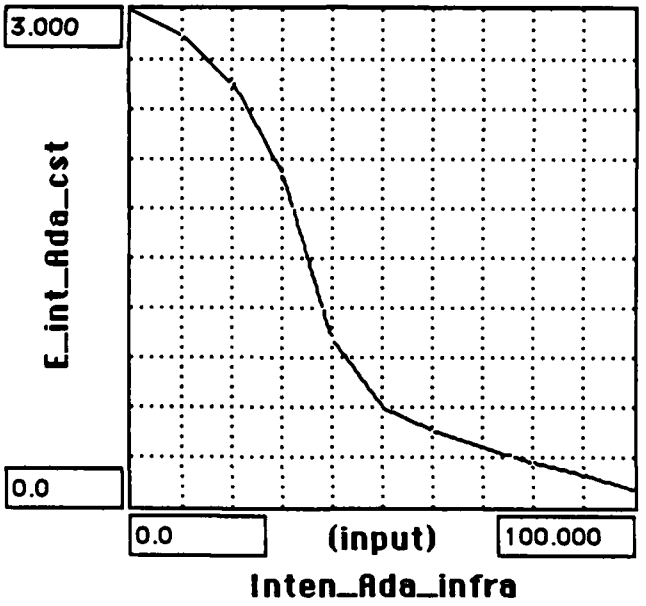
The Effects on Ada costs (abbreviated E_Ada_cost in the model) is a converter variable, that converts specific influences (intensity and incompatibility of infrastructure) into a single aggregate effect.

○ E_Ada_cost = E_infr_Ada_prj_cst*E_inco_A_cst
{Effects on Ada costs (dimensionless)}

Effect of intensity of infrastructure on Ada project costs (Equation #150)

The intensity of infrastructure brought to bear on programming projects has a strong influence on how much the programming costs. More experienced programmers, ample computer facilities for programming, reusable code, software tools, and so on all cause the programming to go faster and more reliably. The aggregate representation of these is the Effect of intensity of infrastructure on Ada project costs.

The Effect of intensity of infrastructure on Ada project costs (abbreviated E_int_Ada_cst in the model) is a converter variable, using a graph function to convert the intensity of Ada infrastructure into an effect on cost.



E_int_Ada_cst = graph(Inten_Ada_infra)
0.0 -> 3.000
10.000 -> 2.850
20.000 -> 2.565
30.000 -> 2.010
40.000 -> 1.000
50.000 -> 0.600
60.000 -> 0.465
70.000 -> 0.360
80.000 -> 0.270
90.000 -> 0.200
100.000 -> 0.100

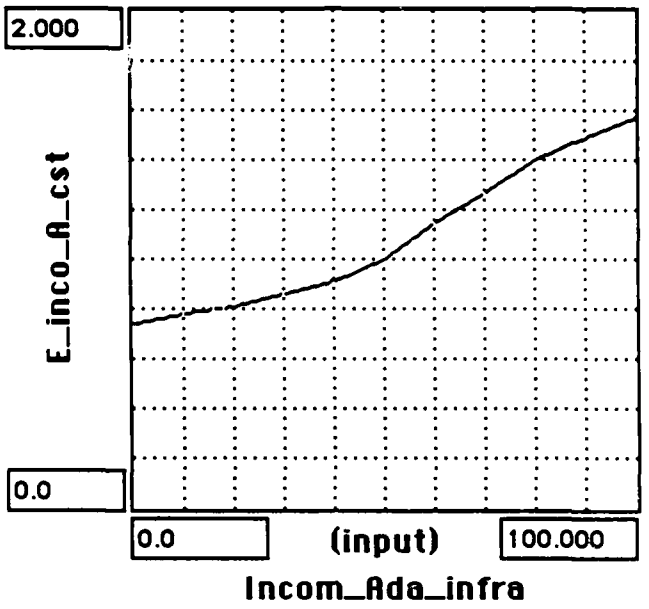
Generally, the graph function causes cost to decrease as infrastructure becomes more intense. The curve is normalized to a reference condition equivalent to the current intensity of non-Ada infrastructure. At that point, the effect is neutral, i.e., it has a value of 1.0. The calibration of this curve is discussed in Appendix A.9.

Effect of incompatibility of infrastructure on Ada project costs (Equation #160)

Incompatibility of infrastructure has both indirect and direct effects on costs. The indirect effects occur when incompatibility inhibits the accumulation of intensity of infrastructure, which can keep costs higher than they otherwise would be. Most of the commonly-discussed ways by which incompatibility increases cost are in this indirect channel. But there is also a direct effect, where incompatibility of infrastructure effects the ability to integrate and maintain large systems, where programming may have been done by several contractors. Systems integration becomes more difficult if programs must be shuttled between different operating systems or recompiled. Assembling a project team is also more difficult in the face of incompatible infrastructure. Incompatibility creates situations where experienced people exist, but are tied up on other projects, and the people who are available may not be experienced with the language, operating system, or tools needed. And hiring someone experienced in language A on operating system B with tools C, D, E, and F is difficult because of the numerous constraints. In essence, incompatibility adds to the costs of the integrating contractor. This is discussed in (Foreman 1985a, Foreman 1985b, and IDA 1985a)

Examples abound of incompatibility raising programming costs in the non-Ada world, but Ada is not exempt. Incompatibility of operating systems means that tools will not be transportable, so that even if major subsystems are developed with extensive use of tools, those may no longer be available during system integration. Even on the same operating system, compiling programs with a different Ada compiler may change program behavior–the Ada specification is not totally complete with respect to how features interact. So use of generics within generics, or tasking within generics may; be handled differently in different validated Ada compilers. The overall consequence of such incompatibilities is higher costs. This effect is represented by the Effect of incompatibility of infrastructure on Ada project costs.

The Effect of incompatibility of infrastructure on Ada project costs (abbreviated E_inco_A_cst in the model) is a converter variable, using a graphic function to convert the index of incompatibility into an effect on cost. Higher incompatibility adds to cost, and conversely.



E_inco_A_cst = graph(Incom_Ada_infra)
0.0 -> 0.740
10.000 -> 0.780
20.000 -> 0.810
30.000 -> 0.860
40.000 -> 0.910
50.000 -> 1.000
60.000 -> 1.150
70.000 -> 1.270
80.000 -> 1.400
90.000 -> 1.490
100.000 -> 1.570

The curve is normalized for a reference condition of today's level of incompatibility of non-Ada infrastructure. At that point, the effect is neutral, i.e., it equals 1.0. The curve has a three-to-one difference in yearly cost between zero incompatibility and one hundred incompatibility (roughly twice the 1985 incompatibility). That is a three-to-one difference between a situation where everyone has experience in exactly the same language, operating system, and tools, and beyond that, the same management style, procedures, formalisms, and so on, and a situation where effectively no one has any of these elements in common with any of the other team members. This is a curve where (IDA 1986) may provide the basis for a modestly rigorous estimate.

Cost per year for Ada maintenance projects (Equation #200)

As with programming done for product development, there is a cost for programming done for maintenance. The Cost per year for Ada maintenance projects measures those costs.

The Cost per year for Ada maintenance projects (abbreviated Cst_yr_Ada_mn in the model) is a converter variable, converting the number of Ada maintenance projects and the Cost per project-year for Ada maintenance projects to a total dollar flow for Ada maintenance programming.

○ Cst_yr_Ada_mn = Ada_maint_prj*Cst_prj_yr_Ada_mn {Cost per year for Ada maintenace projects (dollars/year)}

Cost per project-year for Ada maintenance projects (Equation #210)

Any individual MCCR system will have an erratic curve of maintenance programming expenditures versus time. One year, very little may be done with it, apart from fixing identified bugs. Other years may see a major redevelopment effort due to changed requirements. The model assumes that, however variable and unpredictable the expenditures for individual projects may be, the aggregate of many such projects tends to produce a relatively stable expenditure stream, which is therefore predictable by the law of large numbers. The Cost per project-year for Ada maintenance projects is in effect that predicted cost.

The Cost per project-year for Ada maintenance projects (abbreviated Cst_prj_yr_Ada_mn in the model) is a converter variable, converting what the cost would be under a set of reference conditions (the Reference cost for Ada maintenance projects) and the effect on cost of departing from those reference conditions (the Effects on Ada cost) into an expected cost per standard Ada maintenance project.

○ Cst_prj_yr_Ada_mn = Ref_cst_Ada_mn*E_Ada_cost
{Cost per project-year for Ada maintenance projects (dollars/year)}

Reference cost for Ada maintenance projects (Equation #220)

The cornerstone for evaluating costs is the idea that a given amount of programming work, with a given infrastructure, will cost a predictable amount in the average year to maintain. That "prediction" is the Reference cost for Ada maintenance projects

The Reference cost for Ada maintenance projects (abbreviated Ref_cst_Ada_mn in the model) is a converter variable with no input, i.e., a parameter.

○ Ref_cst_Ada_mn = 2e6 {Reference cost for Ada maintenance projects (Dollars/project/year)}

Appendix A.9 describes the derivation of the value of this reference cost.

Total yearly cost of non-Ada projects (Equation #260)

The structure of equations that lead up to Total yearly cost of non-Ada projects is precisely analogous to the structure for Total yearly cost of Ada projects. The parameter values are also exactly the same, except where explicitly noted and derived in the model calibration section, Appendix A.9. Therefore, no verbal description will accompany the equations defining Total yearly cost of non-Ada projects.

○ Tot_yr_cst_NA = Cst_yr_NA_mn+Cst_Yr_NA_dev
{Total yearly cost of Nonada projects (dollars/year)}

Cost per year for non-Ada development projects (Equation #270)

Figure A.3-3 shows a flow diagram of the computation of non-Ada costs; the diagram is symmetrical with the corresponding diagram for Ada costs.

○ Cst_Yr_NA_dev = NonAda_dev_proj*Cst_prj_yr_NA_dev
{Cost per year for NonAda development projects (dollars/year)}

Cost per project-year for non-Ada development projects (Equation #280)

○ Cst_prj_yr_NA_dev = Ref_cst_NA_dev*E_NA_cst
{Cost per project-year for NonAda development projects (dollars/year)}

Reference cost of non-Ada development projects (Equation #290)

○ Ref_cst_NA_dev = 6E6 {Reference cost for NonAda development projects (dollars/year)}

Effects on non-Ada cost (Equation #350)

○ E_NA_cst = E_int_NA_cst*E_inco_NA_cst
{Effects on NonAda costs (dimensionless)}

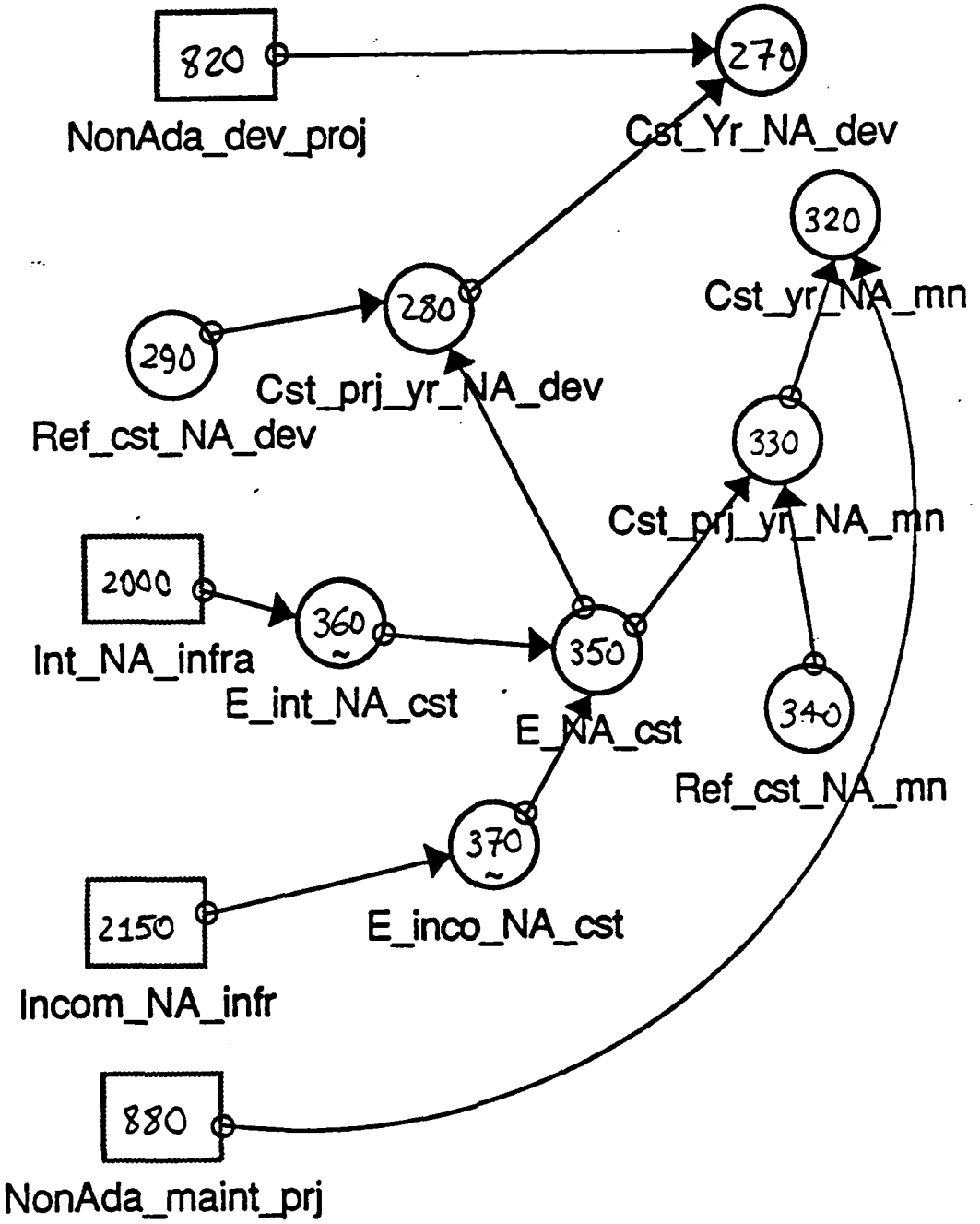
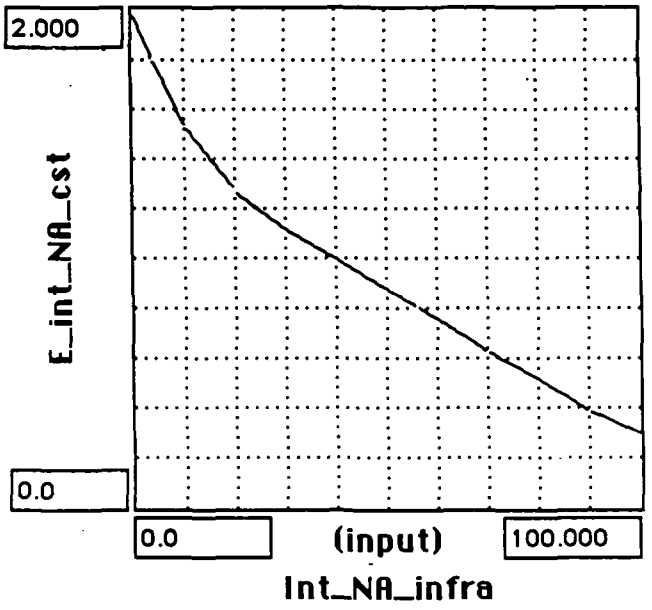


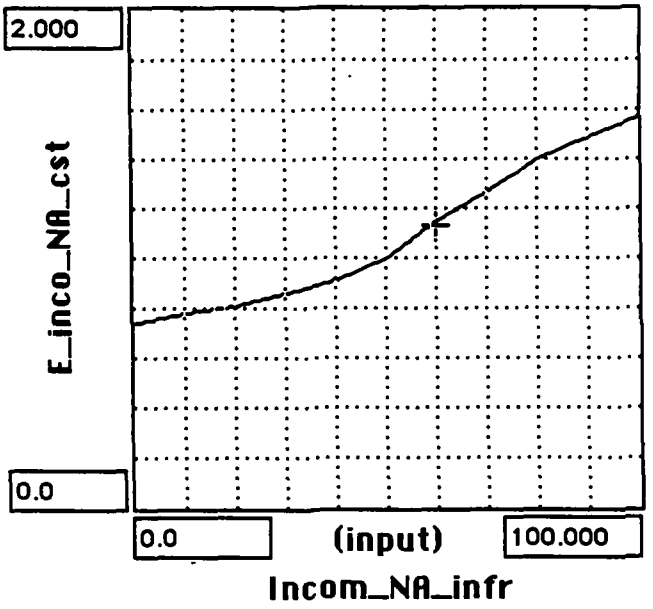
Figure A.3-3. Flow diagram of inputs to Cost per year for non-Ada development projects (Cst_yr_NA_dev).

Effect of intensity of infrastructure on non-Ada cost (Equation #360)



E_int_NA_cst = graph(Int_NA_infra)
0.0 -> 1.970
10.000 -> 1.530
20.000 -> 1.270
30.000 -> 1.120
40.000 -> 1.000
50.000 -> 0.880
60.000 -> 0.760
70.000 -> 0.630
80.000 -> 0.520
90.000 -> 0.390
100.000 -> 0.300

Effect of incompatibility of infrastructure on non-Ada cost (Equation #370)



E_inco_NA_cst = graph(Incom_NA_infr)
0.0 -> 0.740
10.000 -> 0.780
20.000 -> 0.810
30.000 -> 0.860
40.000 -> 0.910
50.000 -> 1.000
60.000 -> 1.150
70.000 -> 1.270
80.000 -> 1.400
90.000 -> 1.490
100.000 -> 1.570

Cost per year for non-Ada maintenance projects (Equation #320)

Cst_yr_NA_mn = NonAda_maint_prj*Cst_prj_yr_NA_mn
{Cost per year for NonAda maintenance projects (dollars/year)}

Cost per project-year for non-Ada maintenance projects (Equation #330)

Cst_prj_yr_NA_mn = Ref_cst_NA_mn*E_NA_cst
{Cost per project-year for NonAda maintenance (dollars/year)}

Reference cost for non-Ada maintenance projects (Equation #340)

○ Ref_cst_NA_mn = 3e6 {Reference cost for Non Ada maintenance projects (dollars/year)}

Total yearly cost of conversions (Equation #420)

Programs developed in non-Ada languages do not need to stay that way. Major redevelopments are one natural point at which conversion to Ada could be considered. There have already been proof-of-concept translations of three avionics packages for fighter aircraft into Ada, with good results: the F4J (DS&E 1985), the F-15 (Stanley 1985), and the F-20 (Suydam 1985). As experience with Ada accumulates, conversions of operational software during major upgrades should become more common.

Surprisingly, routine maintenance also offers opportunities to transit into Ada. One contractor writes routines in Ada, then uses an Ada-to-CMS2 translator to convert the code into CMS2 code compatible with the rest of the system (Mayfield, 1985). Gradually, more and more of the "original source" code is in Ada, with all the benefits thereof. Determining when such conversions are undertaken is represented elsewhere in the model; the cost section assesses the cost of such conversions, with the Total yearly cost of conversions. Figure A.3-4 shows the flow diagram of the computation.

The Total yearly cost of conversions (abbreviated Tot_yr_cst_conv in the model) is a converter variable, converting the number of standard Conversion projects and the Cost per project-year for conversions to a total dollars-per-year figure.

○ Tot_yr_cst_conv = Conv_prj*Cst_prj_yr_Conv {Total yearly cost conversions (dollars/year)}

Cost per project-year for conversions (Equation #430)

The Cost per project-year for conversions represents the yearly cost of any maintenance programming (major or minor) that begins with materials that were used to generate non-Ada source code and ends with Ada source code. This definition therefore excludes translation activities during the development phase such as translation from a specification language or program design language (PDL) into Ada source code. The definition would include, however, retranslation from a PDL into Ada instead of the non-Ada language. So if a PDL version of a program was used to generate source code in JOVIAL, one way of converting the JOVIAL program would be to translate from the PDL into Ada.

Cost per project-year for conversions, like the costs per project year of development and maintenance, is a yearly cost, not a life cycle cost. Even if conversions have a yearly cost comparable to development costs, the cost of completing a given conversion project can be much less if it is accomplished more quickly than developments.

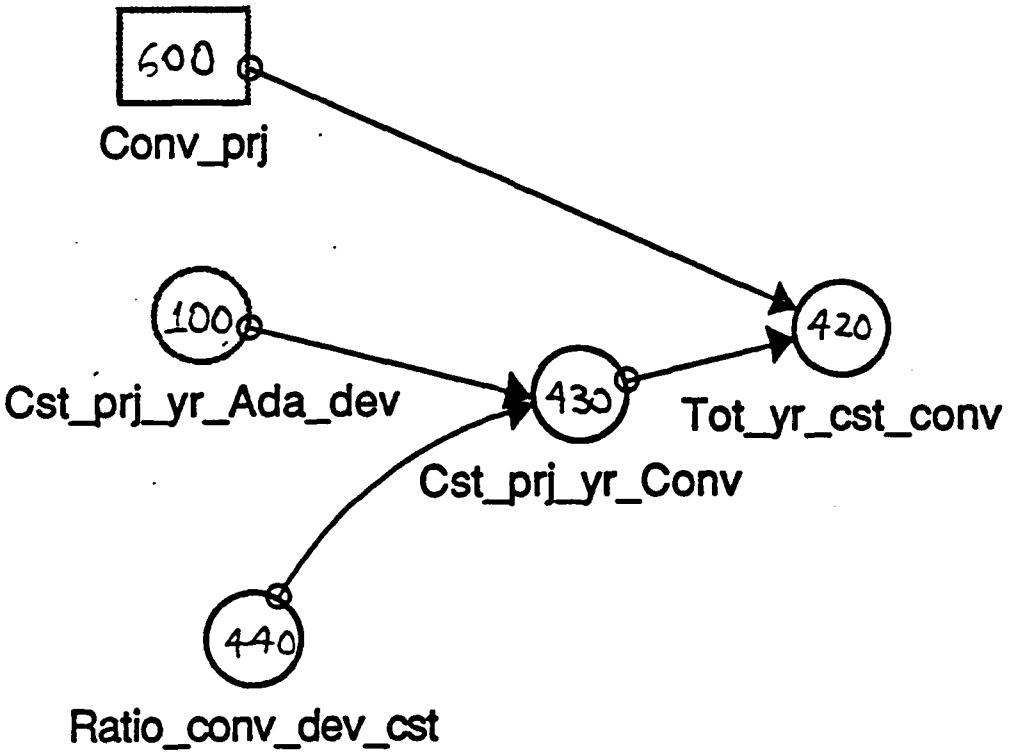


Figure A.3-4. Flow diagram of inputs to Total yearly cost of conversions (Tot_yr_cst_conv).

The Cost per project-year for conversions (abbreviated Cst_prj_yr_conv in the model) is a converter variable, converting current programming costs (represented by the Cost per project-year for Ada development projects) to costs for conversions, using the Ratio of development to conversion costs.

○ Cst_prj_yr_Conv = Cst_prj_yr_Ada_dev * Ratio_conv_dev_cst
{Cost per project-year for conversions (dollars/year)}

The model formulation assumes that the bulk of conversion costs will be incurred during major redevelopments or block upgrades, as opposed to smaller-scale and more routine maintenance. Therefore, the cost of doing such redevelopment/conversion programming in Ada should be very similar to (and therefore should be based on) the cost of developing new programs in Ada.

If a project is undergoing a major redevelopment and conversion to Ada, it is doubtful that cost-conscious managers would also want to simultaneously carry on a major redevelopment in the original non-Ada language. Therefore, once conversion work starts on a piece of software, it is removed from the pool of non-Ada projects immediately. (The projects sector contains the equations that accomplish this.) Of course, routine maintenance of operational software must continue even if the software is in the process of being redeveloped; the model includes these costs as part of conversion, just because the project is accounted for as a conversion project, not included in either the Ada or non-Ada pools of projects. This assumption about routine maintenance costs may create some slight inaccuracies if Ada programming costs are dramatically different from non-Ada programming costs, but the error should still be relatively small if routine maintenance expenses are substantially smaller than redevelopment expenses.

Ratio of conversion to development cost (Equation #440)

The Ratio of conversion to development cost characterizes the relative cost per year of working on a software project for the first time (development) versus a second time, third time, and so forth (redevelopment). As usual, the ratio characterizes averages, rather than any specific project: the average development project, the average extensiveness of pre-planned product improvement, mission change, and so on.

The Ratio of conversion to development cost (abbreviated Ratio_conv_dev_cst in the model) is a converter variable with no input, i.e. a constant.

○ Ratio_conv_dev_cst = 1
{Ratio of conversion to development costs (dimensionless)}

At this point in the development of the model, there is little evidence even as to whether the Ratio of conversion to development cost is greater than or less than 1.0. There are arguments both ways. For redevelopment, much of the software is already well-tested: the software operates to the specifications and the specifications are in fact what is needed. But in the model, conversion costs also include routine maintenance while the conversion is in process. If the architecture is well-worked out, it may be possible to "fan out" programming tasks into modules more rapidly than for a development project--more people involved sooner would give a higher yearly cost, even if the conversion is accomplished much more quickly than a development. And, although the original software may operate well, a formal specification may either not exist or be far out of date, which forces the redevelopment to start almost from scratch. Too, redevelopment is usually triggered by

substantial changes in either mission or capability, which are new programming problems. Might developing a system with new capabilities while operating more or less like the old be more complex and costly than developing a new system without the constraint of previous user training? In the absence of information on such questions, the ratio is set to 1.0; conversion projects are assumed to cost as much per year as new development projects.

Appendix A.4: Ada Projects Sector

The Ada projects sector of the model represents development and maintenance programming work in the Ada language for all Mission-Critical Computer Resources (MCCR). (Non-Ada projects are represented in a parallel but separate sector.) Every MCCR project start that involves software will increase either the number of Ada development projects or the number of Non-Ada development projects. That choice is determined in the language choice sector. Ada development projects become Ada maintenance projects, which usually endure for many years in the language in which they were first programmed. There is also a policy option to translate programs in maintenance phase from a non-Ada language to Ada. During the course of translation, the projects are classified in a separate category, conversion projects, which is also located in the Ada projects sector.

The Ada projects sector affects several other areas in the model. The numbers of projects in the three Ada categories (Ada development, Ada maintenance, and conversion) are the basis for calculating Ada software costs in the cost sector. Moreover, the amount of Ada programming work in progress (as measured by the numbers of projects) also determines how fast Ada infrastructure intensity, coverage, and incompatibility develop.

Ada development projects (Equation #500)

All MCCR systems in the development phase using the Ada language are classified in the model as Ada development projects. The only Ada programming not represented here is in projects already in the maintenance (i.e., post-development) phase, which may include projects to translate programs in the maintenance phase from a non-Ada language to Ada. Section 5 of this report discusses the exact definition of a programming project. Figure A.4-1 shows the inputs to Ada development projects on a flow diagram.

Ada development projects (abbreviated Ada_dev_prj in the model) is a level variable; it accumulates the rate of flow into it (Ada project starts) and is depleted by the outflow rate (Ada development project completions).

```
□ Ada_dev_prj = Ada_dev_prj + Ada_dev_starts - Ada_dev_compl
  INIT(Ada_dev_prj) = 0 {Ada development projects
    (projects)}
```

Ada development projects is initialized at zero, since the simulation starts prior to the definition of Ada as a language and prior to the availability of compilers or any other infrastructure.

Ada development project starts (Equation #510)

All new MCCR development projects by definition use either the Ada language or some non-Ada language. The development work started each year in Ada is called Ada development project starts.

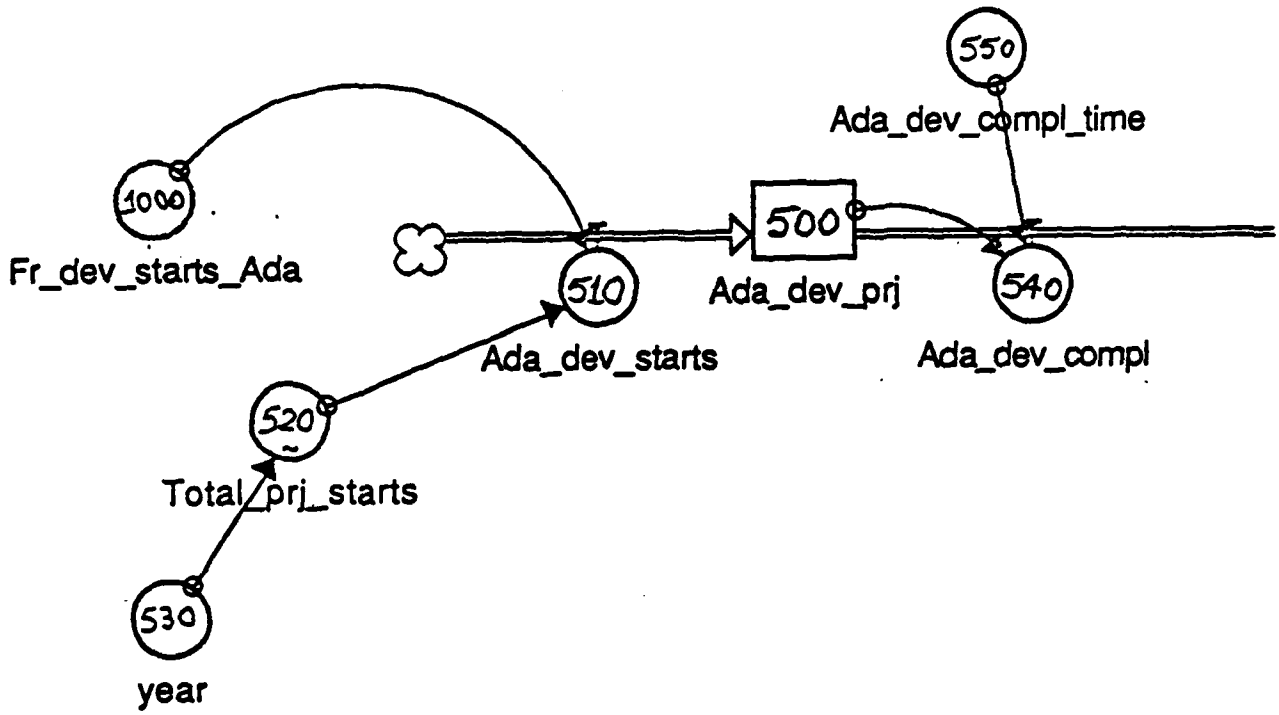


Figure A.4-1. Flow diagram of inputs to Ada development projects (Ada_dev_proj). The flow exiting to the right goes to Ada maintenance projects.

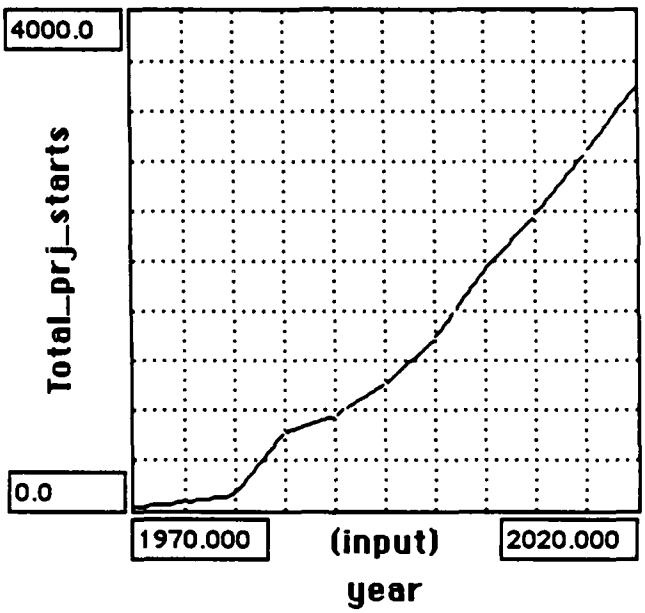
Ada development project starts (abbreviated Ada_dev_starts in the model) is a rate variable, flowing into the level of Ada development projects. A rate variable is a special variation of a converter variable. Here, Total development project starts and the Fraction of development project starts in Ada are converted into a number of projects per year starting in Ada.

○ Ada_dev_starts = Total_prj_starts*Fr_dev_starts_Ada
{Ada development (project) starts (projects/year)}

Total development project starts (Equation #520)

All Mission Critical Computer System projects are classified as development projects until they officially become maintenance projects. The yearly rate at which development projects are initiated is Total development project starts.

Total development project starts (abbreviated Total_prj_starts in the model) is a converter variable, using a graphic function to convert a calendar year into the number of development projects started.



⊘ Total_prj_starts = graph(year)
1970.000 -> 15.000
1975.000 -> 50.000
1980.000 -> 120.000
1985.000 -> 620.000
1990.000 -> 760.000
1995.000 -> 1020.000
2000.000 -> 1390.000
2005.000 -> 1915.000
2010.000 -> 2360.000
2015.000 -> 2870.000
2020.000 -> 3380.000

The derivation of specific numbers for this variable is described in Appendix A.9, "Multivariable Model Calibration."

The number of starts is assumed to increase each year. As computer technology, both software and hardware, continues to advance, more and more applications for computers in MCCR systems become cost-effective. This study focusses on ways of causing costs of software production to fall more rapidly. However, it is beyond the scope of this study to characterize how software costs feed back to influence demand for software. Total development project starts is an exogenous variable, i.e., its value is not influenced by other variables within the system.

It is not expected that assumptions about the number of project starts will be critical in determining which standards policies are most desirable, within a broad range of plausible values for yearly project starts. In any event, this is the primary candidate for contingency testing.

Year (Equation #530)

The yearly progression of time from 1975 to 2015 is represented in the variable called, mnemonically enough, Year.

Year (written Year in the model) is a converter variable, converting the built-in variable TIME into a model variable complete with a flow diagram symbol.

○ year = TIME {years (years)}

The variable, Year, was created to write as an explicit input to a graph function, since the STELLA simulation package's syntax does not allow using the TIME variable directly.

The range of time simulated with the model can be changed in the "Specs" menu. However, if this is done, the tables drawn as functions of time (Year) must be checked to ensure that they are properly defined over the specified range.

Ada development project completions (Equation #540)

The transition from a development project to a maintenance project is a clearly defined and formal event. Sometimes it is marked by handing the project over to a different team; sometimes its is merely a signing of papers and a transition of budget authority. For a contractor delivering on a contract obligation this event has important legal and financial implications. The number of projects each year that make this transition is the rate of Ada development project completions.

Ada development project completions (abbreviated Ada_dev_compl in the model) is a rate variable; it is an outflow to Ada development projects and an inflow to Ada maintenance projects.

○ Ada_dev_compl = Ada_dev_prj/Ada_dev_compl_time
{Ada development (project) completion time
(projects/year)}

This representation of the rate as the level divided by the average dwell time, Ada development completion time, will give an average continuous flow of Ada development project completions.

The completion time is no more than an AVERAGE dwell time in the model, just as in real life. Some projects will be completed within months. Some will persist for two and three times the average. The structure of level and outflow rate used here forms what is known as a first-order delay or lag; the average time a project spends as a development project is the Ada_dev_compl_time.

The mathematics of the first-order delay are such that if a batch of projects were all started at once (with none starting thereafter), both the number of projects remaining and the completion rate would decline exponentially. When time equal to the average completion time has passed, about 70 percent of the projects will already have been completed. About 30 percent of the projects would still be incomplete. About 30 percent of those will remain after 2 times the average completion time, and so on. It is these stragglers that make the average dwell time equal to the Ada_dev_compl_time. For further discussion, see (Forrester 1969, Sections 2.2 and 10.2; Goodman 1974, Ch. 3; and Alfeld and Graham 1976, Section 3.5)

Ada development project completion time (Equation #550)

Although projects vary greatly in the time they take to be complete, with a large enough sample it is possible to think of an average development duration. This parameter is called Ada development project completion time.

Ada development project completion time (abbreviated Ada_dev_compl_time in the model) is a converter variable with no inputs, i.e., a constant.

○ Ada_dev_compl_time = 10 {Ada development (project) completion time (
years) }

The project completion time represents the time it takes from inception of development the weapons platform or other system of which programming is a part, all the way to becoming a developed, completed, maintained, and usually deployed system. This time differs from the time it takes to complete one program module. Before any given module is actually programmed, there are rounds of system definition and specification, and often layers of calling routines to be written first. After a module is programmed, it will usually be rewritten, revised, or possibly discarded entirely due to design changes in other parts of the system, all as part of the development process. The time required for the entire process is what is measured by the Ada development project completion time.

Conversion projects (Equation #600)

Supposing that Ada and its APSE will eventually become an attractive programming alternative, the issue arises as to what to do about the non-Ada programs already deployed and being maintained. As has been discussed in the Cost sector, one alternative is translation to Ada, either on a module-by-module basis, or as an integral part of major redevelopments. The projects where such translations are being done Conversion projects. Figure A.4-2 shows the inputs on a flow diagram.

Conversion projects (abbreviated Conv_prj in the model) is a level variable; it accumulates the rate of flow into it (Conversion project starts) and is depleted by the outflow rate (Conversion project completions).

□ Conv_prj = Conv_prj - Conv_prj_compl + Conv_prj_starts
INIT(Conv_prj) = 0 {Conversion projects (projects)}

Conversion projects is initialized at zero; there can be no conversion projects until there is Ada.

Conversion project completions (Equation #610)

Once the conversion work is done -- once whatever direct translation is possible is finished, along with the new Ada programming -- the project (or fraction of a project) becomes just another Ada maintenance project. The number of such completions per year is the Conversion project completions.

Conversion project completions (abbreviated Conv_prj_compl in the model) is a rate equation which flows out of Conversion projects and into Ada maintenance projects.

○ Conv_prj_compl = Conv_prj / Conv_compl_time
{Conversion project completions (projects/year)}

The rate of Conversion project completions is the outflow rate for a standard first-order lag, with the time constant being the Conversion project completion time.

Conversion project completion time (Equation #620)

Some conversions will take a week or two, if they are merely rewriting a subroutine in Ada. Others will last several years, if they are major redevelopments stemming from substantial mission changes. But in the aggregate, there is an average, which in the model is the Conversion project completion time.

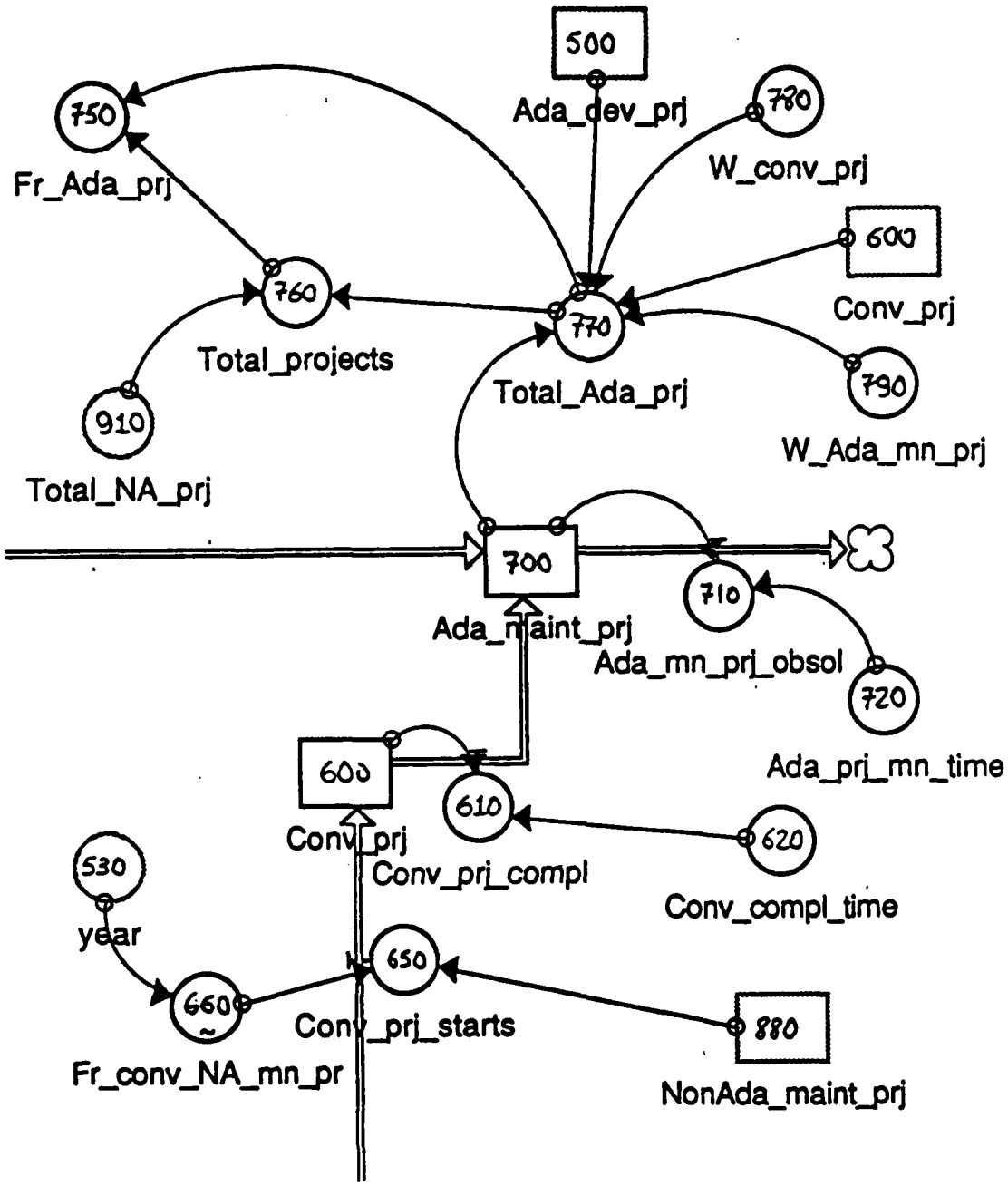


Figure A.4-2. Flow diagram of inputs to Conversion projects (Conv_proj). The flow from the left is Ada development project completions. The flow from the bottom comes from Non-Ada maintenance projects.

Conversion project completion time (abbreviated Conv_compl_time in the model) is a converter variable with no inputs, i.e., a constant.

○ Conv_compl_time = 2 {Conversion project completion time (years)}

Conversion project starts (Equation #650)

The moment programmers start specifying the rewriting of non-Ada code that is to result in Ada code, the project (or part of a project) involved is classified as a conversion project. The yearly rate at which such projects are initiated is the Conversion project starts.

Conversion project starts (abbreviated Conv_prj_starts in the model) is a rate equation. It is the flow of projects out of the Non-Ada maintenance level and into the Conversion projects level.

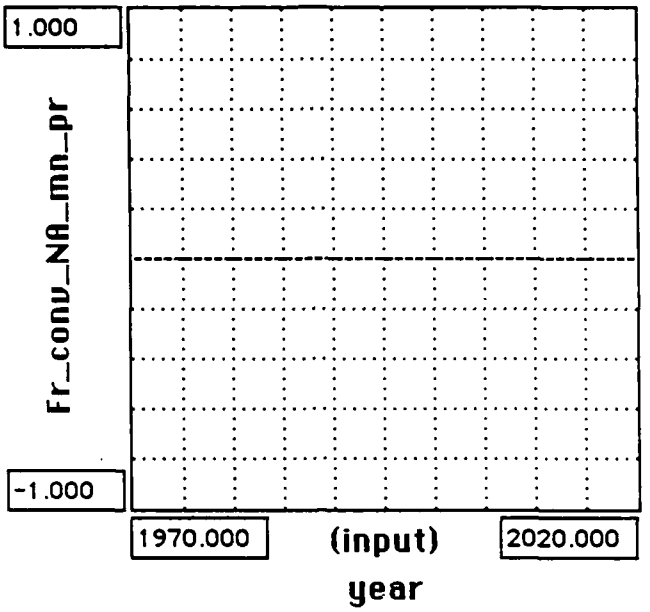
○ Conv_prj_starts = NonAda_maint_prj * Fr_conv_NA_mn_pr {Conversion project starts (projects/year)}

Conversion project starts is formulated as a fraction of the remaining pool of Non-Ada projects. Even if DoD and the services choose a high fractional conversion rate, the absolute number of conversion starts will slow down as the remaining pool of Non-Ada projects gets smaller. This is a realistic because those remaining projects represent the more difficult, less cost-effective conversion efforts, which should either be saved for last when the process is well-understood or not converted, just ignored until they and the system in which they are embedded obsolesce.

Fractional conversion rate of non-Ada maintenance projects (Equation #660)

A DoD policy encouraging the conversion of non-Ada projects to Ada is a potentially powerful lever for influencing how quickly Ada establishes itself. Such a policy is represented by the Fractional conversion rate of non-Ada maintenance projects.

Fractional conversion rate of non-Ada maintenance projects (abbreviated Fr_conv_NA_mn_pr in the model) is a converter variable, which uses a graphical function to convert the calendar Year into a fractional flow rate.



Fr_conv_NA_mn_pr = graph(year)
1970.000 -> 0.0
1975.000 -> 0.0
1980.000 -> 0.0
1985.000 -> 0.0
1990.000 -> 0.0
1995.000 -> 0.0
2000.000 -> 0.0
2005.000 -> 0.0
2010.000 -> 0.0
2015.000 -> 0.0
2020.000 -> 0.0

Conversion policy is a policy lever that is inactive in the base model; the values are set to zero at all times. The fraction is expressed as a graphical function of time to allow experiments that create a flow of conversions. But such a flow would be realistic only after Ada is available as a programming language, i.e., the formulation must provide for zero conversions before the mid-1980's and the potential for conversions thereafter.

Ada maintenance projects (Equation #700)

Ada development projects and conversion projects, when completed, result in software for developed (usually deployed) MCCR systems. There is still, however, a substantial amount of work required to both improve the reliability of such systems through routine maintenance, and to keep the system capabilities up to current military requirements. In the terminology used here, all such activities are termed maintenance. The number of projects being maintained that use the Ada language is called Ada maintenance projects.

Ada maintenance projects (abbreviated Ada_maint_prj in the model) is a level variable. Its inflows are the Ada development project completions and the Conversion project completions. The outflow from the level is the Ada maintenance project obsolescence.

□ Ada_maint_prj = Ada_maint_prj + Ada_dev_compl - Ada_mn_prj_obsol + Conv_prj_compl
INIT(Ada_maint_prj) = 0 {Ada maintainance projects (projects)}

Ada maintenance projects is initialized at 0.0, since at the start of the simulation in 1975, there were no maintenance projects written in Ada.

Ada maintenance project obsolescence (Equation #710)

After their useful lifetime passes, weapons and communications systems finally become too old or antiquated to be worth maintaining any longer. When this happens they are retired from duty, and if computers are embedded in the systems, the software that runs them no longer needs to be maintained. Undoubtedly, a similar process happens for non-embedded systems also -- a given set of programs gets too far from current needs and acquires too many repairs on top of repairs. Or changes in hardware costs over the years make entirely new forms of computation desirable. Eventually, it is cost-effective to develop an entirely new system and scrap the old. There comes a time when every program is no longer used. The number of Ada programming projects that pass out of use each year is represented by the rate of Ada maintenance project obsolesence.

Ada maintenance project obsolescence (abbreviated Ada_mn_prj_obsol) in the model is a rate equation; it depletes the level of Ada maintenance projects.

○ Ada_mn_prj_obsol = Ada_maint_prj/Ada_prj_mn_time {Ada maintainance project obsolescence (projects/year)}

The rate is formulated in the standard first-order delay format, with the Ada project maintenance time as the time constant.

Ada project maintenance time (Equation #720)

The lifetime of a weapon system can be quite long. The B52 as a weapons system, which first entered limited production in 1952, is therefore 34 years old and still going (Fahey 1956). On the other hand, many weapons systems would have much shorter lifetimes. In the aggregate an average of 20 years seems plausible. This time is represented in the model as the Ada project maintenance time.

Ada project maintenance time (abbreviated Ada_prj_mn_time in the model) is a converter variable with no inputs, i.e., a constant.

○ Ada_prj_mn_time = 20 {Ada project maintainance time (years) -- the time it takes for the system in which the technology is embedded to pass out of useful service}

Fraction of Ada projects (Equation #750)

The Fraction of Ada projects represents that portion of the total MCCR programming that is done in the Ada language. This fraction is used in the infrastructure sectors of the model to represent how much Ada programming (and non-Ada programming) is going on, which influences how rapidly the intensity of infrastructure is created.

Fraction Ada projects (abbreviated Fr_Ada_prj in the model) is a converter variable, converting the Total projects and the Total Ada projects into a fraction.

○ Fr_Ada_prj = Total_Ada_prj/Total_projects
{Fraction of Ada projects (dimensionless)}

Total projects (Equation #760)

Total projects is a measure of the amount of programming work that is going on at any particular time. As will be discussed further below, there is a weighted sum of maintenance, conversion, and development projects, the weightings representing the differing amount of programming work implied by a standard project within each of those categories. For example, a system in the maintenance phase might entail less programming work per year than would a system of equivalent complexity in the development phase. (Whether it actually would or not is a matter for further investigation, as discussed elsewhere in this report.)

Total projects (written Total_projects in the model) is a converter variable, summing the Total Ada projects and the Total non-Ada projects.

○ Total_projects = Total_Ada_prj+Total_NA_prj {Total projects (projects)}

Total Ada projects (Equation #770)

Total Ada projects represents the total amount of Ada programming going on at any particular time in all categories of projects: development, maintenance, and conversion.

Total Ada projects (abbreviated Total_Ada_prj in the model) is a converter variable, summing development, maintenance, and conversion projects, with weighting coefficients on the latter two.

○ Total_Ada_prj = Ada_dev_prj+(Ada_maint_prj*W_Ada_mn_prj)+(Conv_prj W_conv_prj)
{Total Ada projects (projects)}

The weighting factors represent the difference in programming effort required for the different types of programming project. It may take less effort to maintain a project than it does to either convert it from non-Ada or to develop it from scratch.

Weight for conversion projects (Equation #780)

Weight for conversion projects represents the programming effort required per year to convert a non-Ada project to Ada, relative to a development project.

Weight for conversion projects (abbreviated W_conv_prj in the model) is a converter variable with no inputs, i.e., a constant.

○ W_conv_prj = 1 {Weight for conversion upgrades (dimensionless)}

The representation as a constant implies that the relative programming efforts for the different kinds of programming projects (development, maintenance, and conversion) does not change much over the span of the simulation. As discussed in Appendix A.3 on the cost sector, the value of 1.0 indicates that it takes as much effort per year to convert a non-Ada project to an Ada project as it does to develop an Ada project *de novo*. (Development projects take longer to complete, so the life cycle cost of development is higher than that of conversion.)

Weight for Ada maintenance projects (Equation #790)

Weight for Ada maintenance projects indicates the programming effort required to maintain Ada projects relative to the effort required to develop them.

Weight for Ada maintenance projects (abbreviated W_Ada_mn_prj in the model) is a converter variable with no inputs. Its value is constant throughout the simulation.

○ W_Ada_mn_prj = .5 {Weight for Ada maintainance projects (dimensionless

The weighting value of .5 means that the model assumes that maintaining an Ada standard project takes half as much programming effort per year does developing one.

Appendix A.5: Non-Ada Projects Sector

The non-Ada projects sector of the model represents development and maintenance programming work in all non-Ada languages. Every development project start that involves software will increase either the number of Ada development projects or the number of Non-Ada development projects. That choice is determined in the language choice sector. Non-Ada development projects become Non-Ada maintenance projects, which usually endure for many years in the language in which they were first programmed. There is also a policy option to translate programs in maintenance phase from non-Ada to Ada. During the course of translation, the projects are classified in a separate category, conversion projects, which are not part of the non-Ada projects sector.

The non-Ada projects sector affects several areas in the model. The numbers of projects in the two non-Ada categories (development, and maintenance) are the basis for calculating non-Ada software costs in the cost sector. Moreover, the amount of non-Ada programming in progress determines how fast non-Ada infrastructure intensity, coverage, and incompatibility develop.

The Non-Ada projects sector is structurally identical to the Ada projects sector, with two exceptions, both due to the presence of conversion projects. First, conversion projects flow out of the Non-Ada maintenance project level and into the Ada maintenance project level. Second, the summation for Total non-Ada projects has no term for conversion projects, since at least the coding phases of those projects are conducted in the Ada language.

The parameters of the Non-Ada projects section are exactly equal to the corresponding parameters in the Ada projects section. Because of the close similarity of the Ada projects sector to the non-Ada projects sector, no further verbal description will accompany the equations for the latter.

Non-Ada development projects (Equation #820)

Figure A.5-1 shows the inputs for non-Ada development projects on a flow diagram.

□ NonAda_dev_proj = NonAda_dev_proj + NA_dev_starts - NA_dev_compl
INIT(NonAda_dev_proj) = 150
{NonAda development projects (projects)}

Non-Ada development project starts (Equation #830)

○ NA_dev_starts = Total_prj_starts * (1-Fr_dev_starts_Ada)
{Non-Ada development (project) starts (projects/year)}

Non-Ada development project completions (Equation #840)

○ NA_dev_compl = NonAda_dev_proj/NA_dev_compl_time
{Non-Ada development (project) completions (projects/year)}

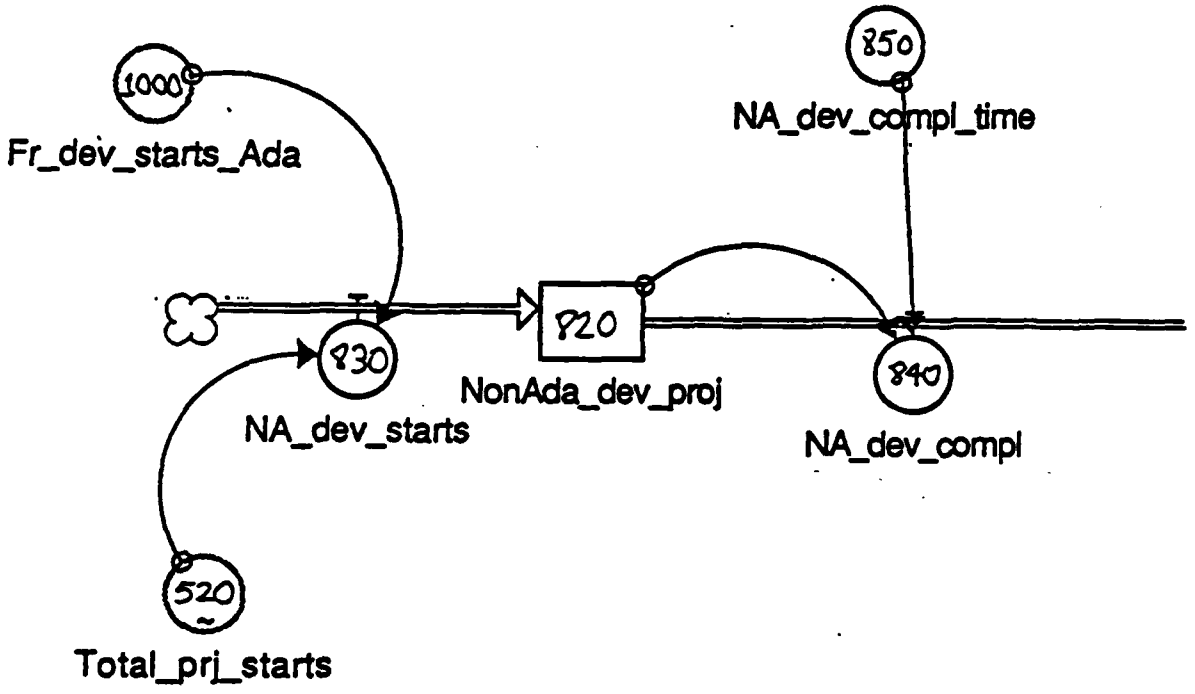


Figure A.5-1. Flow diagram of inputs to Non-Ada development projects (NonAda_dev_proj). The flow exiting to the right goes into Non-Ada maintenance projects.

Non-Ada development project completion time (Equation #850)

○ NA_dev_compl_time = 10
{NonAda development (project) completion time (years)}

Non-Ada maintenance projects (Equation #880)

Figure A.5-2 shows the flow diagram for Non-Ada maintenance projects.

□ NonAda_maint_prj = NonAda_maint_prj + NA_dev_compl - NA_mn_prj_obsol - Conv_prj_starts
INIT(NonAda_maint_prj) = 90
{NonAda maintenance projects (projects)}

Non-Ada maintenance project obsolescence (Equation #890)

○ NA_mn_prj_obsol = NonAda_maint_prj/NA_proj_mn_time {NonAda maintainance project obsolescence (projects/year)}

Non-Ada project maintenance time (Equation #900)

○ NA_proj_mn_time = 20 {NonAda project maintainance time (years)}

Total non-Ada projects (an output) (Equation #910)

○ Total_NA_prj = NonAda_dev_proj+((NonAda_maint_prj-Conv_prj)* w_NA_mn_prj)
{Total NonAda projects (projects)}

Weight for non-Ada maintenance projects (Equation #920)

○ w_NA_mn_prj = .5

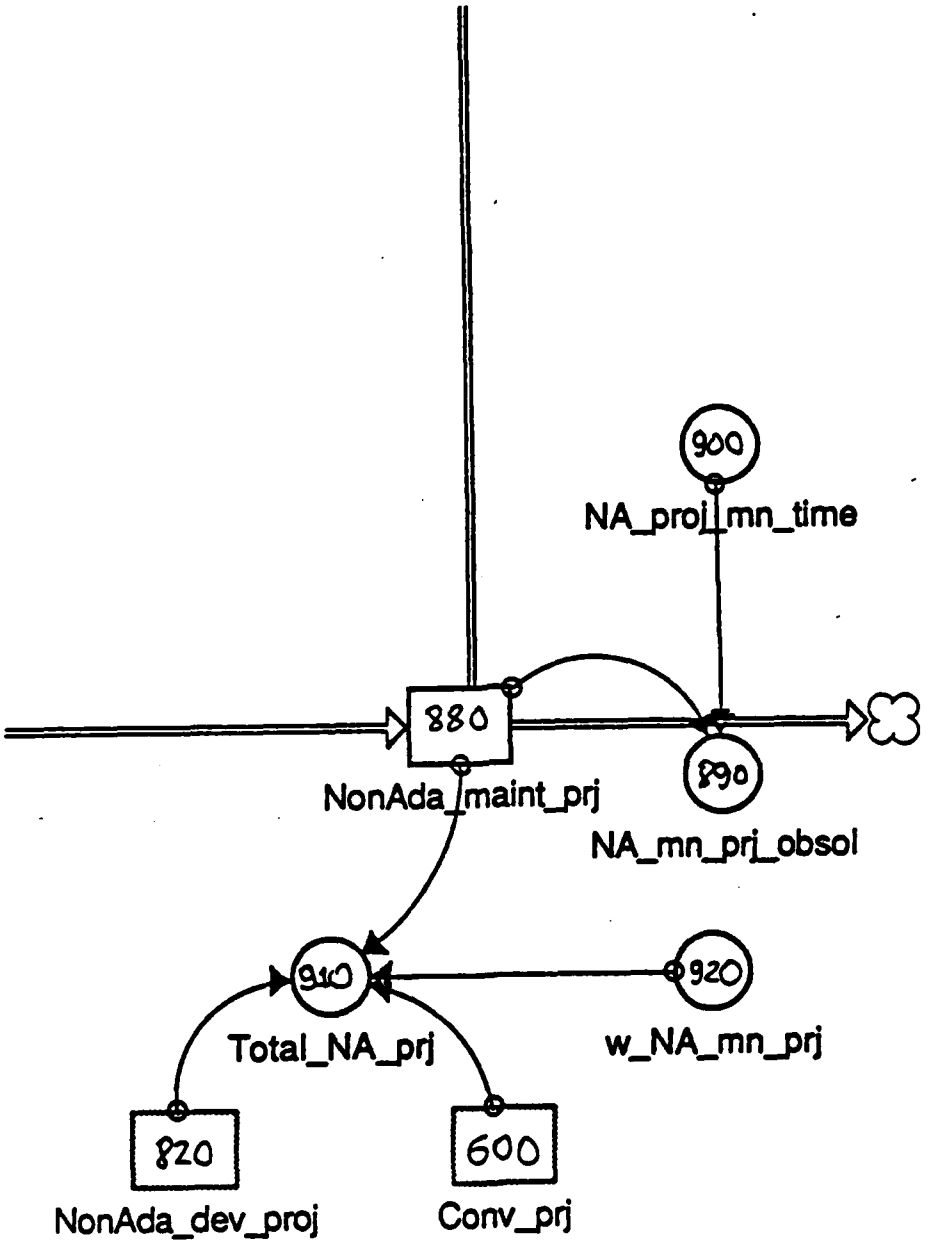


Figure A.5-2. Flow diagram of inputs to Non-Ada maintenance projects (NonAda_maint_proj). The flow exiting upwards goes into Conversion projects. The flow entering from the left is the Non-Ada development project completions.

Appendix A.6: Language Choice Sector

The language choice sector represents the considerations made in choosing the language in which MCCR systems will be programmed. Rather than representing all language options equally, the sector represents choosing Ada versus the aggregate of all other languages. The basis for choosing Ada versus another language is coverage and intensity of infrastructure (i.e., whether sufficient skilled people, software tools, and modern hardware for programmers are available; and available for the desired combination of host and target computers), and beyond these characteristics, the incentives that arise from the DoD having standardized on Ada.

Fraction of development project starts in Ada (Equation #1000) (an output)

In the aggregate, the various considerations about whether or not to use Ada can be summarized in the fraction of programming work starting up that uses Ada. In the model this is the Fraction of development project starts in Ada. Figure A.6-1 shows this variable and its inputs.

Fraction of development project starts in Ada (abbreviated Fr_dev_starts_Ada in the model) is a converter variable, combining the Natural fraction of Ada starts (that would arise from free market choices, without the influence of DoD standardization) with the influence of DoD standardization (the Effect of target on starts).

○ Fr_dev_starts_Ada = Nat_fr_Ada_starts * E_target_on_starts
(Fraction of development (project) starts in Ada
(dimensionless))

Unlike most other sectors, the language choice sector equations are not organized around a level variable; the function of this sector is to gather information about other level variables, infrastructure variables in particular, from other sectors and combine them into a single result. Just as a reminder of this difference with most other sectors, the heading of the first equation calls the Fraction of development project starts in Ada an output variable, to distinguish it from a state variable.

Effect of target on starts (Equation #1010)

The DoD and Congress have set in motion a number of incentives to use Ada beyond the free-market considerations of how well it should work in the project. The Authorization Act of 1983 stated:

> The Department of Defense should accelerate the implementation of the Ada higher order language and constrain to the maximum extent feasible service variations on Ada to ensure the utmost commonality of systems support software.

Accordingly, the "DeLauer Memo" (DeLauer 1983) of June 10, 1983 from Richard DeLauer, then Under Secretary of Defense for Research and Engineering (USDRE), mandated the use of Ada, "consistent with approved introduction plans, in all mission-critical defense systems that enter advanced development status after January 1, 1984 or that enter full-scale engineering development status after July 1, 1984." (quote from DoD Computer Technology (Study Annex): A Report to Congress, 1984). A 1985 memorandum from the current USDRE, Dr. Hicks, confirms this policy (Hicks 1985).

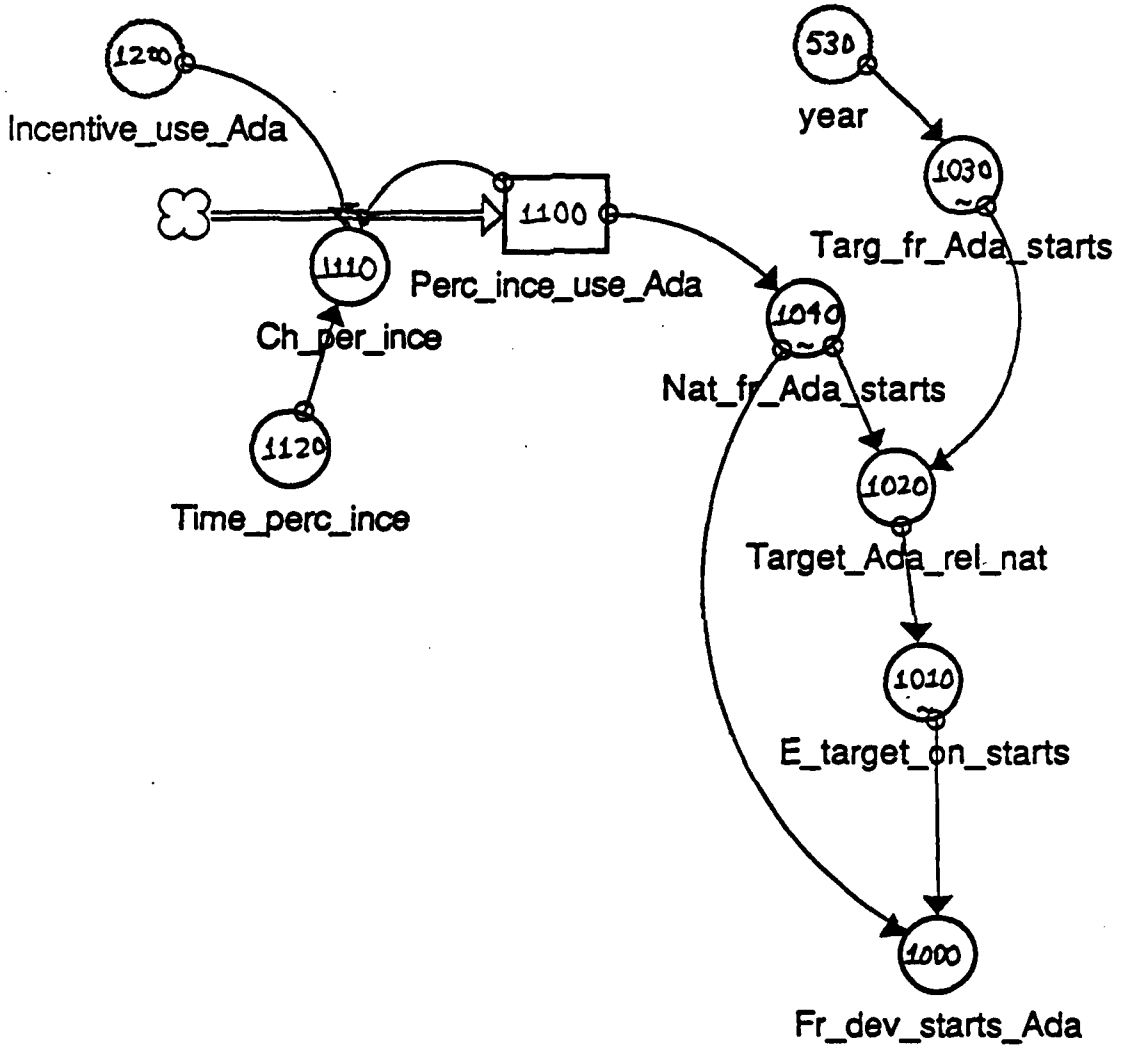
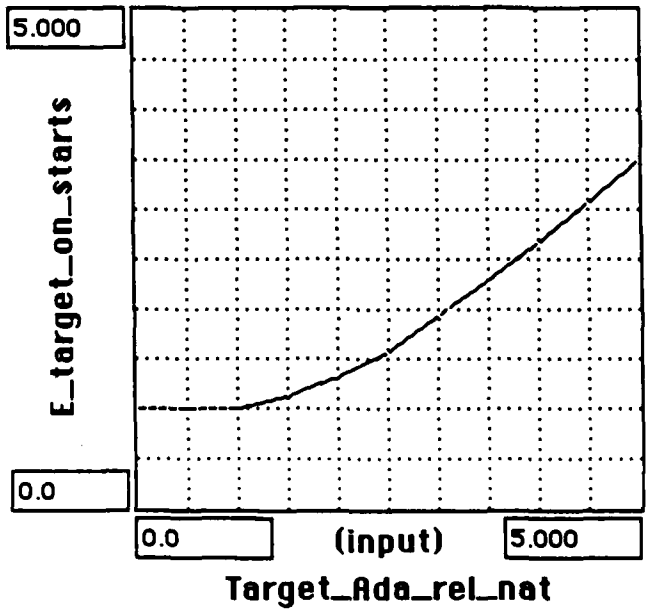


Figure A.6-1. Flow diagram of inputs to Fraction of development starts in Ada (Fr_dev_starts_Ada).

The policies on Ada use create incentives to use Ada above and beyond strict cost and effectiveness issues. Obtaining a waiver to use a non-Ada language is time- and resource-consuming. The process may not generate goodwill between the programming organization and the DoD program officer. Moreover, the chain of evaluations and approvals could cause substantial delays: The program officer must propagate the request for a waiver up the chain of command within the service, and large projects must also involve the DoD's Research, Development, Testing, and Evaluation (RDT&E) approval. Moreover, choosing a non-Ada language causes an organization to lose the ability to accrue Ada experience for later projects. Such considerations can increase the use of Ada beyond what would be indicated by free-market considerations alone. This is represented in the model by the Effect of target on starts.

The Effect of target on starts (abbreviated E_target_on _starts in the model) is a converter variable, which uses a graphic function to convert the discrepancy between the DoD target fraction and the natural fraction into an effect on starts.



E_target_on_starts = graph(Target_Ada_rel_nat)

0.0 -> 1.000
0.500 -> 1.000
1.000 -> 1.000
1.500 -> 1.125
2.000 -> 1.325
2.500 -> 1.575
3.000 -> 1.950
3.500 -> 2.300
4.000 -> 2.675
4.500 -> 3.075
5.000 -> 3.500

This formulation allows the fraction of project starts actually using Ada to be a compromise between the target, or mandated, fraction and the "natural" or free market fraction. If the mandated fraction is smaller than the natural fraction, the actual fraction equals the natural fraction. If the mandated fraction exceeds the natural fraction, the actual rate will be somewhat higher than the natural rate.

If the mandate to use Ada were completely effective (in the sense of compelling the actual fraction of starts to equal the target fraction), the curve would slope upwards at 45 degrees for inputs above 1.0. For those inputs, the effect would then equal the Target for Ada starts relative to natural fraction (Target_Ada_rel_nat). The discussion below will define this variable as the Target fraction for Ada starts (Targ_fr_Ada_starts) divided by the Natural fraction of Ada starts (Nat_fr_Ada_starts). Starting from equation 1000 and substituting,

Fr_dev_starts_Ada

= Nat_fr_Ada_starts * E_target_on_starts

= Nat_fr_Ada_starts * (Target_Ada_rel_nat)

= Nat_fr_Ada_starts * (Targ_fr_Ada_starts / Nat_fr_Ada_starts)

= Targ_fr_Ada_starts.

So if the curve were level at 1.0 for inputs below 1.0, and sloped at 45 degrees for inputs above 1.0, the Fraction of development project starts in Ada would be the natural fraction or the target fraction, whichever was higher.

The graph does not reach the 45-degree line, however, which represents limitations on the extent to which DoD incentives can increase Ada usage above the natural rate. For example, if the natural fraction is 10 percent and the target fraction is 50 percent, the Target for Ada starts relative to natural fraction will be 5.0. The Effect of target on starts, from the table above, will reach 3.5, so the actual Fraction of development project starts in Ada will be 0.10 x 3.5, i.e., 35 percent.

Target for Ada starts relative to natural fraction (Equation #1020)

The goal for the fraction of development starts done in Ada that the DoD aims can differ from the free market fraction. At first, the DoD target may be higher. If Ada is as successful as is hoped, the free market fraction some day may substantially exceed the DoD target. The measure of the difference between the two, a ratio, is called the Target for Ada starts relative to natural fraction.

Target for Ada starts relative to natural fraction (abbreviated Target_Ada_rel_nat in the model) is a converter variable, taking the ratio of the Target fraction for Ada starts divided by the Natural fraction of Ada starts.

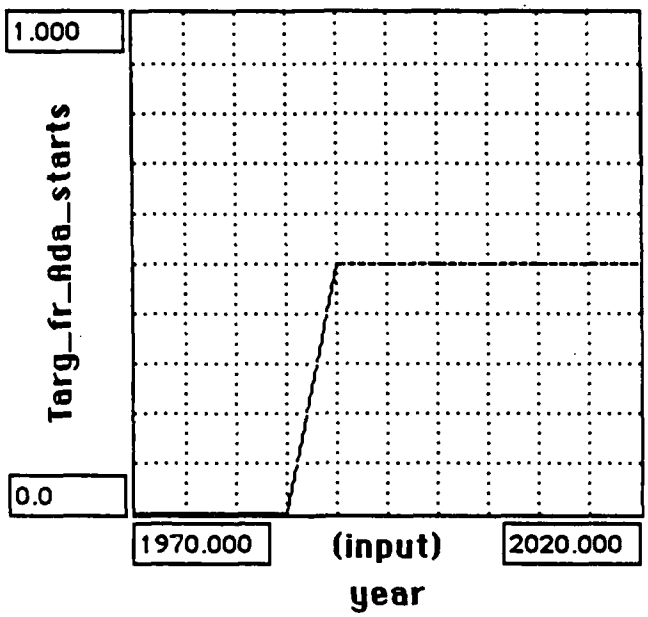
○ Target_Ada_rel_nat = Targ_fr_Ada_starts/Nat_fr_Ada_starts (Target for Ada (starts) relative to natural (fraction) (fraction))

Target fraction for Ada starts (Equation #1030)

The DeLauer memorandum (DeLauer 1983) mandating Ada use defines a category of projects as requiring the use of Ada: those software projects for embedded computer resources (ECR). That memo implicitly defines the mandated fraction of projects using Ada. In the model, the existence of a fraction of MCCR starts for which Ada is mandated is represented by the Target fraction for Ada starts.

The DeLauer memorandum (DeLauer 1983) requires Ada to be used for all mission-critical computer resource (MCCR) programming projects starting in 1984 and later. This is the *de jure* requirement. *De facto*, as a practical matter, there are some systems that will be sufficiently more effective written in some other language that waivers to Ada use will be given. And because experience with Ada builds up gradually, the granting of waivers to use non-Ada languages will be the rule rather than the exception at first. In fact, such a "rule" was formalized as Navy policy up until 1985: a waiver had to be obtained to *use* Ada for embedded applications. The changing set of policies mandating when and how Ada is to be used is represented in the model by the Target fraction for Ada starts.

Target fraction for Ada starts (abbreviated Targ_fr_Ada_starts in the model) is a converter variable, which uses a graphic function to convert the calendar year into the target fraction.



Targ_fr_Ada_starts = graph(year)
1970.000 -> 0.0
1975.000 -> 0.0
1980.000 -> 0.0
1985.000 -> 0.0
1990.000 -> 0.500
1995.000 -> 0.500
2000.000 -> 0.500
2005.000 -> 0.500
2010.000 -> 0.500
2015.000 -> 0.500
2020.000 -> 0.500

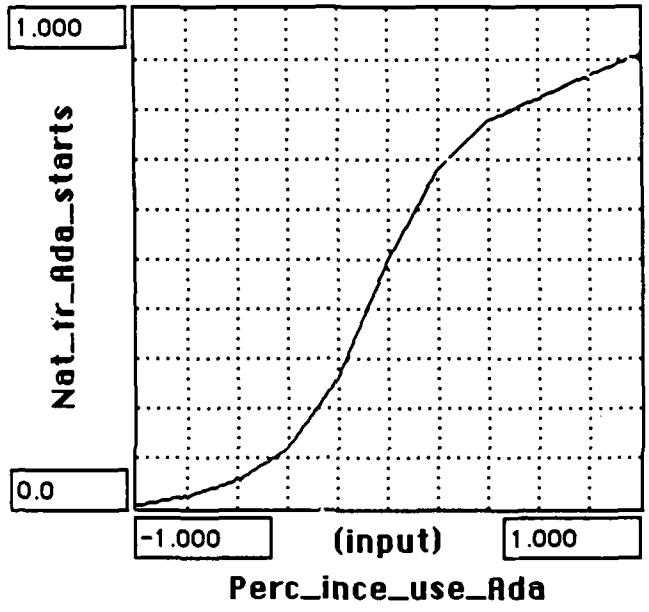
The graphic function in the STELLA simulation software allows only limited resolution in terms of how many data points the graph may contain. A graphic function running from 1970 to 2020 can only show changes in the graph every 5 years. Therefore, the Target fraction for Ada will be zero at 1985 and the full 0.5 five years later in 1990. In between, the target is computed by linear interpolation, so it is 0.20 in 1986, 0.40 in 1987, and so on. This treatment in the model can be thought of as representing the *de facto* rather than the *de jure* mandate: On 1/1/84, there were only two validated Ada compilers, and waivers on Ada use were the policy rather than the exception. In effect, the true target for what is proper for a fraction of Ada starts does in fact ramp up gradually, reflecting the operational swing in policy from non-use to use of Ada over several years.

The post-1990 target for Ada starts is only 0.5 of all MCCR starts, representing a rather lax enforcement of the DeLauer and Hicks memoranda (i.e., the class of projects routinely granted waivers is fully half of MCCR projects).

Natural fraction of Ada starts (Equation #1040)

In the absence of any DoD pressure or guidelines Ada would compete with other languages entirely on its own merits. As infrastructure intensity develops and Ada compilers become available on more hardware the fraction of development project starts for which Ada would properly be chosen would be larger. This concept of free-market choice of Ada is represented as the Natural fraction of Ada starts.

The Natural fraction of Ada starts (abbreviated Nat_fr_Ada_starts in the model) is a converter variable, which uses a graphical function to determine the natural fraction for any given value of Perceived incentives to use Ada.



Nat_fr_Ada_starts = graph(Perc_ince_use_Ada)
-1.000 -> 5.000e-3
-0.800 -> 0.025
-0.600 -> 0.055
-0.400 -> 0.120
-0.200 -> 0.260
0.0 -> 0.500
0.200 -> 0.680
0.400 -> 0.780
0.600 -> 0.825
0.800 -> 0.870
1.000 -> 0.910

As the Perceived incentives to use Ada increase, higher fractions of development projects will use Ada. Even when the Perceived incentives get to 1.0, indicating an overwhelming advantage for Ada and its infrastructure, there will still be special cases where a non-Ada language, quite possibly specially-developed, will be used to obtain some functionality not easily available with Ada.

This graphical function indicates that when the Perceived incentives to use Ada are 0.0, half of all development projects will use Ada. This is an assumption that the design of the Ada language succeeded in its goals: that its features support modern software engineering practices, especially for embedded systems, better than any established language, and about as well as is possible given the state of the practical art in languages. Although this is not the place for a thorough review of the merits of Ada, suffice it to say here that the more evidence on the subject that accumulates, the more it appears that the Ada design meets its design goals. Certainly, the three avionics experiments have been successful (DS&E 1985, Stanley 1985, and Suydam 1985), as well as several other projects more outside Ada's original application domain of embedded systems. Moreover, Ada is perceived as making an important contribution to improving software engineering (SE), and to an extent consistent with the ambivalent reception SE gets within "real" programming organizations (Rogers 1985, pg. 6)

The input to Natural fraction of Ada starts is Perceived incentives to use Ada, which is measured in "Incentive units." These are an artificial creation used as an intermediary between characteristics of the Ada infrastructure viz. non-Ada, and the decision on which language to use. Really, "Incentive units" have no meaning except that implicit in the response of Natural fraction of Ada starts to the incentives. Incentives of 0.0 are defined as neutral–a toss-up between Ada and some non-Ada language, so Nat_fr_Ada_starts equals 0.5. Perceived incentives of -0.5 represent a substantial disadvantage for Ada use, and consequently the Nat_fr_Ada_starts equals 0.115. Incentives of -1.0 represent more or less complete impossibility of usefully employing the Ada language at all; Nat_fr_Ada_starts equals 0.005. (It doesn't go to 0.0 for two reasons: first, to prevent division-by-zero difficulties in the equation for Target for Ada starts relative to natural fraction, and second, to represent "proof_of_principle" kinds of projects, especially by academics, that proceed despite their impracticality.)

On the other side of the coin, incentives of 0.5 represent substantial superiority of Ada over the body of non-Ada languages used for MCCR programming, and Nat_fr_Ada_starts equals 0.8025, i.e. roughly 80 percent. Perceived incentives to use Ada equal to 1.0 represents overwhelming superiority of Ada and its infrastructure, with a Nat_fr_Ada_starts of 0.91. The graphic function is somewhat asymmetrical, in that it is easier for the non-Ada languages to vanquish Ada (with incentives of -1.0) than it is for Ada to vanquish the non-Ada languages (with incentives of +1.0). The asymmetry arises from the aggregation of a multiplicity of languages into the non-Ada category. There can be many specialized languages with well-defined niches that will be very difficult for a general-purpose language like Ada to displace.

Perceived incentives to use Ada (Equation #1100)

Program and project managers want languages and programming support environments (PSEs) that are cost-effective (even if for no other reason than satisfying military customers who want cost effectiveness). Managers also want reliability: software is but a portion of overall systems development, and delays in software can mean delays in the entire program. Managers (in collaboration with their customers) will choose languages and PSEs that meet these goals.

However, choices of language and infrastructure are not guided by perfect information about the future, or even about the present; information, especially subjective judgements, reputations, and "scuttlebut" will always be outdated relative to the actual current state of affairs. One phenomenon that is holding back Ada acceptance is the lag in people's perceptions about Ada behind the actual fact. For example, (Rogers 1985) documents differences between perceptions of the number of members of Ada-related special-interest groups and the actual numbers. The perception, as opposed to the actual facts, of Ada and the Ada Programming Support Environment (APSE) usefulness is represented by the Perceived incentives to use Ada.

The Perceived incentives to use Ada (abbreviated Perc_ince_use_Ada in the model) is with a level variable, whose only rate, an inflow, is the Change in perceived incentives to use Ada.

□ Perc_ince_use_Ada = Perc_ince_use_Ada + Ch_per_ince
INIT(Perc_ince_use_Ada) = -2
{Perceived incentive to use Ada (incentive units)}

The meanings of particular numerical values of perceived incentives or incentives are discussed below.

The Perceived incentives to use Ada is initialized at -2.0, representing for 1975 a completely undesirable language. What contractor could program in a higher-order language (HOL) for which there was not only no compiler (and wouldn't be for several years), but also no defined syntax or semantics?

Change in perceived incentives to use Ada (Equation #1110)

Perceptions are changed by facts only gradually. A reputation, be it of a politician or a computer language, is built up of many individual anecdotes, scientific findings, personal experiences, philosophical biases, and more. Today's facts add only increments to the stew of yesterday's facts, the day before yesterday's facts, and so on. If Ada has the reputation of being too complex, hard to learn, and an inefficient user of computer resources, success stories like the F-15 and F-20 avionics experiments or compiler benchmarks will gradually chip away at that perception. The rate of "chipping away" of perceptions by current facts is represented by the Change in perceived incentives to use Ada.

The Change in perceived incentives to use Ada (abbreviated Ch_per_ince) is a rate variable, flowing into the Perceived incentives to use Ada.

○ Ch_per_ince = (Incentive_use_Ada - Perc_ince_use_Ada)
/Time_perc_ince
{Change in perceived incentives (to use Ada)
(incentive units/year)}

The overall formulation is the standard first-order lag (or first-order delay). The Change in perceived incentives to use Ada will increase perceived incentives if the actual incentives are greater than those perceived, and inversely. In this way, Perceived incentives to use Ada will always move toward the actual Incentives to use Ada. The speed of the motion is governed by the magnitude of the Time to perceive incentives to use Ada.

Because the time to perceive is in the denominator of the equation, a small Time to perceive means large changes in perception, i.e. perception will approach actuality in a small period of time. For more description of the properties of the general first-order delay, see (Forrester 1969, Sections 2.2 and 10.2); Goodman 1974, Chapter 3; or Alfeld and Graham 1976, Section 3.5)

Time to perceive incentives to use Ada (Equation #1120)

The perceptions about the incentives to use Ada will always be somewhat out of date. It takes time before breakthroughs become known, trusted, and acted upon throughout the software community. The length of this delay is represented by the Time to perceive incentives to use Ada.

Time to perceive incentives to use Ada (abbreviated Time_perc_ince in the model) is a converter variable with no inputs; i.e. a constant.

○ Time_perc_ince = 2 {Time to perceive incentives (to use Ada) (years)}

A value of two years indicates how long it take for perceptions to catch up with the actual incentives to use Ada, plus how long it takes to translate those perceptions into actual choice of Ada versus any of the non-Ada languages. Taking into account both delays in perception and delays in influencing choices, two years may be somewhat shorter than the true average delay.

Incentives to use Ada (Equation #1200) (an output)

When a contractor is choosing between Ada and a Non-Ada language, the criterion is effectiveness: What language will permit the lowest-cost, fastest, and most reliable programming? Programming managers and their DoD clients want to choose a language in which programmers can be most productive. There are considerations beyond simple cost. If the infrastructure exists to ensure coordination, a rapid "fan out" of programming tasks allows more programmers to work on a project at once, allowing completion in a shorter elapsed time, which means less difficulty integrating the programming with the rest of the development effort. Also, there are reliability considerations: managers need to count on programming being done to ensure success of the overall system being developed. Managers would like to see experienced programmers, software tools demonstrated to work well with the language, enough completed projects to have defined the pitfalls, and so on. Lack of infrastructure means reliability can't be demonstrated, which is a disincentive to use the language. In the model, all such considerations are combined into a single aggregate measure, the Incentives to use Ada. Figure A.6-2 shows the inputs to this variable.

Incentives to use Ada (abbreviated Incentive_use_Ada in the model) is a converter variable: it combines the incentives from coverage, infrastructure intensity, and policy into an aggregate incentive.

○ Incentive_use_Ada = Ince_rel_int_infr + Ince_rel_cov_infr + Ince_pol
{Incentives to use Ada (incentive units)}

The meaning of incentive units is discussed above in the description of the equation for Natural fraction of Ada starts.

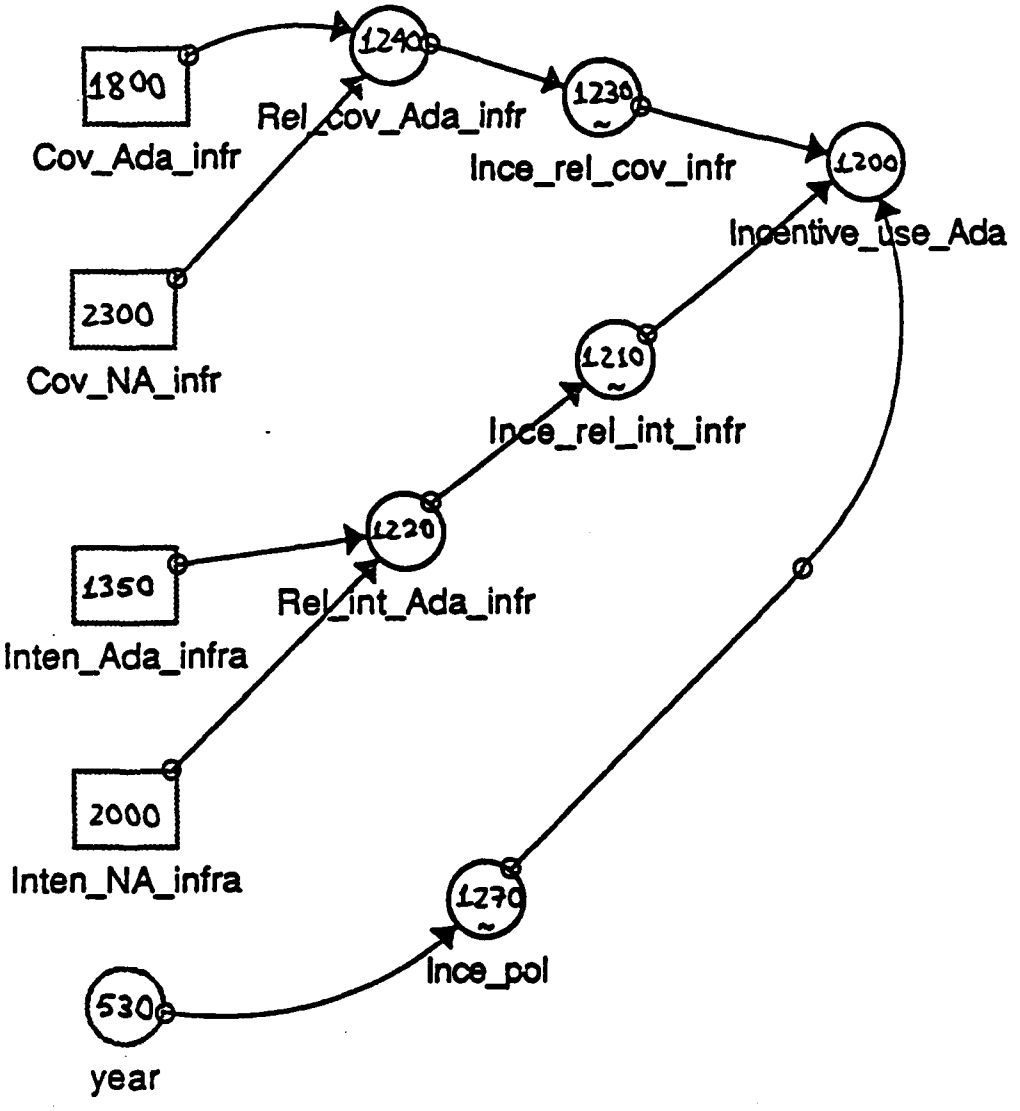
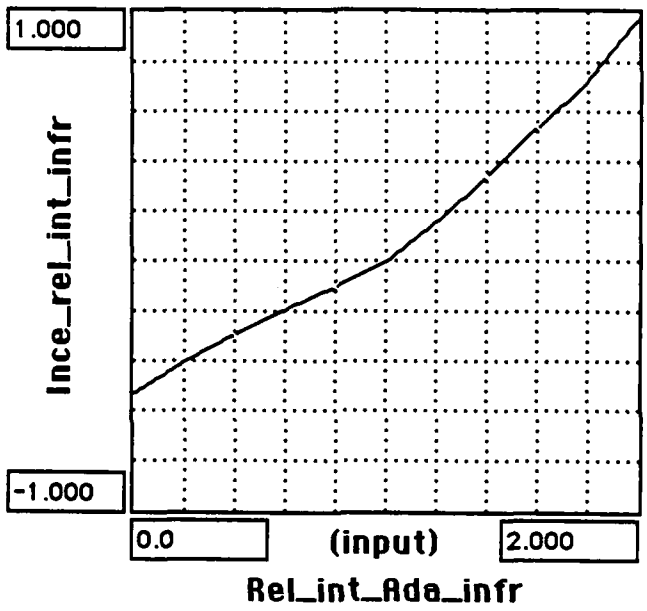


Figure A.6-2. Flow diagram of inputs to Incentives to use Ada (Incentive_use_Ada).

Incentive from intensity of infrastructure (Equation #1210)

As described above, the intensity of infrastructure of a language is a major factor in deciding whether to use it or some other language. People often speak of a "track record" (especially in the context of Ada not having one) but the root cause for having confidence in the cost-effectiveness of a language is not the track record per se, but whether compilers, experienced people, tools, libraries, and so on (which are by-products of that track record) can be brought to bear on the project. In other words, the more infrastructure is available, the more desirable a language and its infrastructure become.

Incentive from intensity of infrastructure (abbreviated Ince_rel_int_infr in the model) is a converter variable, converting the Relative intensity of Ada infrastructure into a measure of incentives.



Ince_rel_int_infr = graph(Rel_int_Ada_infr)
0.0 -> -0.530
0.200 -> -0.400
0.400 -> -0.290
0.600 -> -0.190
0.800 -> -0.100
1.000 -> 0.0
1.200 -> 0.160
1.400 -> 0.340
1.600 -> 0.540
1.800 -> 0.720
2.000 -> 0.970

Relative intensity of Ada infrastructure (Equation #1220)

Desirability of a language is a relative matter. Contractors can't choose the perfect language; they can only choose among existing languages and infrastructures (or those whose construction is straightforward and reliable). Therefore, the model compares the infrastructure intensity available with Ada to that available for the average of all non-Ada languages. The result of this comparison is the Relative intensity of Ada infrastructure

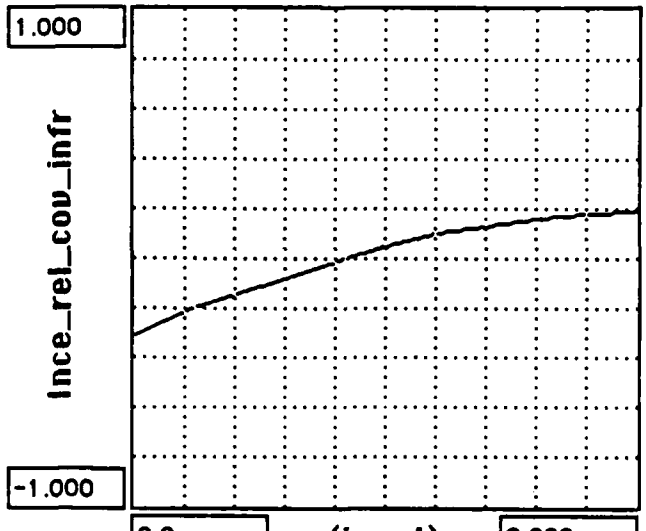
Relative intensity of Ada infrastructure (abbreviated Rel_int_Ada_infr in the model) is a converter variable, dividing the Intensity of Ada infrastructure by the Intensity of non-Ada infrastructure.

○ Rel_int_Ada_infr = Inten_Ada_infra/Inten_NA_infra
{Relative intensity of Ada infrastructure (dimensionless)}

Incentive from coverage of infrastructure (Equation #1230)

Contractors are often constrained as to what host and especially what target machines are to be used for a given project. If Ada infrastructure is not available in sufficient quantity for a given host/target combination, Ada is not desirable for that project. In the aggregate, the narrower the coverage of the infrastructure, the fewer projects will be able to use Ada, and the smaller will be the fraction that use it. The measure of the desirability of Ada based on the relative coverage is called Incentive from coverage of infrastructure.

Incentive from coverage of infrastructure (abbreviated Ince_rel_cov_infr in the model) is a converter variable, using a graphic function to convert the Relative coverage of Ada infrastructure into an incentive.



⊘ Ince_rel_cov_infr = graph(Rel_cov_Ada_infr)
0.0 -> -0.310
0.200 -> -0.215
0.400 -> -0.140
0.600 -> -0.080
0.800 -> -0.015
1.000 -> 0.050
1.200 -> 0.095
1.400 -> 0.125
1.600 -> 0.160
1.800 -> 0.180
2.000 -> 0.190

Using the idea that incentives can represent the issues around coverage may seem odd at first. But lack of coverage just means that there is cost and delay involved in porting the language and environment to the desired target and host. The smaller the coverage, the more projects face such costs if they use Ada. In the aggregate, then, low coverage can be translated to a cost and therefore to an incentive.

Relative coverage of Ada infrastructure (Equation #1240)

In the aggregate, coverage of infrastructure, like intensity, is a relative matter. While Ada and APSE may not be available for all potential target machines, neither is UNIX, or DEC's VMS, or IBM's MVS. Incentives from coverage to use Ada can only come from comparison of the costs of retargeting or rehosting (or needing to change to a software-compatible target or host) incurred for Ada to the corresponding costs for an average non-Ada language. For the non-Ada languages, there is also the opportunity to trade off such costs against the cost of switching to another non-Ada language whose infrastructure is present for the desired host and target. In the model, the aggregate result of such comparisons is represented by the Relative coverage of Ada infrastructure.

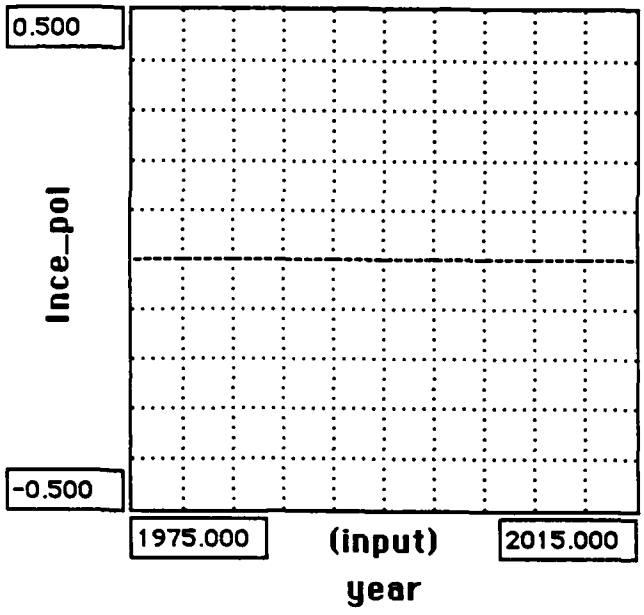
The Relative coverage of Ada infrastructure (Abbreviated Rel_cov_Ada_infr in the model) is a converter variable, converting the Coverage of Ada infrastructure and the Coverage of non-Ada infrastructure into their quotient.

○ Rel_cov_Ada_infr = Cov_Ada_infr/Cov_NA_infr
{Relative coverage of Ada infrastructure (dimensionless)}

Incentive from policy (Equation #1270)

There could be many factors other than the relative intensity of infrastructure and the relative coverage that could influence a project manager's language choice. The language of contracts might be written with incentives to use Ada. The Ada Joint Program Office (AJPO) might sponsor marketing actions -- seminars, journals, advertisements, giveaways, and so on. For the most part, these are actions at a level of detail considerably finer than those the model is able to represent. But even without the ability to explicitly model such activities, there is a way to represent their effects, in order to simulate their impact on acceptance and find out how valuable that might be. The policy lever in the model to be used for such experiments is the Incentive from policy.

Incentive from policy (abbreviated Ince_pol in the model) is a converter variable, which uses a graphic function to convert the calendar year into an incentive.



Ince_pol = graph(year)
1975.000 -> 0.0
1979.000 -> 0.0
1983.000 -> 0.0
1987.000 -> 0.0
1991.000 -> 0.0
1995.000 -> 0.0
1999.000 -> 0.0
2003.000 -> 0.0
2007.000 -> 0.0
2011.000 -> 0.0
2015.000 -> 0.0

The Incentive from policy is an exogenous variable, i.e. one not influenced by any other system variables. Its value for the base scenario is always neutral, set at 0.0.

Appendix A.7: Ada Infrastructure Sector

The Ada infrastructure sector represents the entire ensemble of things used to carry on programming in Ada. As discussed in Section 5 of this report, this ranges from the most basic software tools (e.g., an Ada compiler itself, linkers, loaders, etc.) to advanced productivity tools (e.g., program libraries and configuration managment tools) to hardware to speed programming (such as time-sharing and screen terminals), to the "soft" infrastructure of programmer's experience, programming management, courses of instruction and instructional literature. The dimensions used to characterize this diverse collection are: intensity (on average, how much of it is available?), incompatibility (how many incompatible environments are there?), and coverage (what fraction of host/target configurations are one of the environments available for?)

The characteristics of the Ada infrastructure affect both the cost of programming in Ada and the extent to which it is used in MCCR programming. The cost sector represents the ability of intensity of Ada infrastructure to increase programmer productivity and thereby reduce software costs. That sector also represents incompatibility of infrastructure (even Ada can have non-linguistic incompatibilities) increasing costs. The language choice sector represents the impact of such cost/effectiveness considerations on the frequency of Ada use -- the extent to which it is used in non-embedded computer applications, and the extent to which waivers for use of a non-Ada language are obtained for embedded applications.

Intensity of Ada infrastructure (Equation #1350)

The infrastructure supporting Ada has many components and characteristics; some of which are so obvious that they are taken for granted. For example, does a compiler exist? How many programmers, textbooks, courses, etc. are there? How sophisticated is the APSE; and how many programming tools are available? As these components of the infrastructure develop and become widely available, programming becomes more efficient. All of these components of infrastructure are combined and represent in the model by the Intensity of Ada infrastructure. Figure A.7-1 shows the inputs.

Intensity of Ada infrastructure (abbreviated Inten_Ada_infra in the model) is a level equation. It accummulates the rates of flow into it, Creation of intensity of Ada infrastructure and Injection of government furnished Ada infrastructure. It is depleted by its outflow rate, Obsolesence of intensity of Ada infrastructure.

☐ Inten_Ada_infra = Inten_Ada_infra + Crea_int_Ada_infr -
Obsol_int_Ada_inf + Inj_GFE_Ada_infr
INIT(Inten_Ada_infra) = 0
{Intensity of Ada infrastructure (intensity units)}

Obsolescence of intensity of Ada infrastructure (Equation #1360)

Ada infrastructure, like all others, gradually and continuously obsolesces. Some of this obsolescence represents the actual outdating and discarding of out-of-date textbooks, tools, implementations, etc. Obsolesence also occurs when expert programmers retire or move on to different endeavers they take with them knowledge and experience that must be replaced or infrastructure will decline. Both kinds of loss are represented in the aggregate by the Obsolescence of intensity of Ada infrastructure.

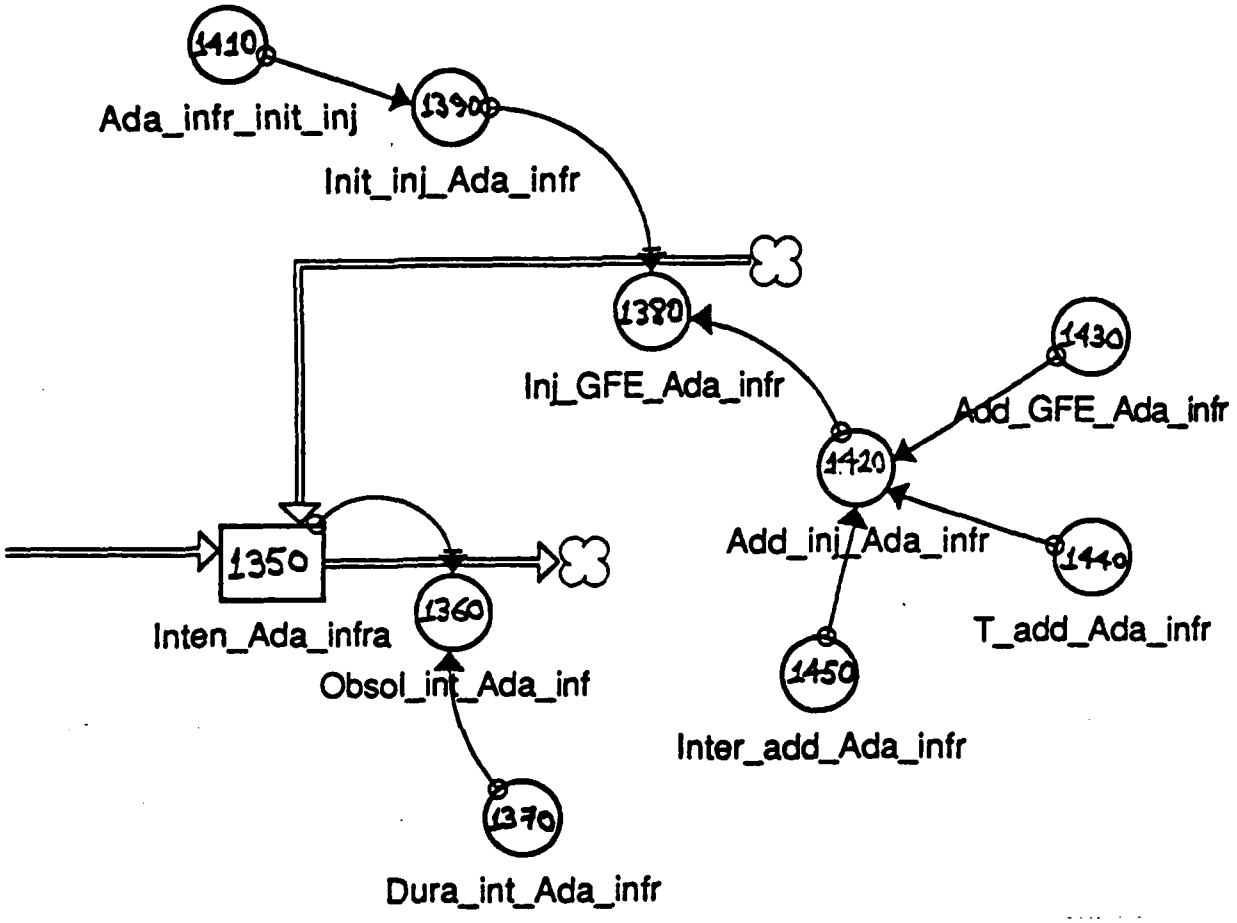


Figure A.7-1. Flow diagram of inputs to Intensity of Ada infrastructure (Inten_Ada_infr). The flow entering from the left is the Creation of intensity of Ada infrastructure.

Obsolescence of intensity of Ada infrastructure (abbreviated Obsol_int_Ada_inf in the model) is a rate equation that depletes the level of Intensity of Ada infrastructure.

○ Obsol_int_Ada_inf = Inten_Ada_infra/Dura_int_Ada_infr
{Obsolesence of intensity of Ada infrastructure (units/year)}

The equation for obsolescence is the familiar first-order delay, where, if there is no inflow, the contents of the level will fall off exponentially, like radioactive decay. When a time equal to the time constant (the Duration of intensity of Ada infrastructure) has passed, approximately 30 percent of the initial value of the level will remain.

Duration of intensity of Ada infrastructure (Equation #1370)

Every component of Ada infrastructure has a finite useful lifetime. For textbooks it may be a decade, for programmers it may be one or two decades before they join management, retire, or seek other professions. For software tools, the useful lifetime may vary considerably. Only a year may pass before some software tools are replaced by newer tools. At the other extreme, some compilers and editors endure for one or two decades with only minor maintenance changes. Standards for infrastructure endure for even longer; the fundamental structures of the FORTRAN and COBOL languages have endured from the early 1950s, with no sign of imminent demise. The average for all the components of Ada infrastructure is here called Duration of intensity of Ada infrastructure.

Duration of intensity of Ada infrastructure (abbreviated Dura_int_Ada_infr in the model) is a converter equation with no inputs, i.e., a constant.

○ Dura_int_Ada_infr = 30 {Duration of intensity of Ada infrastructure (years)}

The duration of intensity is coupled to the total duration of programs during development and maintenance, which, at 10 and 20 years respectively, sum to 30 years. The assumption is that only a fraction of the components of the infrastructure will obsolesce before the programs themselves, especially the programmer's skills. Another fraction of the infrastructure must be maintained as long as the programs are being maintained; this includes operating systems, test sets, and other software tools without which the programs cannot be maintained. The final fraction of the infrastructure will outlive the programs; for instance, this fraction includes styles and procedures of programming (assembly language, FORTRAN, or ALGOL-style programming, standards (ASCII and all the editors that work on ASCII files)), and operating systems (the IBM 360/370 and DEC PDP-11 are going on twenty years and promise another 20 at least).

In the absence of more specific data, the duration is set equal to the duration of programs, 30 years.

Injection of GFE'd Ada infrastructure (Equation #1380)

The Injection of GFE'd Ada infrastructure represents a somewhat broader group of actions than might be indicated by the normal meaning of Government-Furnished Equipment (GFE). This variable is the conduit for representing several differing additions to Ada infrastructure. There is an initial surge of infrastructure created by the validation of compilers as Ada. This surge was certainly done at the behest of DoD, although the compilers are not literally GFE. There is a policy option to create more infrastructure after

the initial surge. That creation could represent literally a GFE'd environment. Alternatively, such a creation could represent the increase in infrastructure available to Ada programmers if SVID were adopted as an interim tool interface, such that UNIX tools suddenly became available and officially acceptable for Ada programming.

Injection of government furnished Ada infrastructure (abbreviated Inj_GFE_Ada_infr in the model) is a rate equation that increases the level Intensity of Ada infrastructure.

○ Inj_GFE_Ada_infr = Init_inj_Ada_infr + Add_inj_Ada_infr
{Injection of GFE (government-furnished equipment) for Ada infrastructure (infrastructure units/year)}

Initial injection of Ada infrastructure (Equation #1390)

Project work in the official Ada language could not begin until the very first Ada compilers were validated in 1983. The structure and design of Ada had been discussed in the computer literature for years. Experience programming in other languages carried over into Ada to some extent. Combined with these precedents, the validation of compilers sharply created an available Ada infrastructure. In the model this is represented by the Initial injection of Ada infrastructure.

Initial injection of Ada infrastructure (abbreviated Init_inj_Ada_infr in the model) is a converter equation which uses the special STELLA function called PULSE to covert the calendar year into a pulse in the rate of flow into the level of Intensity of Ada infrastructure.

○ Init_inj_Ada_infr = PULSE(Ada_infr_init_inj,1981,1e11)
{Initial injection of Ada infrastructure (infrastructure units/year); the one-time construction of a few initial compilers, loaders, etc.}

The pulse function introduces a brief (one time unit) increment to a variable. It has three arguments; the first determines the size of the increment, the second indicates the time of the first pulse, and the third specifies the interval of any subsequent repititions of the pulse. By choosing 1e11 (i.e. the interval is 100 billion years) here, we do not get any repetitions within the time span of the model's policy runs. The 1981 injection date is somewhat in error, but it should be noted that there was work going on with nonvalidated compilers and so on before the first official validations.

Ada infrastructure initially injected (Equation #1410)

The introduction of the first approved Ada compilers was the first significant and substantial increase to the intensity of Ada infrastructure. This boost is called Ada infrastructure inititial injection.

Ada infrastructure initial injection (abbreviated Init_inj _Ada_infr in the model) is a converter variable with no inputs, i.e., a constant.

○ Ada_infr_init_inj = 20
{Ada infrastructure inititially injected (infrastructure units)}

The value of 20 assumes that compilers and their closely-associated tools (linkers, loaders, perhaps debuggers) represent about half of the current infrastructure currently used

to support non-Ada programming, which is arbitrarily defined as intensity equals 40. See Section 5 for further discussion of infrastructure units.

Additional injection of Ada infrastructure (Equation #1420)

If the DoD adopts SVID, or a small number of commercial operating systems as an interim standard, this will represent a second major boost to the available infrastructure intensity. In the model, any such additional boosts after the validation of the compilers is represented with the Additional injection of Ada infrastructure.

Additional injection of Ada infrastructure is a converter variable that uses a special purpose PULSE function to add brief periodic increments to infrastructure.

○ Add_inj_Ada_infr = PULSE(Add_GFE_Ada_infr,T_add_Ada_infr,
Inter_add_Ada_infr)
{Additional inection of Ada infrastructure (infrastructure units/year)}

The three arguments specify the size of the increment, the time of the first injection, and the time interval between injections.

Additional GFE'd Ada infrastructure (Equation #1430)

The relative size of the injection by the government is called Additional GFE'd Ada infrastructure.

Additional GFE'd Ada infrastructure (abbreviated Add_GFE_Ada_infr in the model) is a converter variable with no inputs, i.e., a constant.

○ Add_GFE_Ada_infr = 0
{Additional GFE'd Ada infrastructure (infrastructure units)}

Time for additional Ada infrastructure (Equation #1440)

Time for additional Ada infrastructure determines when the injection of additional infrastructure intensity will take place.

Time for additional Ada infrastructure (abbreviated T_add_Ada_infr in the model) is a converter variable with no inputs, i.e., a constant.

○ T_add_Ada_infr = 1990
{Time for additional Ada infrastructure (year)}

1990 is a plausible, albeit somewhat optimistic, date for adopting a standard CAIS. ((KITIA 1985 even suggests fourth quarter 1988, but the schedule leading up to that date has already slipped by several months as of February 1986.) In the scenario testing other times might be be chosen.

Interval to add Ada infrastructure (Equation #1450)

To explore the possibility that the government might adopt a strategy of successive injections of Ada infrastructure in order to accelerate its evolution, the model has a variable called Interval to add Ada infrastructure.

Interval to add Ada infrastructure (abbreviated Inter_add_Ada_infr in the model) is a converter variable without an input, i.e., a constant.

○ Inter_add_Ada_infr = 1e11 {Interval to add Ada infrastructure (years)}

The enormous default value of 1e11 effectively turns this facility off.

Creation of intensity of Ada infrastructure (Equation #1500)

The creation of intensity of infrastructure is primarily a free-market process. A considerable amount of Ada compiler development, for example, has occurred without direct government funding. Corporations have allocated funds because of the perceived future benefits of having Ada products. This has happened both in large corporations preparing for Ada contract work, and in small corporations looking to sell Ada products in the marketplace. Creation of infrastructure is influenced by many factors. It can be accelerated by increased use of Ada, by more advanced competing Non-Ada infrastructures that can be borrowed from, by perceptions that is a potentitially profitable language to develop tools for, and by policy declarations. It can be retarded by incompatibility within the Ada infrastructure, lack of use of Ada, perception by tool developers that Ada is an inferior language, and by policy declarations. All of these effects are combined in Creation of intensity of Ada infrastructure. Figure A.7-2 shows a flow diagram of the inputs.

Creation of intensity of Ada infrastructure (abbreviated Crea_int_Ada_infr in the model) is a rate equation that increases the level Intensity of Ada infrastructure.

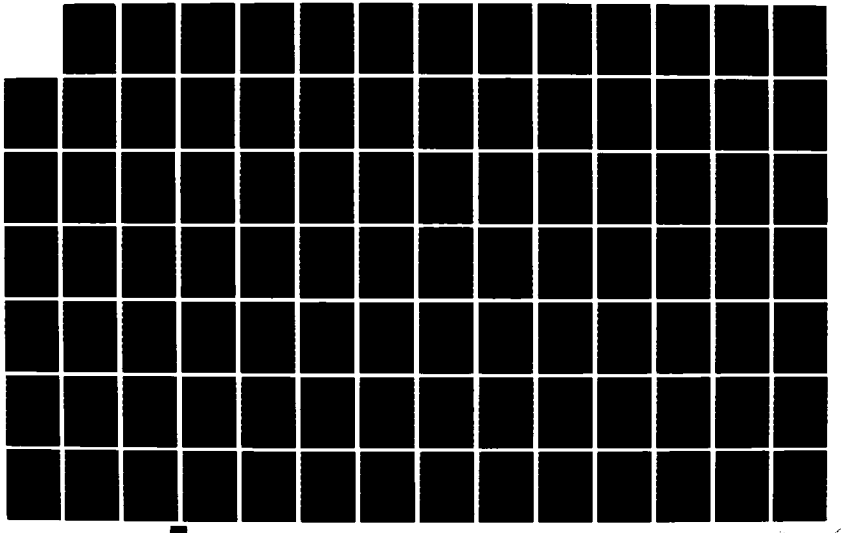
○ Crea_int_Ada_infr = Norm_cr_int_A_infr*E_tech_cr_int*
E_inc_int_Ada_infr*E_use_int_A_infr*E_inco_int_A_infr *
E_pol_int_A_infr*E_rel_infr_int_Ada
{Creation. of intens. of Ada infrastr. (infr. units/yr.)}

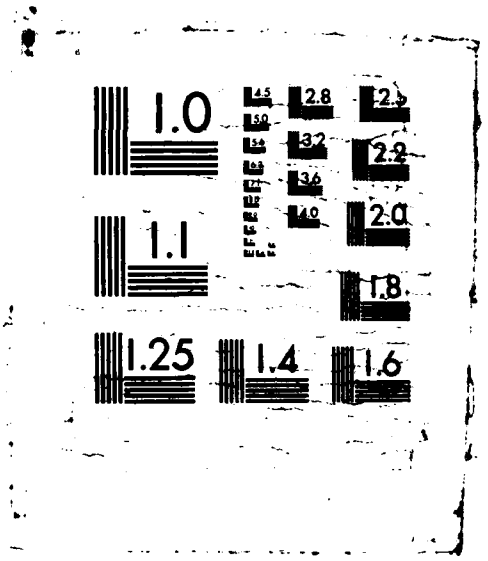
This creation represents both the development of new kinds of infrastructure (i.e., creation of a never-before-used tool) and the replacement of infrastructure lost through normal obsolesence (i.e., replacing retiring programmers.)

The separate effects are multiplied rather than added. A zero in any component is enough to overwhelm all the other effects even if they are high. For example, if for some hypothetical reason absolutely no one used Ada, there would be no more creation of infrastructure even if the infrastructure had low incompatibility and advanced infrastructure. The equation format above, with a normal constant multiplied by several variables centered around unity is useful and common; see (Alfeld and Graham 1976, Section 5.3) or (Richardson and Pugh 1981, pp. 152-56) for discussion.

AD-A175 352 COST EFFECTIVENESS TRADEOFFS IN COMPUTER STANDARDIZATION AND TECHNOLOGY I..(U) INSTITUTE FOR DEFENSE ANALYSES ALEXANDRIA VA A A HOOK ET AL. JUN 86 2/4

UNCLASSIFIED IDA-P-1931 IDA/HQ-86-31052 MDA903-84-C-0031 F/G 9/2 NL




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0

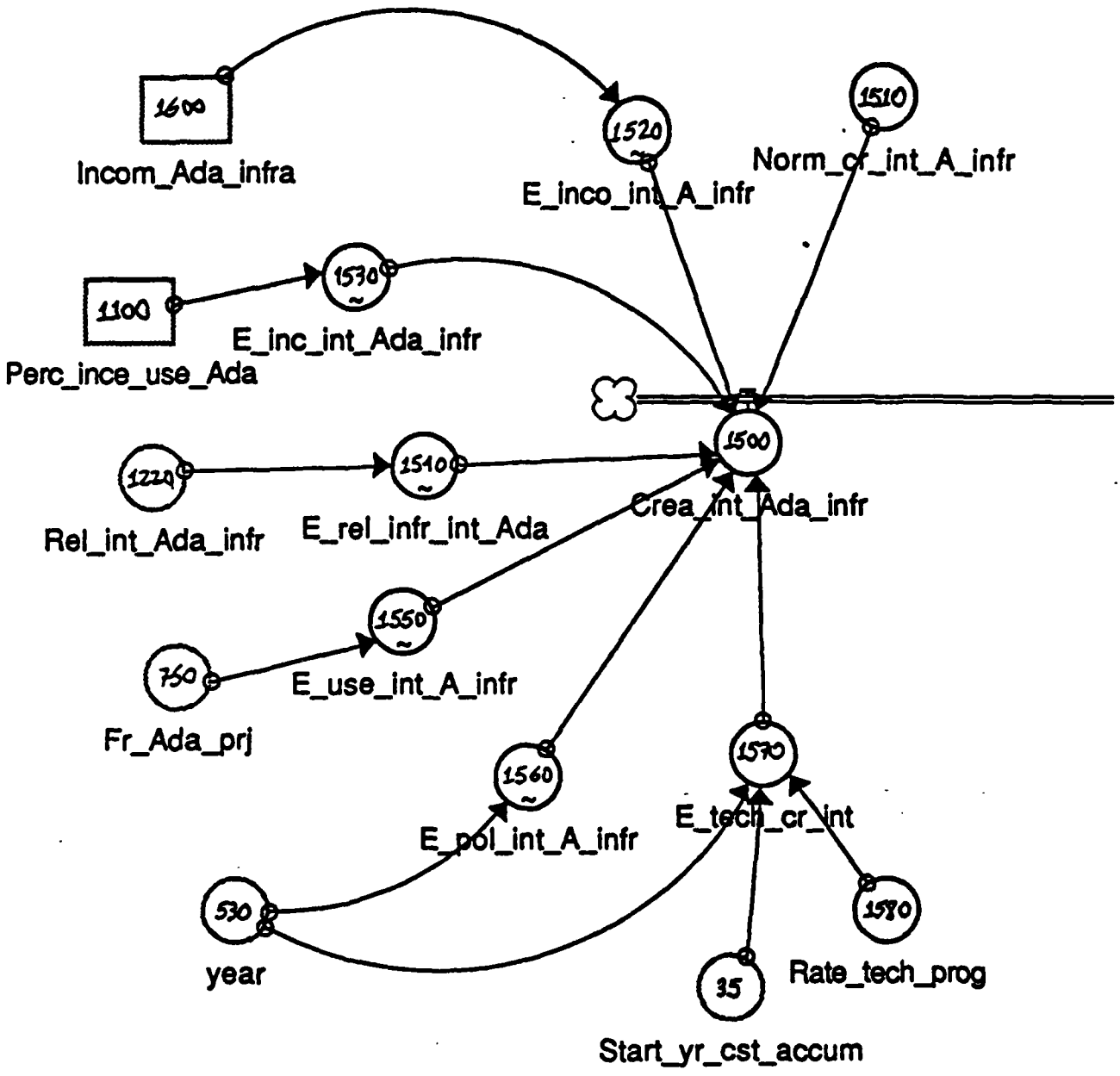


Figure A.7-2. Flow diagram of inputs to Creation of intensity of Ada infrastructure (Crea_int_Ada_infr). The flow exiting to the right goes into Intensity of Ada infrastructure.

Normal creation of intensity of Ada infrastructure (Equation #1510)

In processes which are influenced by many factors it is often convenient express the functional relationship as the product of two terms: a normal base rate of activity (representing what the output value would be if all inputs were at some specified reference values), and a term giving the effects of the inputs not being at their reference values. Here the Normal creation of intensity of Ada infrastructure represents the rate of adding to infrastructure when the other factors influencing that rate are respectively equal to their reference values.

Normal creation of intensity of Ada infrastructure (abbreviated Norm_cr_int_A_infr in the model) is a converter variable with no inputs, i.e., a constant.

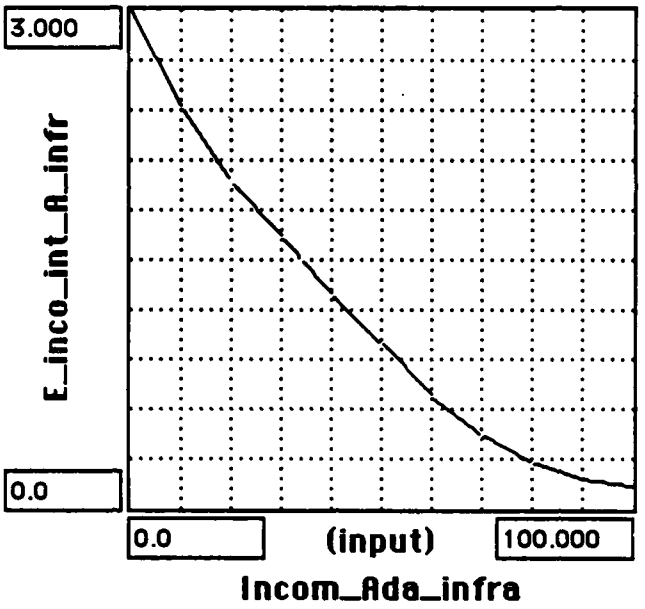
○ Norm_cr_int_A_infr = 2.4
{Normal creation of intensity of Ada infrastructure (intensity units/year)}

The value was chosen on the basis of having a proper relation to the corresponding parameter for non-Ada, and to give a buildup of infrastructure over a plausible time horizon. Appendix A.9 on multivariable model calibration provides details of the parameter estimation.

Effect of incompatibility on intensity of Ada infrastructure (Equation #1520)

The more incompatible infrastructures there are, the smaller the incentives to create more infrastructure for any one in particular. Incompatible infrastructures divide the total market for programming environments into incompatible segments, which reduces the market size for any particular piece of infrastructure, be it a tool, a program library, a programmer, or whatever. For example, building tools based on the Pick operating system for personal computers is a poor investment; there aren't that many Pick OS's being used. Similarly, building tools for JOVIAL environments, all other factors being equal, would be less profitable than building tools for a single, transportable Ada environment used in all DoD work. In the model, such phenomena are represented by the Effect of incompatibility on intensity of Ada infrastructure.

Effect of incompatiblility on intensity of Ada infrastructure (abbreviated E_inco_int_A_infr in the model) is a converter variable. It uses a graphic function to convert a given level of Incompatibility of Ada infrastructure into a multiplicative effect on the Creation of intensity of Ada infrastructure.

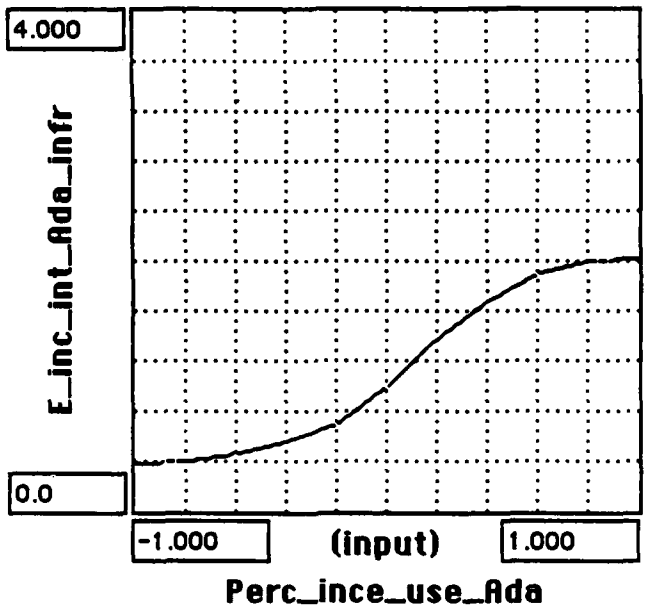


E_inco_int_A_infr = graph(Incom_Ada_infra)
0.0 -> 3.000
10.000 -> 2.415
20.000 -> 1.965
30.000 -> 1.635
40.000 -> 1.290
50.000 -> 1.000
60.000 -> 0.675
70.000 -> 0.450
80.000 -> 0.285
90.000 -> 0.180
100.000 -> 0.135

Effect of incentives on intensity of Ada infrastructure (Equation #1530)

One motivation for creating Ada infrastructure is the perception of incentives for using Ada, even if it isn't yet in widespread use. Proof-of-concept programming projects can create showy successes. APSEs can be created with "technically sweet" tools that promise great efficiencies in use. If the software community begins to perceive Ada as the wave of the future because such incentives exist for its use, companies will invest in tool development even before Ada is being used widely. Such effects are represented by the Effect of incentives on intensity of Ada infrastructure.

Effect of incentives on intensity of Ada infrastructure (abbreviated E_inc_int_Ada_infr in the model) is a converter variable that uses a graphic function to translate Perceived incentives to use Ada into a multiplicative effect on the Creation of intensity of Ada infrastructure.

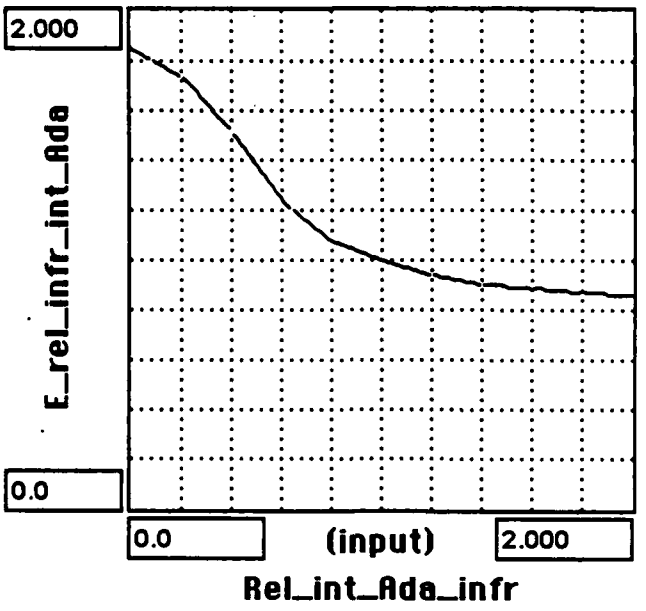


E_inc_int_Ada_infr = graph(Perc_ince_use_Ada)
-1.000 -> 0.380
-0.800 -> 0.400
-0.600 -> 0.460
-0.400 -> 0.560
-0.200 -> 0.700
0.0 -> 1.000
0.200 -> 1.380
0.400 -> 1.680
0.600 -> 1.900
0.800 -> 2.000
1.000 -> 2.020

Effect of relative infrastructure on intensity of Ada infrastructure (Equation #1540)

Software developers usually try to build on past successes, whether their own or that of others. Developers of Ada infrastructure will attempt to incorporate features, tools, programming styles, etc. of any Non-Ada infrastructure that develops that is superior to what is available, at the time, for Ada. Indeed, for example, the reason that so many of the current crop of Ada compilers come with debuggers is because of the demonstrated usefulness of debuggers in past non-Ada programming. In the model, this technology tranfer is called Effect of relative intensity of Ada infrastructure.

Effect of relative infrastructure on intensity of Ada infrastructure (abbreviated E_rel_inf_A_int in the model) is a converter variable, which uses a graphic function to translate any given value of the Relative intensity of Ada infrastructure into the approproate multiplier effect on the Creation of Ada infrastructure.



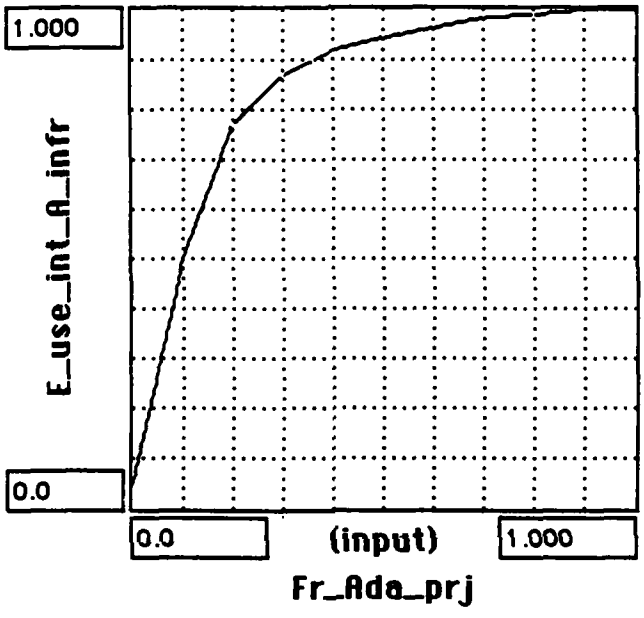
E_rel_infr_int_Ada = graph(Rel_int_Ada_infr)
0.0 -> 1.850
0.200 -> 1.740
0.400 -> 1.510
0.600 -> 1.240
0.800 -> 1.080
1.000 -> 1.000
1.200 -> 0.940
1.400 -> 0.900
1.600 -> 0.890
1.800 -> 0.870
2.000 -> 0.860

Effect of relative use on intensity of Ada infrastructure (Equation #1550)

Ada infrastructure is created by Ada use. Program libraries are continually added to in the course of use. Programmer's experience accumulates by virtue of programming activity in Ada. Steady use justifies acquisition of hardware to aid programming such as CRT terminals and sufficient system resources to guarantee quick system response time. Moreover, one can look at the creation of software tools as an economic market driven by perceptions of opportunities for profit. The opportunities to market Ada tools or whole environments depend very much on what the future of Ada use is assessed to be. One of

the most persuasive pieces of evidence about the extent to which Ada will be used in the future is the extent to which Ada is used in the present. Here is half of a "bandwagon effect" where use creates infrastructure. In the model this is represented by the Effect of use on intensity of Ada infrastructure.

Effect of use on intensity of Ada infrastructure (abbreviated E_use_int_Ada_infr in the model) is a converter variable that uses a graphic function to translate any given value of the Fraction of Ada projects into the appropriate multiplier effect on the Creation Ada infrastructure.

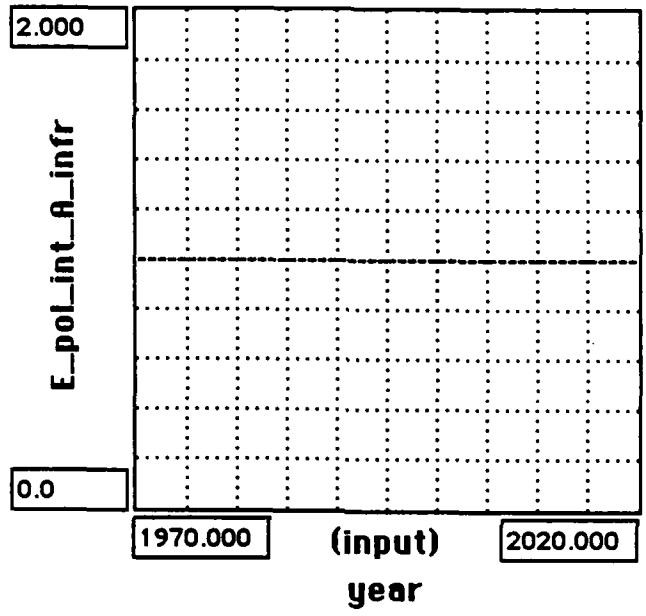


E_use_int_A_infr = graph(Fr_Ada_prj)
0.0 -> 0.050
0.100 -> 0.500
0.200 -> 0.770
0.300 -> 0.870
0.400 -> 0.920
0.500 -> 0.945
0.600 -> 0.965
0.700 -> 0.985
0.800 -> 0.990
0.900 -> 1.000
1.000 -> 1.000

Effect of policy on intensity of Ada infrastructure (Equation #1560)

In addition to all the free market effects discussed so far, there could be effects on the rate of creation of Ada infrastructure by policy directives, guidelines, mandates, marketing campaigns, etc. by the DoD. These effects can be portrayed with the variable Effect of policy on intensity of Ada infrastructure.

Effect of policy on intensity of Ada infrastructure (abbreviated E_pol_int_A_infr in the model) is a converter variable that uses a graphic function with year as an input. This policy facility allows one to specify the magnitude of the effect for any given year.



E_pol_int_A_infr = graph(year)
1970.000 -> 1.000
1975.000 -> 1.000
1980.000 -> 1.000
1985.000 -> 1.000
1990.000 -> 1.000
1995.000 -> 1.000
2000.000 -> 1.000
2005.000 -> 1.000
2010.000 -> 1.000
2015.000 -> 1.000
2020.000 -> 1.000

The default values of this graphic function depict a totally neutral policy condition. The use of this variable to represent policies is not clear at present; one use considered is part of representing the evolution of a CAIS specification: the hypothesis is that software

developers would defer heavy investment in Ada software tools if they knew that a standardized environment would be required and widely used a few years down the road—who wants their new software tools partially obsoleted by a change in operating system?

Effect of technology on creation of intensity (#1570)

Things are happening that will make it easier to accumulate infrastructure, which are not in turn controlled by events in DoD programming, infrastructure, and standards policies. Hardware is getting cheaper, faster, and possessing more memory all the time. The tradeoff of programmer time versus hardware expense is steadily tilting toward supplying more equipment to aid programmers. As time passes, it becomes easier to pile up workstations and time-shared screen-based terminals, user-friendly facilities, and tools. (Not only do tools cost less to buy or develop, they consume hardware resources; the cheaper those resources are, the more tools it is cost-effective to acquire.)

But hardware is not the only "external" to be changing, for fundamental conceptual insights into programming also steadily make it easier to accumulate infrastructure. For example, structured programming is a conceptual discovery, first enunciated forcefully by Djikstra (see Dijkstra 1969). By itself, that idea had virtually no impact. But as people learned how to program in a structured manner, an element of infrastructure was accumulating that allowed at least the "trendy" programmers that kept up with academic ideas to program more effectively. Gradually, languages evolved that explicitly supported structured programming, adding another element to the infrastructure. With such languages available, regular 9 to 5 programmers gradually began to learn the style of structured programming. Indeed, one of the functions of the Ada language is to make several of such conceptual advances available and practical for the average programmer. Of course, structured programming is but one of many conceptual advances that made it easier to pile up infrastructure, both in programmer's experiences and skills, and features built in to software. In the model, all the technological changes, including both the conceptual and the hardware, that increase the ability to create infrstructure is represented by the Effect of technology on the creation of intensity.

The Effect of technology on creation of intensity (abbreviated E_tech_cr_int in the model) is a converter variable, converting the calendar year into an effect.

○ E_tech_cr_int = EXP((year - Start_yr_cst_accum) * Rate_tech_prog) {
Effect of technology on creation of

The effect is a simple exponential, with a time constant of the Rate of technological progress. Because many of the model parameters are calibrated around a reference year in the middle of the simulated period (as opposed to the beginning), the effect is defined so that it reaches 1.0 at the Starting time for cost accumulation. As explained in the cost sector, the model therefore begins accumulating Total cost at the Starting time for cost accumulation. That time is assumed to be the time that the analysis is taking place, which is therefore the appropriate point about which to define reference values, such as the Effect of technology on creation of intensity.

Rate of technological progress (#1580)

The preceding paragraphs argue that as time passes, events and discoveries outside the sphere of DoD programming projects make it easier to accumulate intensity of infrastructure. Given that the level of detail of the model (and our lack of foreknowledge about future advances) prohibits an event-by-event description, the simplifying assumption is that each year, the creation of infrastructure under a given set of circumstances is some small fraction greater than it would have been the year before. In the model this fraction is the Rate of technological progress.

The Rate of technological progress (abbreviated Rate_tech_progress in the model) is a converter variable with no inputs, i.e., a constant.

○ Rate_tech_prog = 0
{Rate of technological progress (fraction/year)}

The Rate of technological progress was set to 0.0 to simplify the analysis of model behavior as the model was being developed. As the model calibration section discusses, parameters should be set during policy analysis at their most likely values, rather than attempting to guess which direction to bias the parameter to be conservative. Here, then, a non-zero setting for the Rate of technological progress is indicated. Over the last 200 years, the nation as a whole has increased the productivity of a given amount of capital and labor about 2 percent per year. While the quickly-moving computer field may seem as if the fraction should be much higher, there is a perception that the spread of extensive hardware support or new programming concepts has been quite slow, on the order of decades. Although these are not direct indicators of ease of accumulating infrastructure, they are related. Perhaps a figure of 3.0 percent per year would be plausible. In twenty years, without compounding, that rate would increase the normal creation of infrastructure 0.03 x 20 = 0.6, or 60 percent, which may even be on the high side.

Incompatibility of Ada infrastructure (Equation #1600)

An inevitable by-product of developing tools on different hardware, and broader application areas is the creation of incompatibility within the infrastructure. An environment written for a VAX will not run on an IBM computer without extensive adaptation. Students who learn Ada in classes on microcomputers will need some reorientation when they get jobs that require working on mainframes. When Ada gains in popularity and is used in broader application areas, new problems may arise that require extension, or specialized program libraries, all of which will introduce incompatibility into the infrastructure. In aggregate, the net effect of these incompatibilities in operating systems, style of use, program libraries, run-time packages, and so on is to reduce programmer efficiency. (See Section 5 for more detailed discussion.) In the model, these incompatibilities are characterized in the aggregate by the Incompatibility of Ada infrastructure. Figure A.7-3 shows a flow diagram of the inputs.

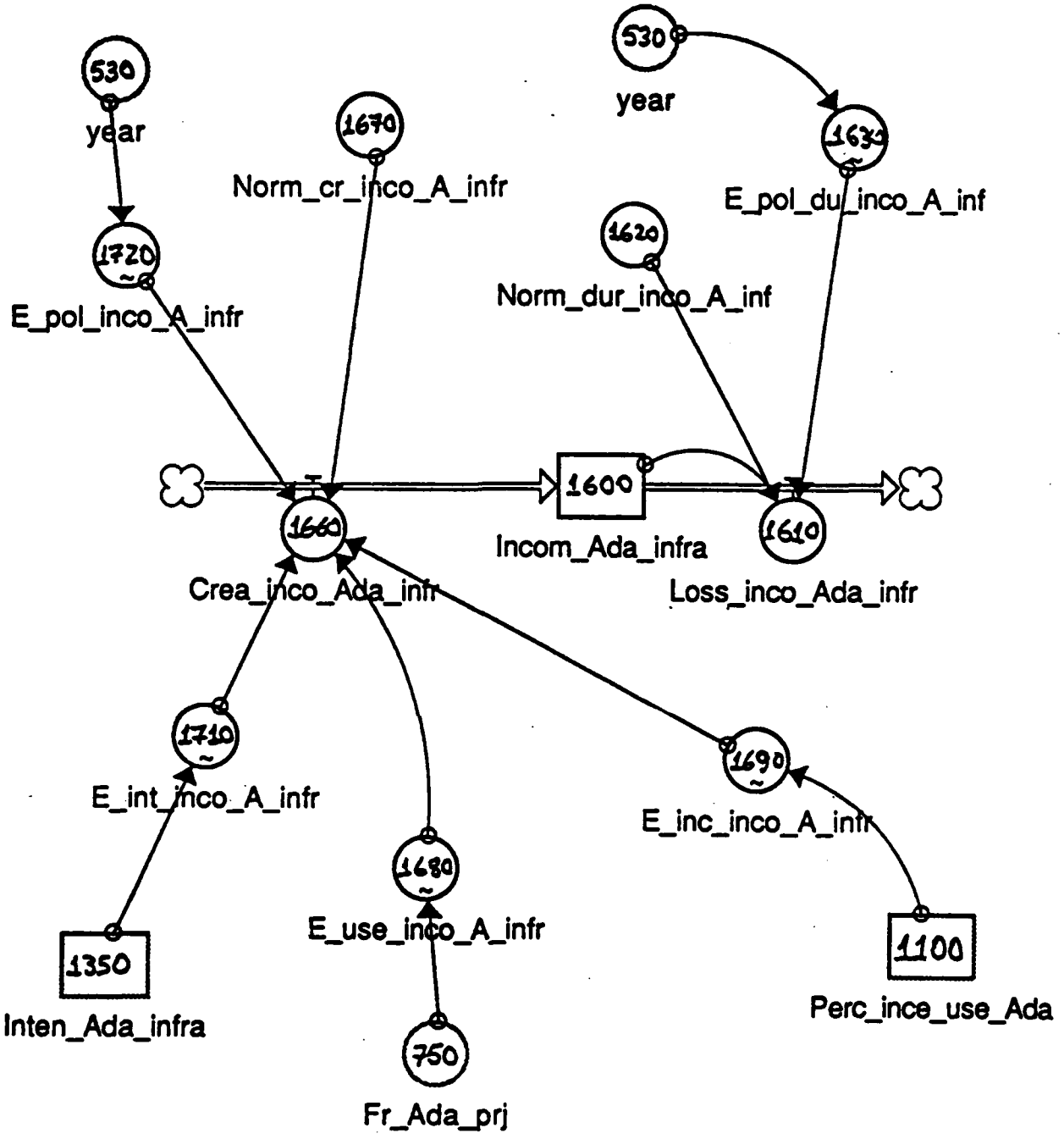


Figure A.7-3. Flow diagram of inputs to Incompatibility of Ada infrastructure (Incom_Ada_infra).

Incompatibility of Ada infrastructure (abbreviated Incom_Ada_infra in the model) is a level variable, which accumulates the inflow rate Creation of incompatible Ada infrastructure and is decreased be the outflow rate Loss of incompatibility of Ada infrastructure.

□ Incom_Ada_infra = Incom_Ada_infra - Loss_inco_Ada_infr + Crea_inco_Ada_infr
INIT(Incom_Ada_infra) = 2 {Incompatibility of Ada infrastructure (incompatability units)}

The initial value of 2 indicates a very low amount of incompatibility, since all that exists at the beginning of the simulation in 1975 is a set of specifications. This contrasts to the initial value of 40 for Non-Ada incompatibility, representing a great diversity among the different Non-Ada infrastructures.

Loss of incompatibility of Ada infrastructure (Equation #1610)

Once incompatibility develops, it is difficult to get rid of. Deployed systems that use a given piece of hardware and compiler version mean that the environment that supports programming with that hardware and software must be retained for maintenance. But the incompatible elements of infrastructure will eventually pass away. At one end of a continuum, elements of infrastructure just dissappear, and take their incompatibility with them. Target hardware passes out of use ("undeployed"?). Host hardware may pass out of use more quickly. The old editors and operating systems associated with the old hardware may likewise pass on. Programmers skilled in older, less used languages lose the skill while programming in newer languages, or retire or move to a different profession.

At the other end of the continuum, elements of incompatible infrastructure are overhauled to reduce the incompatibility. Programs written in variant dialects are gradually rewritten in standard dialects. (Look at how many dialects of even a standard language like JOVIAL have been used!) Programming and management styles gradually become more standardized. (Soon, structured programming may even become the norm!) A standard run-time environment for Ada may develop in a few more years. Market forces may standardize on tools, and cause, e.g., general-purpose editors or configuration management software to replace incompatible equivalents. Even though these various processes often take decades, they do happen. They are represented in the model by the Loss of incompatibility of Ada infrastructure.

Loss of incompatibility of Ada infrastructure (abbreviated Loss_inco_Ada_infr in the model) is a rate variable, which is an outflow rate from the level of Incompatibility of Ada infrastructure.

○ Loss_inco_Ada_infr = Incom_Ada_infra/
(Norm_dur_inco_A_inf * E_pol_du_inco_A_inf)
{Loss of incompatable Ada infrastructure (incompatability units/year)}

The form of the equation is the standard first-order delay, except that the effective delay time, Norm_dur_inco_A_inf * E_pol_du_inco_A_infr, has been made variable to represent possible policies, as explained below.

Normal duration of incompatibility of Ada infrastructure (Equation #1620)

The average lifetime of a unit of infrastructure incompatibility is called the Normal duration of incompatibility of Ada infrastructure.

Normal duration of incompatibility of Ada infrastructure (abbreviated Norm_dur_inco_A_inf) is a converter variable with no inputs, i.e., a constant.

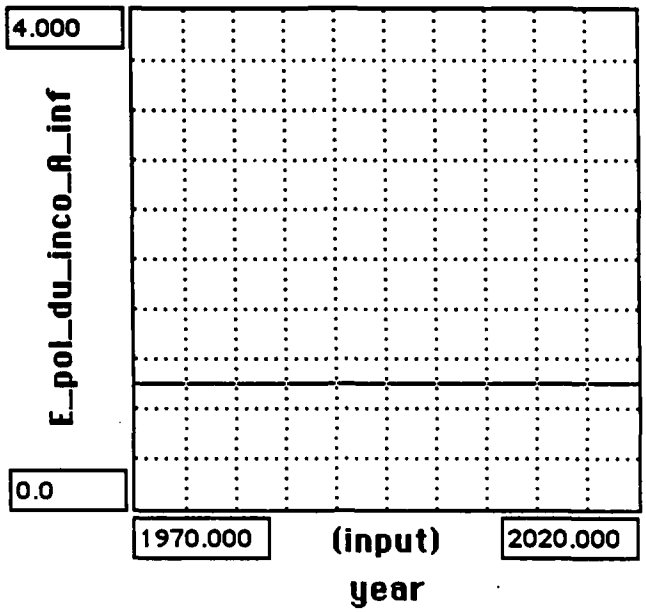
○ Norm_dur_inco_A_inf = 30 {Normal duration of incompatibility of Ada infrastructure (years)}

Of the various elements of infrastructure represented by incompatibility, some will vanish before the programming and hardware with which they are associated, the demise of some will be simultaneous with that of the hardware and software, and some will endure longer. Indeed, various characteristics of ECR programs can show all three behaviors. Programs written in a specific dialect may be slightly modified to compile as a standard dialect; here the incompatibility may last only a few years. When the system in which programs are embedded are mothballed, the programs are lost, along with whatever incompatible operating system, hardware, tools, libraries, and so on, were used with them. The language developed for such systems (as opposed to dialects and implementations of the language) will often endure far longer than any one MCCR system. There is a similar story to tell with most components of incompatibility. In the absence of better information, the Normal duration of incompatibility of Ada infrastructure is set equal to the lifetime of the projects whose infrastructure is being represented. The sum of the durations of development projects (10 years) and maintenance projects (20 years) is 30 years.

Effect of policy on incompatibility of Ada infrastructure (Equation #1630)

Establishing a standard which must be used in subsequent programming projects reduces the creation of incompatibility; actively transferring existing projects on to a standard increases the loss of incompatibility. Just as there are many ways that a given infrastructure can be incompatible with others, there are many ways that the DoD and services might use "retroactive" standardization to reduce incompatibility. In the Ada sphere, this might mean standardizing tools, or core program libraries, or just a tool interface, setting up some program where existing programming projects, either in development or maintenance phase, are moved over to the standard. In the non-Ada sphere, retro-standardization could also include translation into standard dialects of non-Ada languages, which would reduce the linguistic incompatibility of the non-Ada infrastructure. To provide the model with a policy level to experiment with the consequences of such policies, there is the Effect of policy on duration of incompatibility of Ada infrastructure.

The Effect of policy on duration of incompatibility of Ada infrastructure (abbreviated E_pol_du_inco_A_inf in the model) is a converter variable, converting the calendar year into an effect.



E_pol_du_inco_A_inf = graph(year)
1970.000 -> 1.000
1975.000 -> 1.000
1980.000 -> 1.000
1985.000 -> 1.000
1990.000 -> 1.000
1995.000 -> 1.000
2000.000 -> 1.000
2005.000 -> 1.000
2010.000 -> 1.000
2015.000 -> 1.000
2020.000 -> 1.000

For every year, the value is set at 1.0, so unless the graphic function is altered for an experiment, this multiplicative effect has no effect on the loss of incompatibility.

Creation of incompatibility of Ada infrastructure (Equation #1660)

The creation of incompatibility of infrastructure is basically a free market process. With one exception, incompatibility is a by-product, rather than the primary aim, of new product development. (The exception is the so-called "vanity standards"--a unique standard designed to lock customers in to a hardware manufacturer's products.) Incompatibility otherwise arises as a consequence of introducing an innovative product, for any of several reasons. The features and formats of other products may not be feasible to use, either because there are no comparable products, or because new functionality precludes their use. The "not invented here" syndrome may demotivate the use of formats and features

originated outside the organization. The product may simply have not been designed to work with some other piece of software that users turn out to want to use. (For example, neither the early microcomputer pioneers nor any major computer/office automation company anticipated that people would want their word processor documents to contain graphics.)

Even if Ada has solved some of the linguistic compatibility problems, opportunities still abound for introducing incompatible products into the programming support infrastructure. Even within Ada, there are still the matters of linkability and transportability of object code, the run-time environment, communications protocols, and most of all, calls to the operating system that are not standardized. Given the lack of any dominant standards in most of these areas, every new Ada product, even just a compiler, is likely to introduce some incompatibility into the infrastructure.

Similarly, any software tool is likely to create some incompatibility, through file formats if nothing else. Were it not for ASCII, the file formats would have gotten out of hand long ago. But merely having a standard for characters is far short of having compatibility. For example, it is rare for mainframe databases to be readable by more than one database program. (In microcomputers, the situation is different by virtue of a small number of widely-accepted standard database programs, plus the much greater simplicity of the databases.)

Given some normal free market creation of incompatibility of infrastructure, there are four major effects that can accelerate or retard the creation of incompatibility. The more Ada is used the more opportunity there is for creating incompatibility. Also, if it looks like Ada will be popular in the future developers will have incentives to create new products (tools, books, etc.) that will again tend to increase incompatibilility. Another influence on Ada incompatibility comes from the intensity of infrastructure. A higher infrastructure creates higher barriers to entry thereby reducing the creation of infrastructure. A final influence on the creation of incompatibility can come from DoD policy. Creation of incompatibility of Ada infrastructure combines all of these effects.

Creation of incompatibility of Ada infrastructure is a rate equation which flows into the level of Incompatibility of Ada infrastructure.

○ Crea_inco_Ada_infr = Norm_cr_inco_A_infr*E_inc_inco_A_infr*
E_int_inco_A_infr*E_use_inco_A_infr*E_pol_inco_A_infr {Creation of incompatibility of Ada infrastructure (incompatibility units/year)}

The equation follows the standard format of normal constant multiplied by several multiplicative effects. The normal constant gives what the rate of flow would be under some reference set of conditions, and the multiplicative effects specify the change to the flow that occur when conditions depart from the reference conditions.

Normal creation of incompatilibity of Ada infrastructure (Equation #1670)

The rate at which incompatibility will be created under the reference conditions is the Normal creation of incompatibility of Ada infrastructure.

Normal creation of incompatibility of Ada infrastructure is a converter variable with no inputs, i.e., a constant.

○ Norm_cr_inco_A_infr = .75
{Normal creation of incompatibility of Ada infrastructure (fraction)}

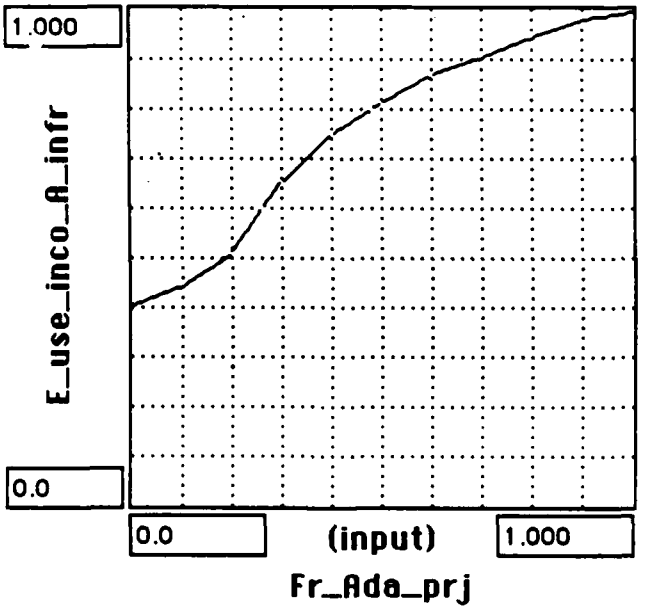
The reference conditions for which the normal creation is defined pertain to how much Ada is being used, the perceived incentives for its use, and the intensity of Ada infrastructure. There is a multiplicative effect on the rate of Creation of incompatibility of Ada infrastructure for each of these three conditions. The graph for each effect passes through 1.0 at some input value; that value is the reference condition for that effect.

Estimation of the normal creation parameters is discussed in Appendix A.9, "Multivariable Model Calibration."

Effect of relative use on incompatibility of Ada infrastructure (Equation #1680)

Use of Ada indicates market opportunities for new Ada-related products. Therefore, all other things being equal, the more Ada is used relative to non-Ada languages, the more motivation there will be to introduce new Ada-related products, and, in the normal course of events, the more incompatibility will be created. This influence is represented by the Effect of relative use on incompatibility of Ada infrastructure.

The Effect of relative use on incompatibility of Ada infrastructure (abbreviated E_use_inco_A_infr in the model) is a converter variable that uses a graphic function to specify an effect of incompatibility for any given "market share" of Ada, represented by the Fraction of Ada projects.



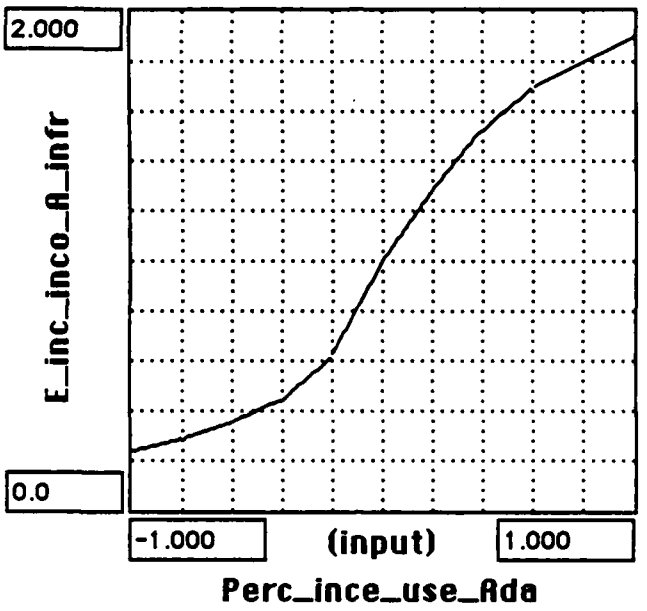
E_use_inco_A_infr = graph(Fr_Ada_prj)
0.0 -> 0.405
0.100 -> 0.445
0.200 -> 0.510
0.300 -> 0.655
0.400 -> 0.750
0.500 -> 0.815
0.600 -> 0.870
0.700 -> 0.905
0.800 -> 0.945
0.900 -> 0.980
1.000 -> 0.995

Using a Fraction of Ada projects as an indicator of market size for Ada products implicitly assumes some total market size, such that the fraction gives the size of the Ada part of the pie. The limitations of such an implicit assumption are discussed in the appendix on questions for further investigation.

Effect of incentives on incompatibility of Ada infrastructure (Equation #1690)

Perceived effectiveness of Ada, i.e., existence of incentives for others to be using Ada, also increases the perception of opportunities for introducing new products, and hence new incompatibility. This is called the Effect of incentives on incompatibility of Ada infrastructure.

Effect of incentives on incompatibility of Ada infrastructure (abbreviated E_inc_inco_A_infr in the model) is a converter variable that uses a graphic function to specify the effect of a given degree of Perceived incentives to use Ada

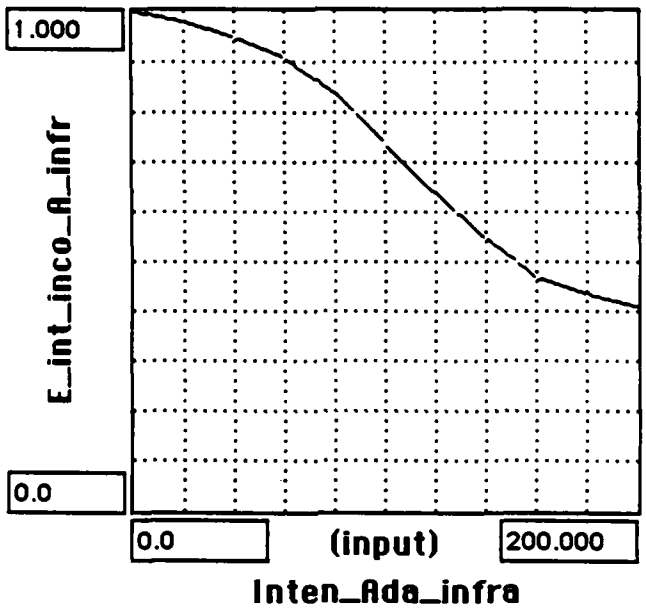


E_inc_inco_A_infr = graph(Perc_ince_use_Ada)
-1.000 -> 0.240
-0.800 -> 0.290
-0.600 -> 0.360
-0.400 -> 0.450
-0.200 -> 0.620
0.0 -> 1.000
0.200 -> 1.280
0.400 -> 1.530
0.600 -> 1.700
0.800 -> 1.800
1.000 -> 1.900

Effect of intensity on incompatibility of Ada infrastructure (Equation #1710)

How much effort does it take to introduce new, sometimes incompatible infrastucture? That depends on what the current infrastructure is like. If all that's required is a new compiler, then many companies have the resources to introduce new products. If compilers, APSEs with their myriad tools, and people who know how to use the present infrastructure already exist in abundance, considerably more resources will be required to create a new viable, and yet incompatible infrastucture. Fewer companies would have such resources, and therefore, there would be less creation of incompatibility. In other words, a high intensity of infrastructure constitutes a barrier to entry for new products, so intensity of infrastructure would reduce creation of incompatibility. This is represented by the Effect of intensity on incompatibility of Ada infrastructure.

The Effect of intensity on incompatibility of Ada infrastructure (abbreviated E_int_inco_A_infr in the model) is a converter variable that converts the Intensity of Ada infrastructure into an effect on creation of incompatibility.



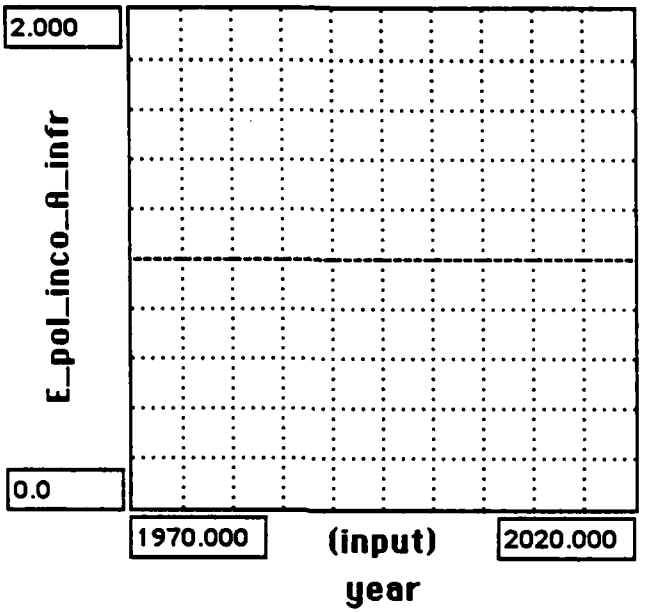
E_int_inco_A_infr = graph(Inten_Ada_infra)
0.0 -> 1.000
20.000 -> 0.980
40.000 -> 0.950
60.000 -> 0.910
80.000 -> 0.840
100.000 -> 0.735
120.000 -> 0.640
140.000 -> 0.550
160.000 -> 0.470
180.000 -> 0.435
200.000 -> 0.410

Effect of policy on incompatibility of Ada infrastructure (Equation #1720)

In addition to the free market effects described above, DoD policy can also have significant effects on the creation of incompatibility of Ada infrastructure. The Ada standard is assumed to reduce the creation of incompatibility. (This is represent by the normal creation being lower for Ada than for non-Ada.) But beyond the standard for language, standards for tools or operating system interfaces could significantly reduce the creation of incompatibility. Supplying a GFE'd APSE and allowing no others would reduce the generation of incompatibility even more. The policy lever in the model that

represents such policies and their effects on incompatibility is called the Effect of policy on incompatibility of Ada infrastructure.

The Effect of policy on incompatibility of Ada infrastructure (abbreviated E_pol_inco_A_infr in the model) is a converter variable, converting the calendar year to an effect on creation of incompatibility.



E_pol_inco_A_infr = graph(year)
1970.000 -> 1.000
1975.000 -> 1.000
1980.000 -> 1.000
1985.000 -> 1.000
1990.000 -> 1.000
1995.000 -> 1.000
2000.000 -> 1.000
2005.000 -> 1.000
2010.000 -> 1.000
2015.000 -> 1.000
2020.000 -> 1.000

Coverage of Ada infrastructure (Equation #1800)

The desirability of Ada as a language depends in part on its availability on various hosts and targets. A contractor is less likely to use Ada when no compiler will produce machine code for the hardware that's being embedded. Even having only one compiler available is not a comfortable situation -- the contractor is at the mercy of whoever produced the compiler to ensure that it works well. (Passing a validation test is NOT the same as

working well!) Moreover, software producers often have heavy investments in host computers and operating systems up front, which cannot be used if such machines and operating systems do not yet support any Ada compilers. In the aggregate, the more available Ada is on common hosts and targets, the more incentive there is to use Ada. In the model, the measure of that availability is the Coverage of Ada infrastructure. Figure A.7-4 shows a flow diagram of the inputs to this variable.

Coverage of Ada infrastructure (abbreviated Cov_Ada_infr in the model) is a level variable, whose one inflow rate is the Change in coverage of Ada infrastructure.

□ Cov_Ada_infr = Cov_Ada_infr + Ch_cov_Ada_infr
INIT(Cov_Ada_infr) = 0 {Coverage of Ada infrastructure (dimensionless)}

Coverage of both Ada and non-Ada infrastructure can range between 0, representing complete unavailability, and 1.0, representing complete availability on all hosts and targets. For discussion of the precise meaning of coverage for intermediate values, see Section 5 of this report. Coverage of Ada infrastructure is initialized at zero, since initially Ada is not available for any hosts or targets.

Change in coverage of Ada infrastructure (Equation #1810)

Like creation of intensity of infrastructure itself, additions to the coverage of host/target combinations seems to be a free-market affair, with companies creating products for sale, and contractors creating products for anticipated internal use. Therefore, additions to coverage will respond to the market incentives as they appear to potential developers of new PSEs (usually starting with a compiler). However, there will be a delay before vendors actually have the products with coverage responsive to those incentives. The gradual introduction of new compilers and their supporting software tools that extend coverage of Ada to new hosts and targets is represented by the Change in coverage of Ada infrastructure.

The Change in coverage of Ada infrastructure (abbreviated Ch_cov_Ada_infr in the model) is a rate variable, which flows into the level of Coverage of Ada infrastructure. Rate variables are a type of converter variable; here, the rate variable converts the difference between an Indicated coverage of Ada infrastructure and the actual coverage into a rate of change of the actual coverage.

○ Ch_cov_Ada_infr = (Ind_cov_A_infr-Cov_Ada_infr)/
(T_ch_cov_A_infr * E_pol_t_ch_cov_A)
{Change in coverage of Ada infrastructure (fraction/year)}

The form of the equation for Change in coverage of Ada infrastructure is the familiar first-order delay. Here, taking the difference between an indicated and actual coverage represents the delays inherent in the diffusion process: If economic considerations in the aggregate indicated that suppliers of Ada compilers and the associated software tools can make money on a certain fraction of all possible combinations of hosts and targets, it will take some time for the additional marketing opportunities to be discovered, and Ada compilers etc. to be written, rehosted, or retargeted.

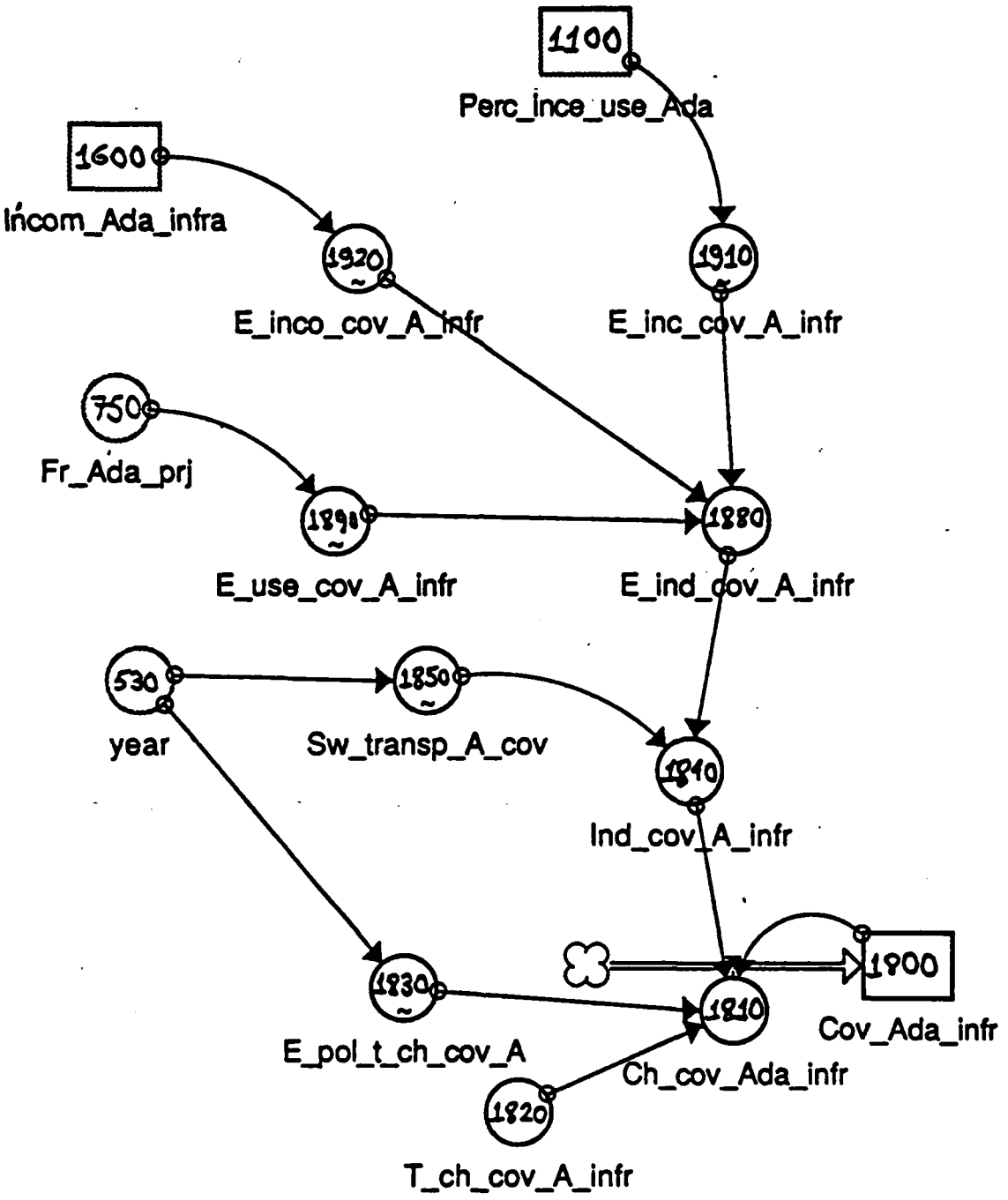


Figure A.7-4. Flow diagram of inputs to Coverage of Ada infrastructure (Cov_Ada_infr).

Time to change coverage of Ada infrastructure (Equation #1820)

In the aggregate, there will be an average time that it takes for market opportunities to be filled by products. Market opportunities are not always obvious or easy to realize, for the evidence of them is diffuse and multifarious. Weapons programs or other programs might use hosts or targets upon which Ada is not yet available. Reports of ongoing programming efforts gradually form a perception that Ada is being used more frequently, indicating a generally larger market and therefore larger niche markets in the less common combinations of hosts and targets. Announcements of new software tools for Ada and Ada environments could engender the same type of reasoning. There is also delay in taking advantage of opportunities once they are perceived. A new company may need to be formed, or new divisions formed within existing organizations. Then the actual work takes time to create an Ada compiler and so on, or more frequently, rehosting or retargeting existing software. The total average time taken for all of these processes is the Time to change coverage of Ada infrastructure.

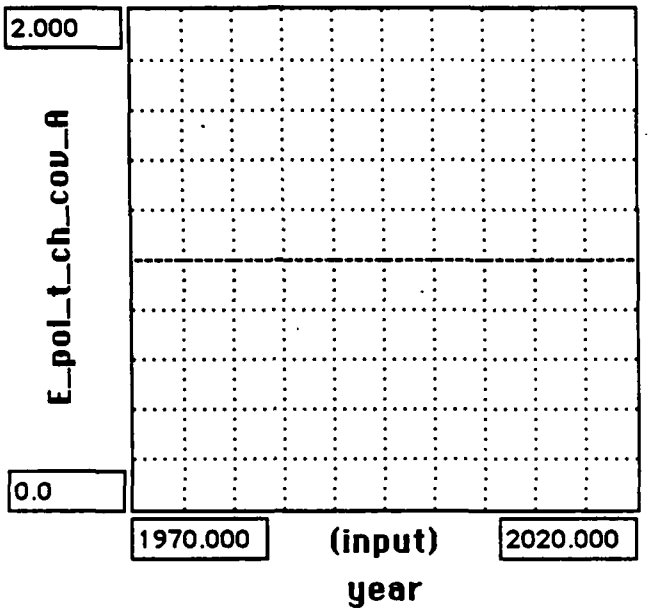
The Time to change coverage of Ada infrastructure (abbreviated T_ch_cov_A_infr in the model) is a converter variable with no inputs, i.e., a constant.

○ T_ch_cov_A_infr = 5
{Time to change coverage of Ada infrastructure (years)}

Effect of policy on time to change coverage of Ada infrastructure (Equation #1830)

There are several policies that could potentially influence the time it takes for Ada to spread to new hosts and targets. One such policy might be use of SVID as an interim standard tool interface for Ada-related software. Because of the reservoir of experience in porting UNIX-like systems to new machines, and the design of UNIX that makes this straightfoward, transporting the compiler and environment is much less time-consuming. The mechanism that accomplishes this reduction is the Effect of policy on time to change coverage of Ada infrastructure.

The Effect of policy on time to change coverage of Ada infrastructure is a converter variable, converting the calendar year into an effect for that year.



E_pol_t_ch_cov_A = graph(year)
1970.000 -> 1.000
1975.000 -> 1.000
1980.000 -> 1.000
1985.000 -> 1.000
1990.000 -> 1.000
1995.000 -> 1.000
2000.000 -> 1.000
2005.000 -> 1.000
2010.000 -> 1.000
2015.000 -> 1.000
2020.000 -> 1.000

The effect is set equal to 1.0 at all times in the base scenario model, representing no departure by policy from the present as regards propagation of Ada to new machines.

Indicated coverage of Ada infrastructure (Equation #1840)

As a convenient economic fiction, one can conceive of the coverage of Ada infrastructure that would exist eventually, given that the economic incentives stay as they are. This equilibrium coverage would represent a balance between the slow spread of Ada and its software environment into the last, marginally profitable combinations of host and target, and the "despreading" of host/target combinations no longer being used. In the model this equilibrium coverage is the Indicated coverage of Ada infrastructure.

Indicated coverage of Ada infrastructure is a converter variable, converting the Effects on indicated coverage of Ada infrastructure and a switch to the indicated coverage.

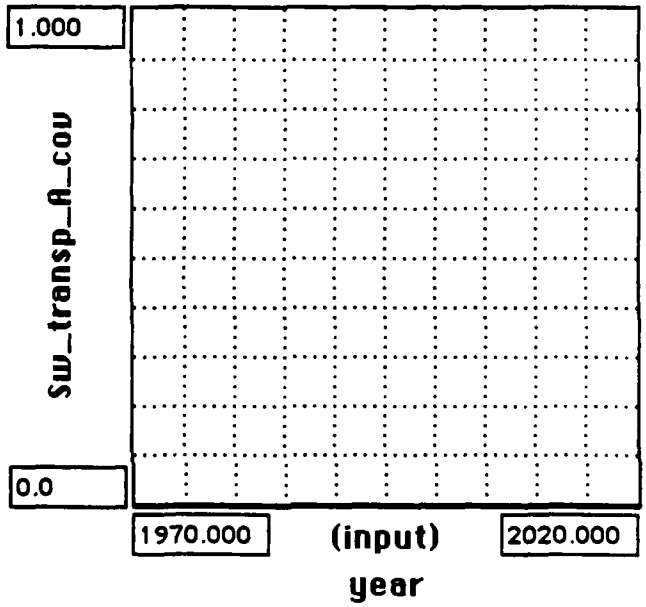
○ Ind_cov_A_infr = IF (Sw_transp_A_cov=1) THEN 1 ELSE E_ind_cov_A_infr
Indicated coverage of infrastructure units (infrastructure units))

The form of the equation for indicated coverage selects between two cases. In case the operating system and tool interface is standardized and easily transportable, the indicated coverage is 1.0, i.e., 100 percent: there are no substantial economic barriers to speak of that prevent implementing Ada and its environment on any viable combination of hosts and targets. (Viable in this context meaning, e.g., that no one will attempt using a tiny Sinclair personal microcomputer as a host.) In case the operating system and tool interface is not standardized and therefore expensive to transport, the indicated coverage will depend on the economic indicators that determine how many host/target combinations will be profitable and how many will not. In the latter case, the indicated coverage equals the Effects on indicated coverage of Ada infrastructure. The modeller controls which case is represented by the model at what time by the Switch for transportability on Ada coverage.

Switch for transportability on Ada coverage (Equation #1850)

One policy option to be explored in the model is whether an APSE is made available (either by GFE or by specification and private development), that is transported easily. The model represents the effect such an APSE would have on coverage with the Switch for transportability on Ada coverage.

The Switch for transportability on Ada coverage (abreviated Sw_transp_A_cov in the model) is a converter variable that uses a graphic function to choose between inputs. When drawn to equal 1 indicated to indicate an infrastructure that is intentially design to be transportable indicated coverage becomes one. When equal to zero indicated converage equals the endogenously determined Effects on indicated coverage of Ada infrastructure.



Sw_transp_A_cov = graph(year)
1970.000 -> 0.0
1975.000 -> 0.0
1980.000 -> 0.0
1985.000 -> 0.0
1990.000 -> 0.0
1995.000 -> 0.0
2000.000 -> 0.0
2005.000 -> 0.0
2010.000 -> 0.0
2015.000 -> 0.0
2020.000 -> 0.0

Effects on indicated coverage of Ada infrastructure (Equation #1880)

Many considerations go into a decision about whether to attempt to open a new market, be it automobiles or software products. There are of course considerations about what type of product to offer, but for Ada developers a major decision is what host and target machines the software should work on. The major consideration here is whether there will be demand for Ada on a particular configuration by projects in the same organization, contractors, and military programmers. In the aggregate, the future demand for Ada programming on all configurations cannot be known, only inferred from various indications, such as current use, cost-effectiveness of currently available Ada compilers and infrastructure, and incompatibility of existing infrastructure (which limits the market size for new products). In the model, the composite of these various considerations is the Effects on indicated coverage of Ada infrstructure.

The Effects on coverage of Ada infrastructure is a converter variable, converting effects from use, incompatibility, and incentives on coverage of Ada infrastructure into a single effect.

E_ind_cov_A_infr = E_inco_cov_A_infr*E_inc_cov_A_infr*
E_use_cov_A_infr (Effects on indicated coverage of Ada infrastructure)

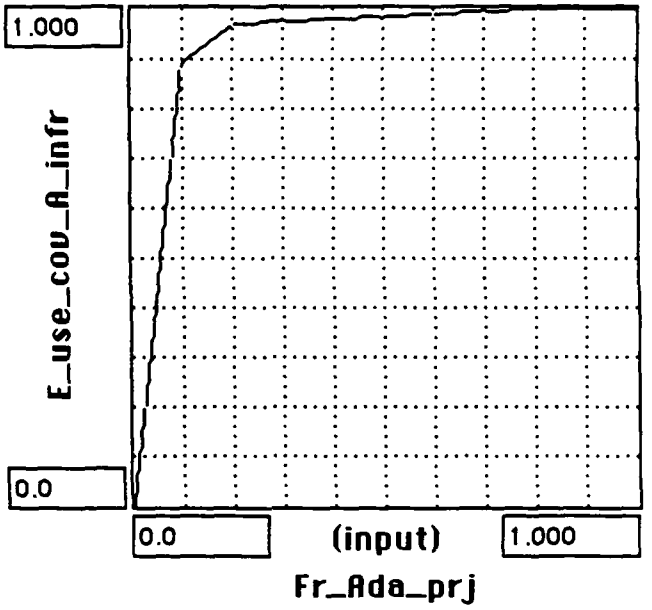
The multiplicative form of the equation above allows any single indicator to strongly restrict coverage if it is unfavorable enough. If no one were to use Ada, there would be no good marketing prospects for making Ada available on more machines. If Ada were terribly cost-ineffective, that would not be a sign that users would flock to Ada and demand products, at least not soon. If the Ada infrastructure were splintered into many incompatible environments, creating products for any one of the environments would address only a very small market, so the potential return on developing Ada products for new machines would be very small.

Effect of relative use on coverage of Ada infrastructure (Equation #1890)

Probably the most persuasive evidence that a given language and PSE represent a marketing opportunity is actual use. The more use of Ada is in evidence, the more incentives there are to create Ada environments to new host/target configurations (or to adapt and transport environments from other configurations. In the model, the effect of

such incentives on the development and spread of Ada products is represented by the Effect of relative use on coverage of Ada infrastructure.

The Effect of relative use on coverage of Ada infrastructure is a converter variable, converting the Fraction of Ada projects into an effect.



E_use_cov_A_infr = graph(Fr_Ada_prj)
0.0 -> 0.0
0.100 -> 0.895
0.200 -> 0.970
0.300 -> 0.975
0.400 -> 0.980
0.500 -> 0.985
0.600 -> 0.990
0.700 -> 0.995
0.800 -> 1.000
0.900 -> 1.000
1.000 -> 1.000

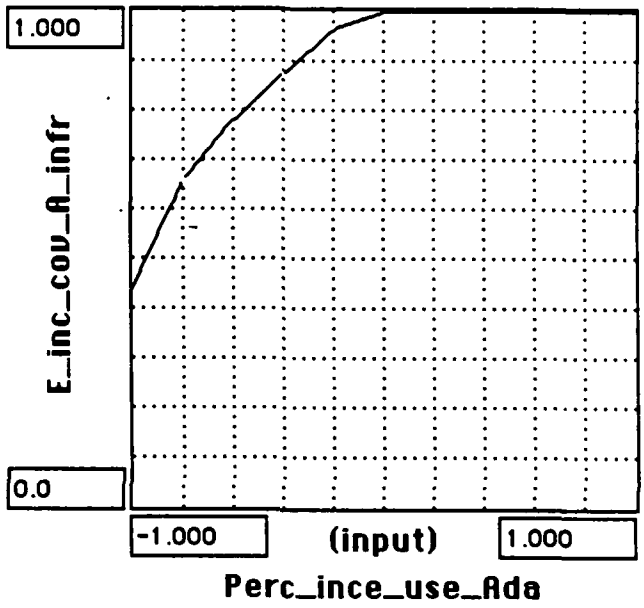
The Fraction of Ada projects is the fraction of DoD MCCR programming currently going on using the Ada lnaguage. It is a measure of how much Ada is being used.

Effect of incentives on coverage of Ada infrastructure (Equation #1910)

Another strong piece of evidence that a language and PSE represent a market for more products is if the cost/risk incentives indicate that the language and PSE are desirable for

programmers to use. Even if use has not yet blossomed, if Ada and APSE show good results, people will be encouraged to make them available on many machines. Conversely, if the current state of the APSE is poor, there is little experience with it, and so on, then there is considerably less promise in the Ada market (at least in the near future) and correspondingly less motivation to port APSE to new combinations of machines. The impact of cost/effectiveness incentives on desire to market Ada products on new host/target configurations is represented in the model by the Effect of incentives on coverage of Ada infrastructure

The Effect of incentives on coverage of Ada infrastructure is a converter variable, converting the Perceived incentives to use Ada into an effect.

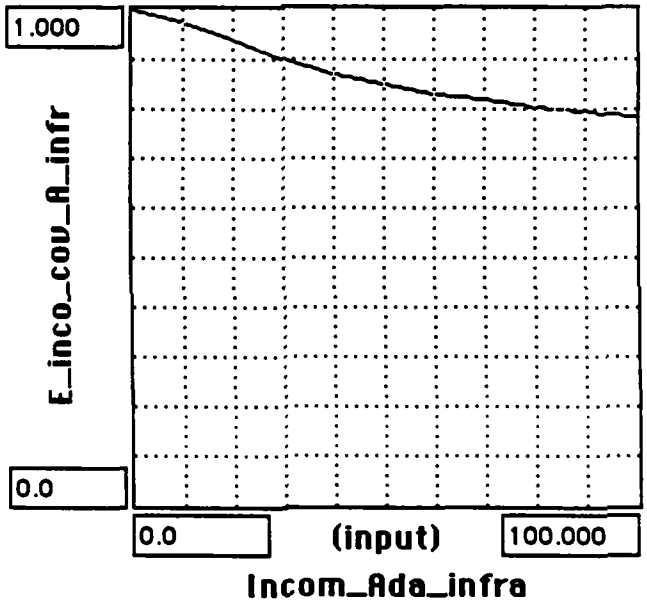


E_inc_cov_A_infr = graph(Perc_ince_use_Ada)
-1.000 -> 0.440
-0.800 -> 0.655
-0.600 -> 0.780
-0.400 -> 0.875
-0.200 -> 0.965
0.0 -> 1.000
0.200 -> 1.000
0.400 -> 1.000
0.600 -> 1.000
0.800 -> 1.000
1.000 -> 1.000

Effect of incompatibility on coverage of Ada infrastructure (Equation #1920)

Market fragmentation diminishes the marketing opportunities for new products. For example, if there is one standard Ada environment, creating a product (like a compiler) to host/target configuration XY means that it can be sold to anyone who uses or wants to use that standard environment on XY. But suppose there are 12 incompatible Ada environments. Creating a product for environment number 6 out of 12 to XY means that the market is only those using environment number 6 AND who want to use configuration XY; users of numbers 1-5 and numbers 7-12 would have to convert to a strange environment to use the new product. Accordingly, when an infrastructure is fragmented by incompatibility, the incentives to create new Ada products, in particular products for new configurations is diminished. This effect is represented in the model by the Effect of incompatibility on coverage of Ada infrastructure.

The Effect of incompatibility on coverage of Ada infrastructure is a converter variable, converting the index, Incompatibility of Ada infrastructure, into an effect.



E_inco_cov_A_infr = graph(Incom_Ada_infra)

0.0 -> 1.000
10.000 -> 0.975
20.000 -> 0.940
30.000 -> 0.900
40.000 -> 0.870
50.000 -> 0.850
60.000 -> 0.830
70.000 -> 0.820
80.000 -> 0.805
90.000 -> 0.795
100.000 -> 0.785

Appendix A.8: Non-Ada Infrastructure Sector

The explanations given above for Ada will not be repeated for non-Ada, since for the most part, the formulation and meaning of non-Ada infrastructure exactly parallels those of Ada infrastructure. The Non-Ada equations lack some policy levers possessed by the corresponding Ada structures, which requires no further comment. Values of corresponding Ada and non-Ada parameters have been made exactly equal unless an argument exists for a difference. Those arguments are given in Appendix A.9, "Multivariable Model Calibration." The 1975 initial conditions for non-Ada language infrastructure differ considerably from those of the (nonexistent) Ada infrastructure of that time; these differences have been described in Section 5 of this report, and are elaborated on in Appendix A.9. It suffices, therefore, to give the non-Ada equations and their flow diagrams.

Figure A.8-1 shows the flow diagram for inputs to the first of the three level variables in the non-Ada infrastructure sector, the Intensity of non-Ada infrastructure.

Intensity of non-Ada infrastructure (Equation #2000)

□ Inten_NA_infra = Inten_NA_infra - Obsol_int_NA_infr + Crea_int_NA_infr
INIT(Inten_NA_infra) = 35
{Intensity of non-Ada infrastructure
(intensity units)}

Obsolescence of intensity of non-Ada infrastructure (Equation #2010)

○ Obsol_int_NA_infr = Inten_NA_infra / Dura_int_NA_infr
{Obsolescence of intensity of non-Ada infrastructure
(intensity units/year)}

Duration of intensity of non-Ada infrastructure(Equation #2020)

○ Dura_int_NA_infr = 30 {Duration of intensity of NonAda infrastructure (years)}

Creation of intensity of non-Ada infrastructure (Equation #2050)

Figure A.8-2 shows a flow diagram of the inputs to the Creation of intensity of non-Ada infrastructure.

○ Crea_int_NA_infr = Norm_cr_int_NA_infr *E_tech_cr_int*
E_use_int_NA_infr * E_inco_int_NA_infr * E_inc_A_int_NA *
E_rel_infr_int_NA
{Creation of intensity of Non-Ada infrastructure
(intensity units/year)}

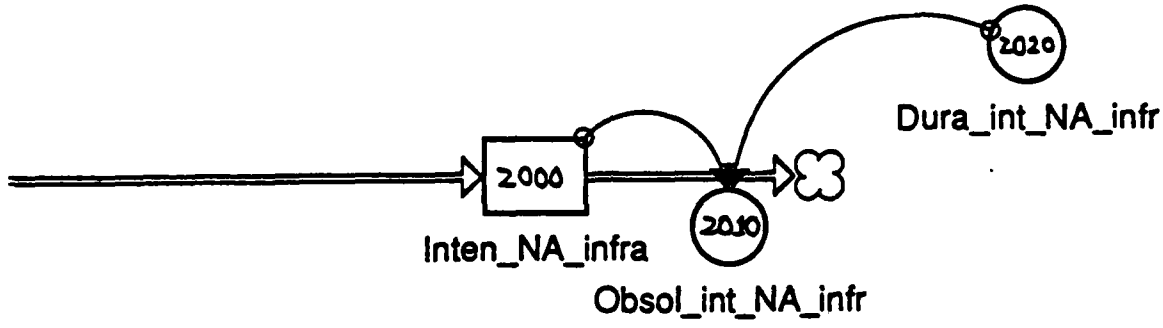


Figure A.8-1. Flow diagram of inputs to Intensity of non-Ada infrastructure (Intens_NA_infr). The rate of flow coming from the left side of the diagram is the Creation of intensity of non-Ada infrastructure.

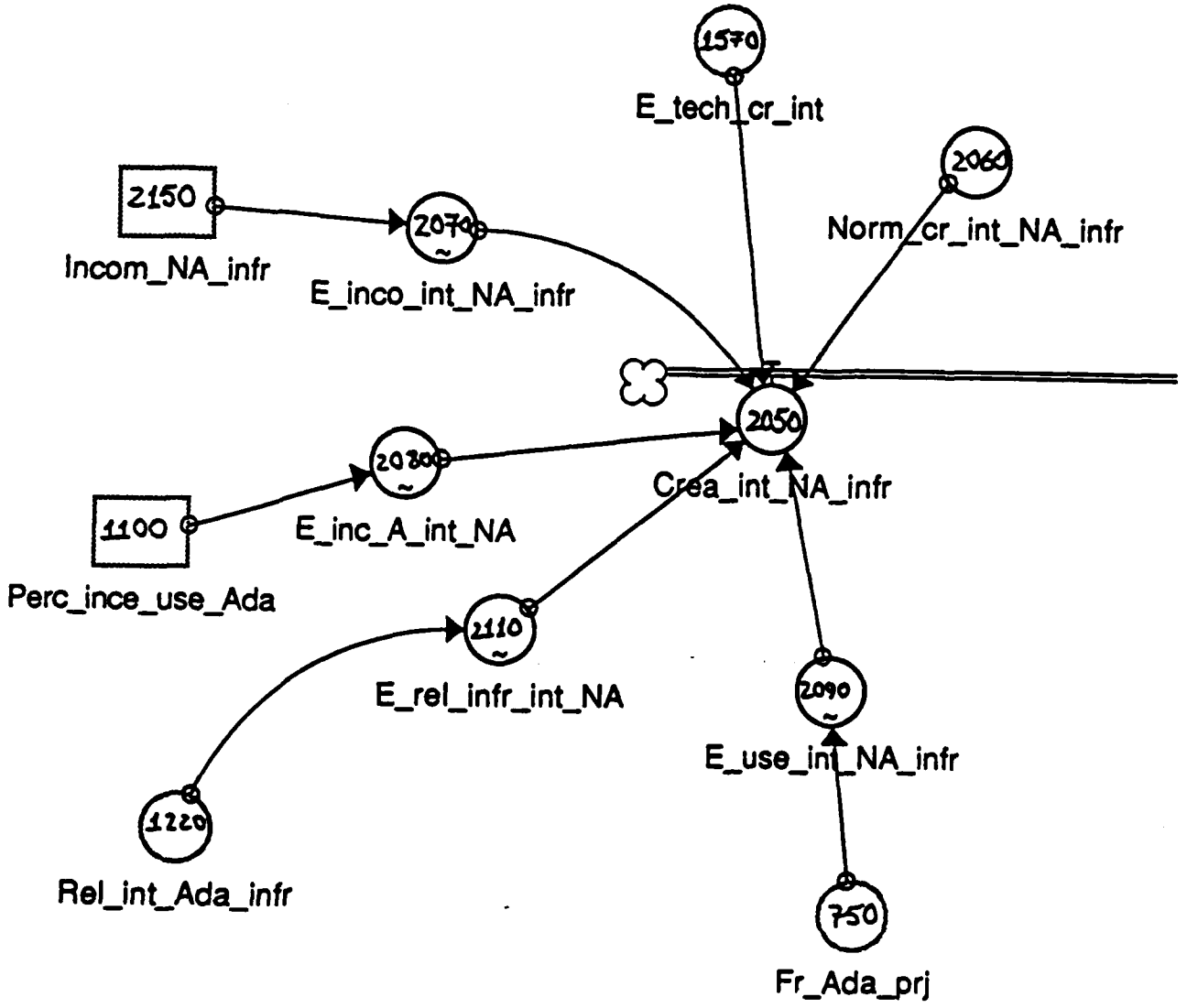
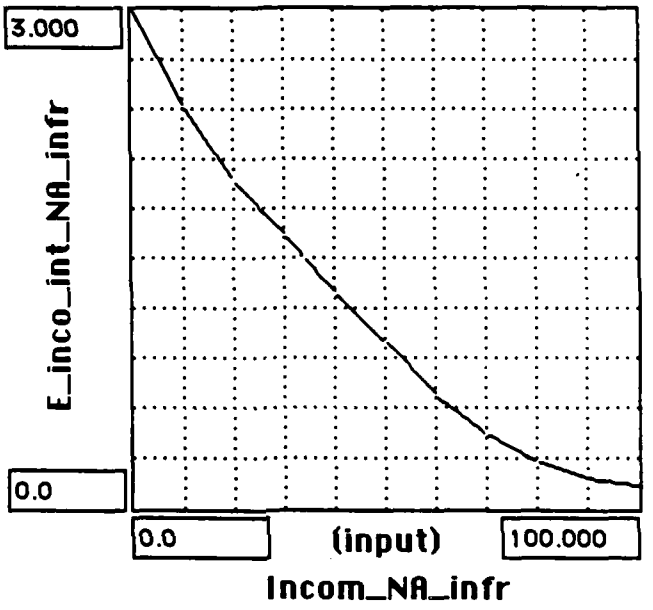


Figure A.8-2. Flow diagram of inputs to Creation of intensity of non-Ada infrastructure (Crea_int_NA_infr). The rate exits the right side of the diagram and flows into the Intensity of non-Ada infrastructure.

Normal creation of intensity of non-Ada infrastructure (Equation #2060)

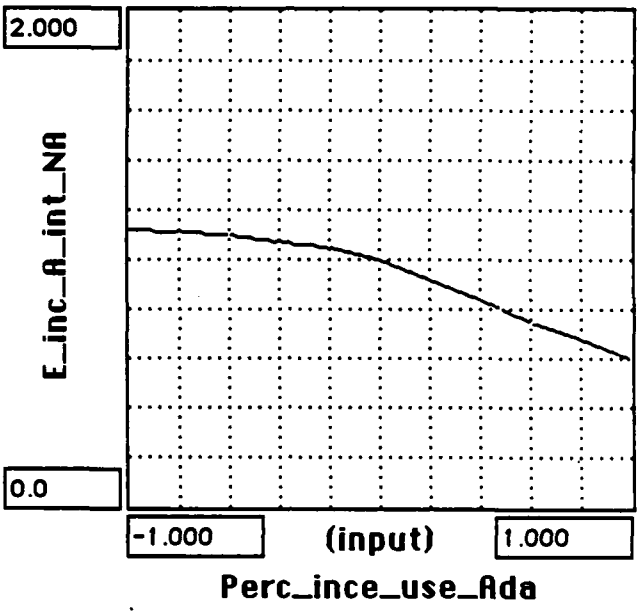
○ Norm_cr_int_NA_infr = 1.6
{Normal creation of intensity of non-Ada infrastructure (intensity units/year)}

Effect of incompatibility on intensity of non-Ada infrastructure (Equation #2070)



⊘ E_inco_int_NA_infr = graph(Incom_NA_infr)
0.0 -> 3.000
10.000 -> 2.415
20.000 -> 1.965
30.000 -> 1.635
40.000 -> 1.290
50.000 -> 1.000
60.000 -> 0.675
70.000 -> 0.450
80.000 -> 0.285
90.000 -> 0.180
100.000 -> 0.135

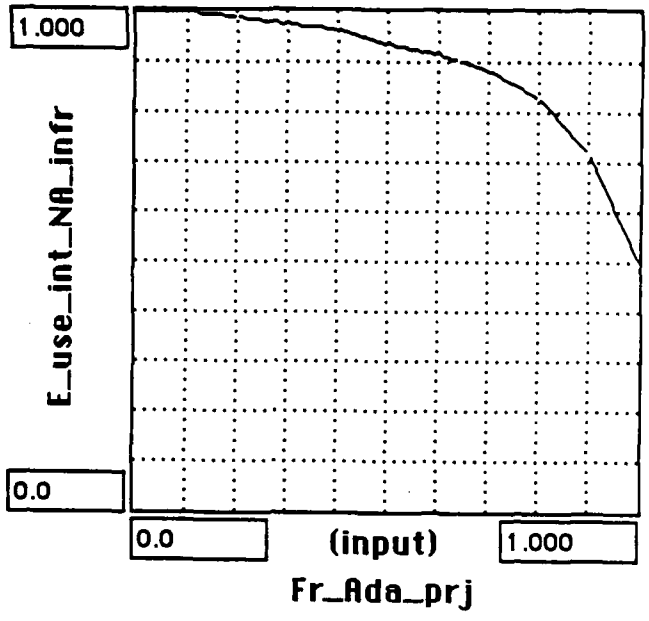
Effect of incentives for Ada use on intensity of non-Ada infrastructure (Equation #2080)



E_inc_A_int_NA = graph(Perc_ince_use_Ada)

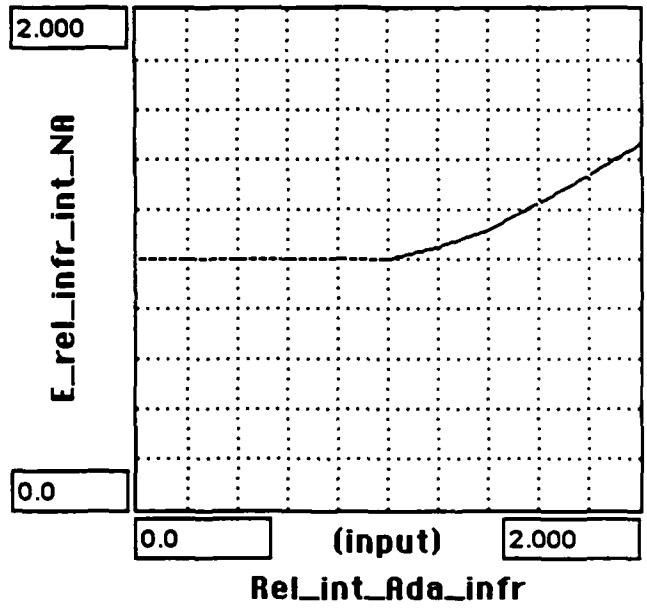
-1.000 -> 1.120
-0.800 -> 1.110
-0.600 -> 1.100
-0.400 -> 1.070
-0.200 -> 1.050
0.0 -> 1.000
0.200 -> 0.920
0.400 -> 0.840
0.600 -> 0.750
0.800 -> 0.680
1.000 -> 0.600

Effect of use on intensity of non-Ada infrastructure (Equation #2090)



E_use_int_NA_infr = graph(Fr_Ada_prj)
0.0 -> 1.000
0.100 -> 1.000
0.200 -> 0.990
0.300 -> 0.975
0.400 -> 0.965
0.500 -> 0.935
0.600 -> 0.915
0.700 -> 0.885
0.800 -> 0.830
0.900 -> 0.715
1.000 -> 0.500

Effect of relative infrastructure on intensity of non-Ada infrastructure (Equation #2110)



E_rel_infr_int_NA = graph(Rel_int_Ada_infr)
0.0 -> 1.000
0.200 -> 1.000
0.400 -> 1.000
0.600 -> 1.000
0.800 -> 1.000
1.000 -> 1.000
1.200 -> 1.050
1.400 -> 1.120
1.600 -> 1.230
1.800 -> 1.340
2.000 -> 1.460

Incompatibility of Non-Ada infrastructure (Equation #2150)

Figure A.8-3 shows a flow diagram of the inputs to Incompatibility of non-Ada infrastructure.

Incom_NA_infr = Incom_NA_infr - Loss_inco_NA_infr + Crea_inco_NA_infr
INIT(Incom_NA_infr) = 40 {Incompatibility of non-Ada infrastructure (incompatibility units)}

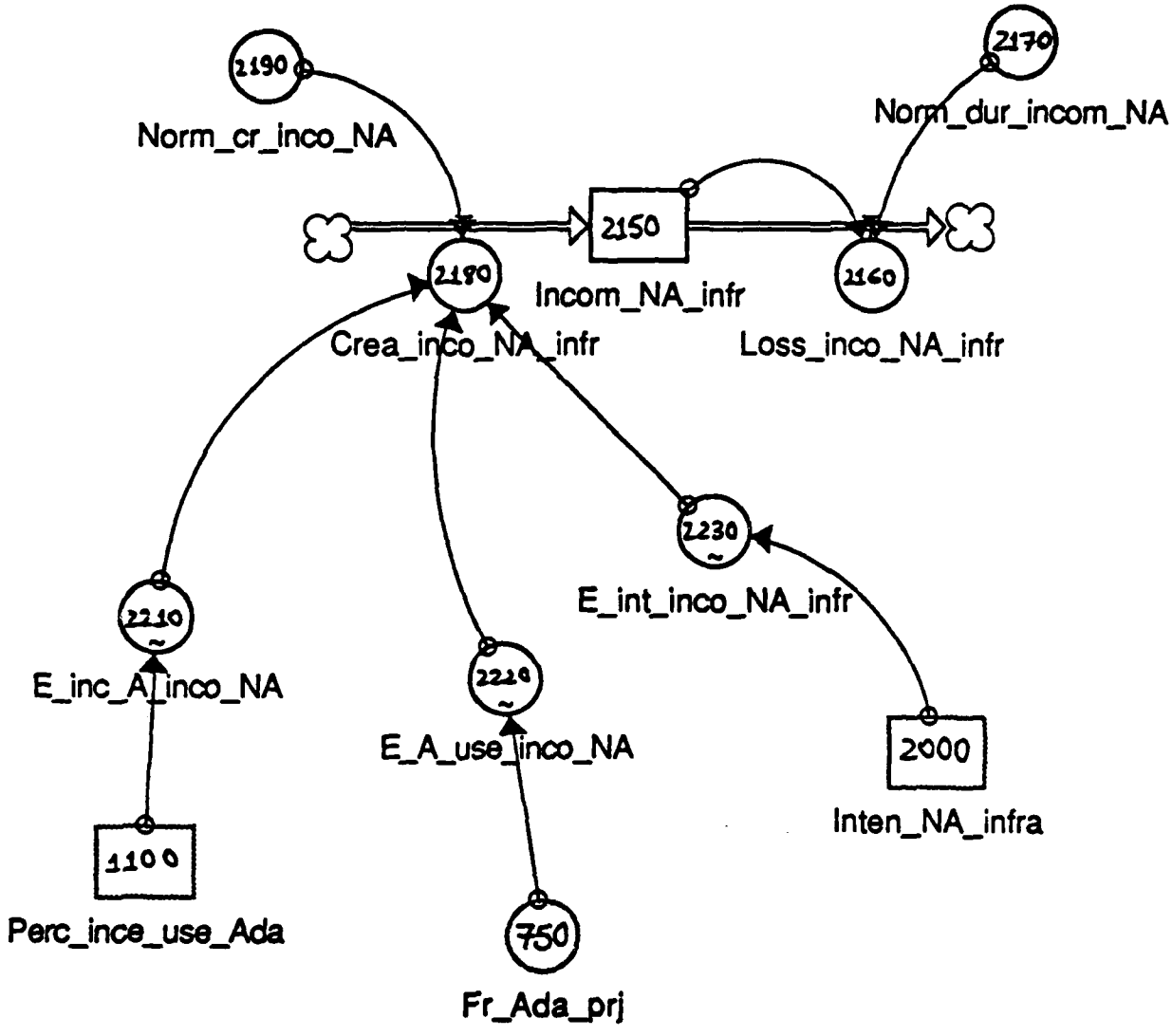


Figure A.8-3. Flow diagram of inputs to Incompatibility of non-Ada infrastructure (Incom_NA_infr).

Loss of incompatibility of non-Ada infrastructure (Equation #2160)

○ Loss_inco_NA_infr = Incom_NA_infr / Norm_dur_incom_NA {Loss of incompatibility of non-
Ada infrastructure (incompatibility units/year)}

Normal duration of incompatibility of non-Ada infrastructure (Equation #2170)

○ Norm_dur_incom_NA = 30 {Normal duration of incompatibility of non-Ada infrastructure (years)}

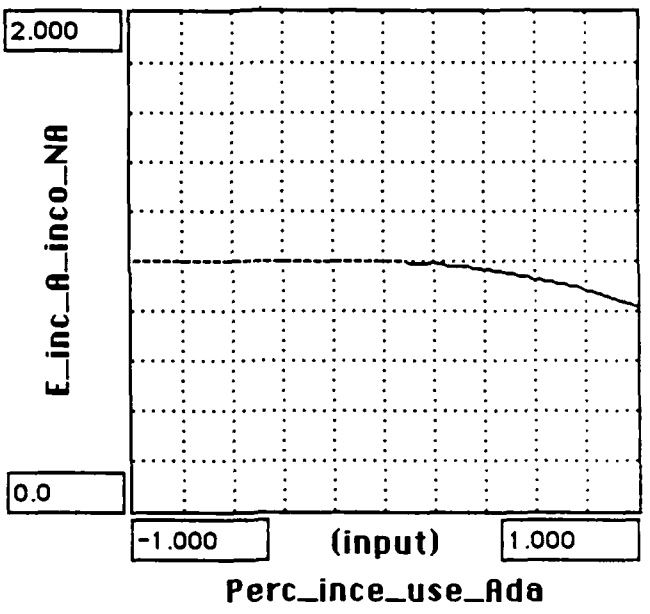
Creation of incompatibility of non-Ada infrastructure (Equation #2180)

○ Crea_inco_NA_infr = Norm_cr_inco_NA * E_int_inco_NA_infr * E_A_use_inco_NA * E_inc_A_inco_NA
{Creation of incompatible Nonada infrastructure (incompatibility units/year)}

Normal creation of incompatibility of non-Ada infrastructure (Equation #2190)

○ Norm_cr_inco_NA = 3 {Normal creation of incompatibility of non-Ada infrastructure (incompatibility units/year)}

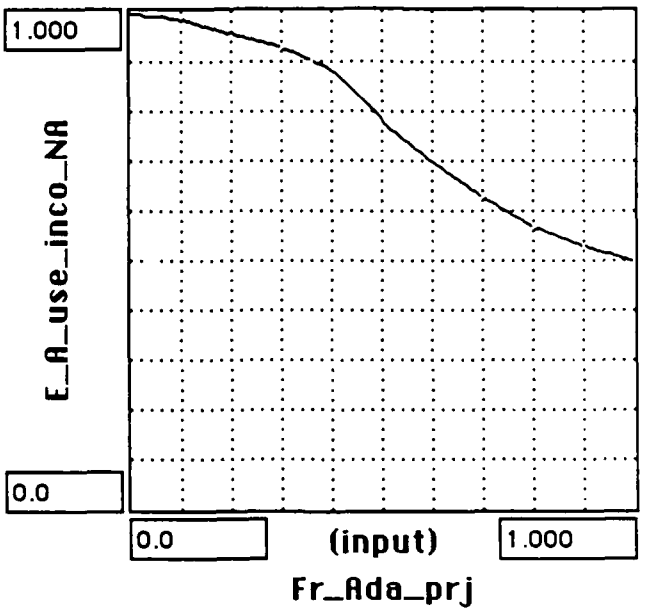
Effect of incentives for Ada use on incompatibility of non-Ada infrastructure (Equation #2210)



E_inc_A_inco_NA = graph(Perc_ince_use_Ada)

-1.000 -> 1.000
-0.800 -> 1.000
-0.600 -> 1.000
-0.400 -> 1.000
-0.200 -> 1.000
0.0 -> 1.000
0.200 -> 0.990
0.400 -> 0.970
0.600 -> 0.930
0.800 -> 0.890
1.000 -> 0.820

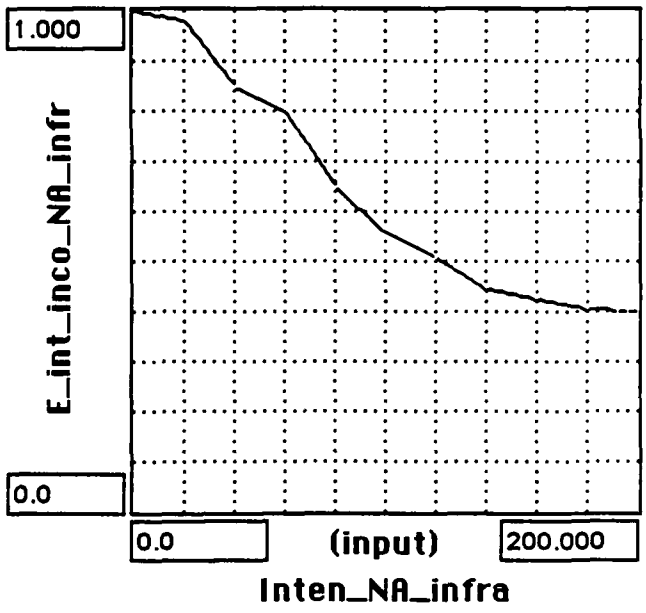
Effect of Ada use on incompatibility of non-Ada infrastructure (Equation #2220)



E_A_use_inco_NA = graph(Fr_Ada_prj)

0.0 -> 0.995
0.100 -> 0.985
0.200 -> 0.955
0.300 -> 0.930
0.400 -> 0.885
0.500 -> 0.780
0.600 -> 0.700
0.700 -> 0.630
0.800 -> 0.570
0.900 -> 0.530
1.000 -> 0.500

Effect of intensity on incompatibility of non-Ada infrastructure (Equation #2230)



E_int_inco_NA_infr = graph(Inten_NA_infra)
0.0 -> 1.000
20.000 -> 0.980
40.000 -> 0.850
60.000 -> 0.800
80.000 -> 0.650
100.000 -> 0.560
120.000 -> 0.510
140.000 -> 0.445
160.000 -> 0.425
180.000 -> 0.405
200.000 -> 0.400

Coverage of Non-Ada infrastructure (Equation #2300)

Figure A.8-4 shows a flow diagram of the inputs to Coverage of non-Ada infrastructure.

Cov_NA_infr = Cov_NA_infr + Ch_cov_NA_infr
INIT(Cov_NA_infr) = .85
{Coverage of non-Ada infrastructure (dimensionless)}

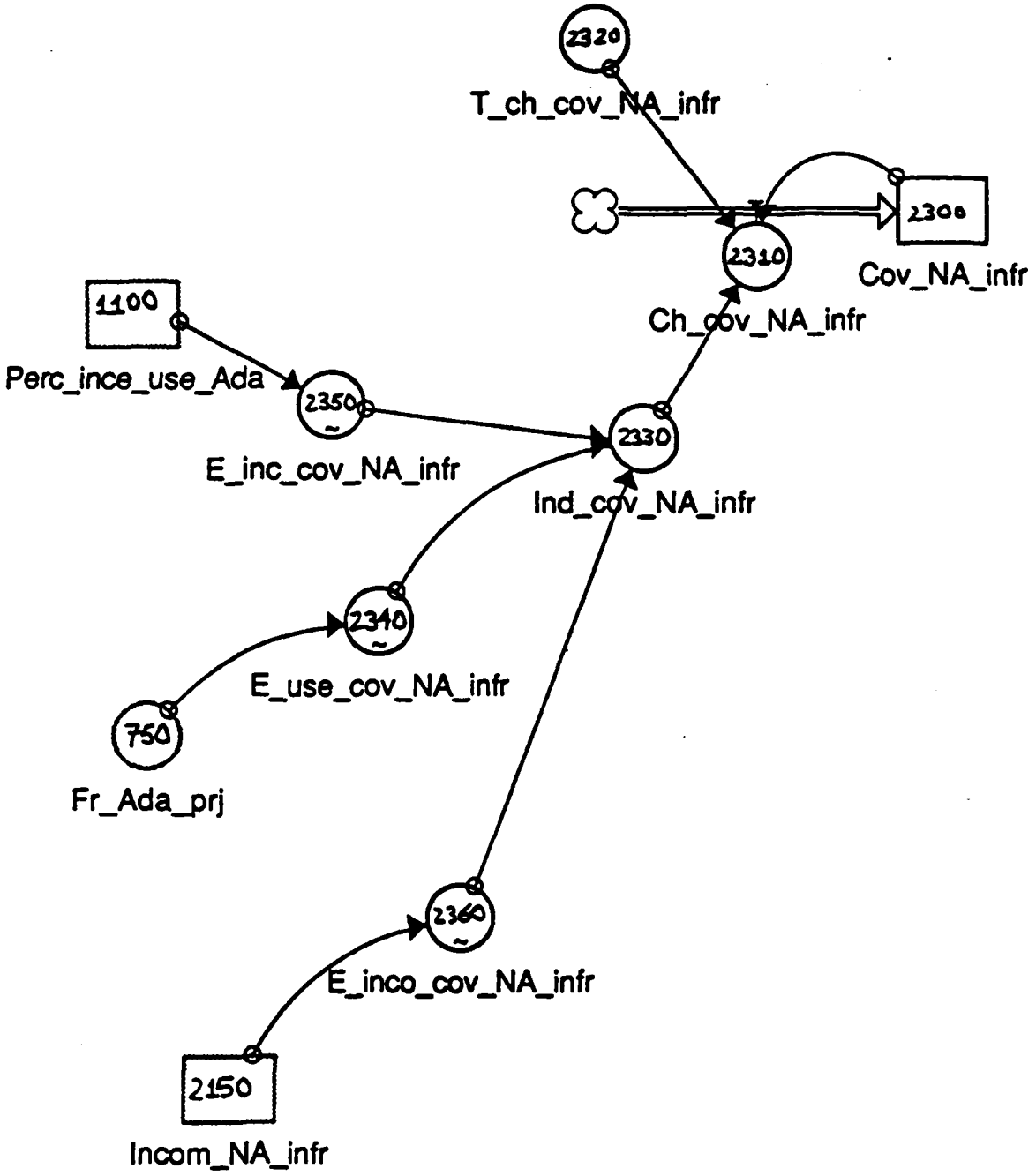


Figure A.8-4. Flow diagram of inputs to Coverage of non-Ada infrastructure (Cov_NA_infr).

Change in coverage of non-Ada infrastructure (Equation #2310)

○ Ch_cov_NA_infr = (Ind_cov_NA_infr-Cov_NA_infr)/T_ch_cov_NA_infr
{Change in coverage of non-Ada infrastructure
(coverage units/year)}

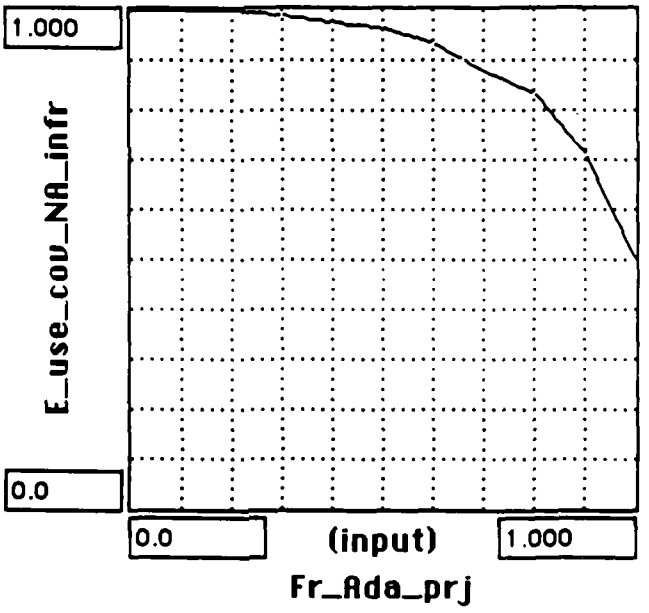
Time to change coverage of non-Ada infrastructure (Equation #2320)

○ T_ch_cov_NA_infr = 5 {Time to change converage of
non-Ada infrastructure (years)}

Indicated coverage of non-Ada infrastructure (Equation #2330)

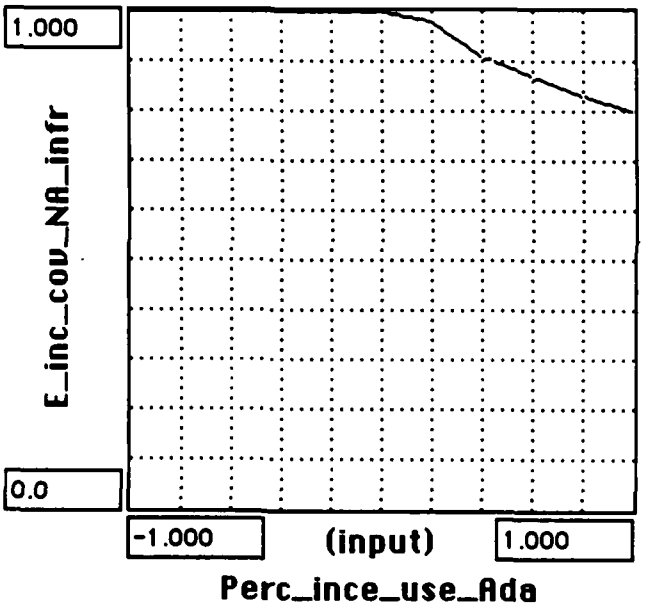
○ Ind_cov_NA_infr = E_use_cov_NA_infr*E_inc_cov_NA_infr*
E_inco_cov_NA_infr {Indicated coverage of
non-Ada infrastructure (dimensionless)}

Effect of relative use on coverage of non-Ada infrastructure (Equation #2340)



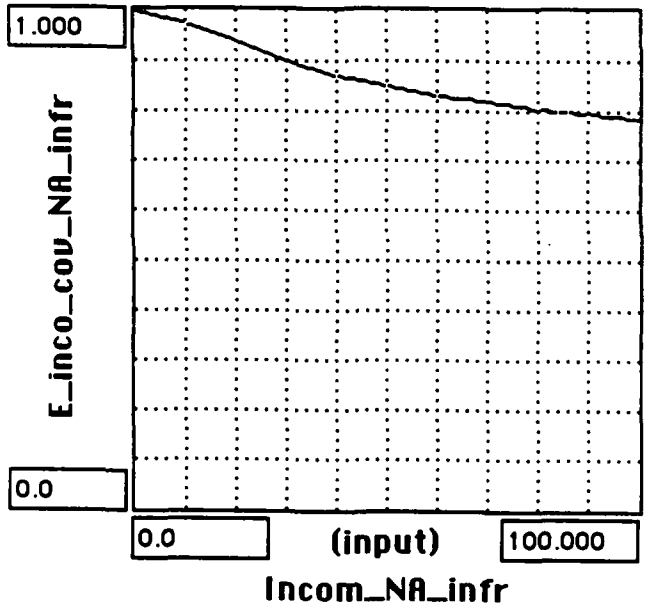
E_use_cov_NA_infr = graph(Fr_Ada_prj)
0.0 -> 1.000
0.100 -> 1.000
0.200 -> 1.000
0.300 -> 0.990
0.400 -> 0.975
0.500 -> 0.965
0.600 -> 0.935
0.700 -> 0.880
0.800 -> 0.835
0.900 -> 0.710
1.000 -> 0.500

Effect of incentives on coverage of non-Ada infrastructure (Equation #2350)



E_inc_cov_NA_infr = graph(Perc_ince_use_Ada)
-1.000 -> 1.000
-0.800 -> 1.000
-0.600 -> 1.000
-0.400 -> 1.000
-0.200 -> 1.000
0.0 -> 1.000
0.200 -> 0.980
0.400 -> 0.910
0.600 -> 0.870
0.800 -> 0.830
1.000 -> 0.800

Effect of incompatibility on coverage of non-Ada infrastructure (Equation #2360)



E_inco_cov_NA_infr = graph(Incom_NA_infr)

0.0 -> 1.000
10.000 -> 0.975
20.000 -> 0.940
30.000 -> 0.900
40.000 -> 0.870
50.000 -> 0.850
60.000 -> 0.830
70.000 -> 0.820
80.000 -> 0.805
90.000 -> 0.795
100.000 -> 0.785

Appendix A.9: Multivariable Model Calibration

The description of state variables and their inputs reports some calibration of the model, which is included as a short rationale after the equation is described. This section completes the description of model calibration, with the instances where calibration is either fairly involved, or involves several parameters at once. In particular, all parameter estimations where the respective Ada and non-Ada parameters differ are described here, rather than in the equation descriptions.

The descriptions below relate to calibrations actually performed. Possible future calibrations are described in Appendix C, "Areas for Further Investigation."

A note about calibration philosophy is in order. For analyses involving a single policy, like (Clapp, et al. 1977) on cost-benefits of mandating a standard HOL, it often suffices to simply make "conservative" assumptions in the model or calculation and report the results. However, in an analysis that scrutinizes multiple policies, making "conservative" assumptions is tricky at best, and distorting at worst. This is because what may be a conservative assumption for one policy may be not only not conservative, but exaggerate the effects of some other policy.

For example, consider assumptions about the number of programming projects started in the future; there is very little hard evidence out beyond 1990, and yet the analysis requires an assumption out to 2015 or so. For analyzing the policy of mandating Ada, a conservative assumption would be to assume low growth in project starts. This assumption would shrink the size of the programming pie, and hence be conservative in estimating the impact of standardizing on Ada on the cost of that pie. But for other policies, such an assumption may exaggerate their effects.

For example, one of the policies discussed here is migrating non-Ada programs to a standardized operating system. The effectiveness of that policy relative to policies concerning Ada language programs hinges on the proportions of Ada and non-Ada systems in development and maintenance. With very low growth, the older programs in non-Ada languages will be proportionately more numerous than would be the case with rapid growth. Therefore, policies that address non-Ada costs will appear more desirable relative to those that address Ada costs with a low-growth assumption than with a higher-growth "best guess" assumption. The bottom line, then, is that simply making conservative assumptions and reporting results is not appropriate for the multiple-policy analysis. The proper procedure is to make "best guess" assumptions to evaluate policies relative to one another, and then use sensitivity testing to ensure that the relative desirability of policies holds for any plausible set of circumstances, as prescribed in DoD Instruction 7041.3.

In a model produced by rapid prototyping such as this, the parameter estimation procedures are quite informal and quick relative to the ponderous data massage of formal econometric regression. For a discussion of the role of *a priori* and informal parameter estimations, see (Graham 1980).

Cost sector calibration

Reference costs. The general procedure for estimating the reference costs (equations 110, 220, 290, and 340, in the cost sector) is to arbitrarily define one, implicitly defining what is meant in the model by a "project." The parameter thus defined is the Reference cost of non-Ada development projects (Ref_cst_NA_dev). Then the value for the Reference cost of non-Ada maintenance projects (Ref_cst_NA_mn) is chosen in order to have an appropriate relationship to the development cost. Then the parameters for Ada development

and maintenance are similarly derived to have an appropriate relationship to the corresponding non-Ada parameters and to each other. So one parameter is chosen as a matter of defining the scale of effort denoted by a "project" and the other parameter values are derived from assumed relationships with the first parameter chosen.

As discussed in Section 5 of this report, the definition of the amount of programming work that constitutes a standard "project" is selectable rather arbitrarily, subject only to consistency of treatment. For the present round of model development, the definition of a "project" is implicitly specified when the cost per project year of a project is defined. A Non-Ada development project is defined to cost, on average, 6 million dollars (Ref_cst_NA_dev = 6×10^6, or in model notation, 6e6).

While the selection of development costs is somewhat arbitrary, the selection of maintenance costs must be choosen to be consistent with development costs. Heuristically, the cost of maintenance can be derived from more basic observations. Suppose that a given piece of software in the maintenance phase has major upgrades periodically, such that one year out of every three, it is undergoing major revisions. Further suppose that major revisions cost comparably to a development project. Two years out of three maintenance will be routine, and much less costly. Reflecting only the cost of major revisions, the maintenance cost for an average year should be one-third the yearly cost of development. Reflecting both major and minor revisions, the average cost should be somewhat more than the one-third. Here, one-half of the development cost is the ratio chosen; Ref_cst_NA_mn = 3e6). Here, then, maintenance costs one-half as much per year as development. (If the maintenance phase of a projects life cycle lasts longer than the development phase, the amount spent on maintenance may exceed that spent on development over the course of the life cycle.)

Similarly, Ada development costs are estimated from their relation to non-Ada development costs. A value has been chosen that asserts that doing a project in the Ada language, if the infrastructure intensity for programming in Ada were comparable to the intensity of infrastructure available for non-Ada languages, would cost somewhat less than in a typical non-Ada language (5 million versus 6 million per year). Ada is assumed to be somewhat more efficient than non-Ada languages if the infrastructures were comparable by virtue of its design supporting good software practices.

It is true that good software practices can be followed without the support of the language used; people have programmed generics in FORTRAN, and even BASIC programs can be developed in a structured way. But it seems plausible to assume that for a given amount of programming experience in a language, better practices will be followed if the language supports those practices, and less cost will result from better practices. Ada's support of both top-down and bottom-up programming, structured programming, generics, packages, strongly-checked types, and so on, suggests that for a given level of infrastructure, Ada projects will be less expensive than non-Ada projects. This assumption is born out by the details reported in (Foreman 1985a and b) and (IDA 1985).

The parameters for cost (the reference costs and the graphic functions that define the influence of incompatibility and intensity of infrastructure on costs) have been chosen such that in 1985, Ada projects cost more than the comparable non-Ada projects, despite the lower reference cost. This is due to Ada's present lack of infrastructure (programming experience, programming tools, program libraries, courses of instruction, etc.) by comparison to older languages.

The yearly maintenance cost for Ada, by arguments similar to those above, is set less than the yearly development cost. Moreover, the maintenance cost is proportionately less of the development cost than is the case for non-Ada langauges. Again, the difference is attributed to the design of Ada. More than any other computer language, Ada is designed to be maintainable, by virtue of features that yield clear program structure and isolate parts of programs from each other. Non-Ada maintenance yearly expenditures per project were set at 50 percent of the development cost; the corresponding Ada cost is set at 40 percent (Ref_cst_A_mn = 2e6).

Effect of infrastructure intensity. Non-Ada prgramming costs respond to intensity of infrastructure. The curve that defines that response, the graph function Effect of intensity of non_Ada infrastructure on cost (E_int_NA_cst), ranges from nearly 2.0 to 0.3. The multiplier (a "cost driver") gives changes in cost when the intensity of infrastructure departs from a reference value, which is chosen to be the 1985 intensity, 40 intensity units. Therefore, the multiplier is 1.0 when Intensity of non-Ada infrastructure (Inten_NA_infra) equals 40. Less infrastructure yields higher costs, in the extreme of no infrastructure, nearly double the refernce cost. More infrastructure yields lower costs.

There are a number of studies that relate software cost to what amounts to the infrastructure. Foremost among these are Boehm's studies that resulted in the COCOMO model (Boehm 1981). No formal derivation of the cost curve from such a model has been done yet. The slope of the curve is about twice the slope of the curve in COCOMO that relates tools to productivity. However, infrastructure is not just tools. The quality of people, the extensiveness of support hardware, the size of program libraries, and the methods of managing programming all play a role. Given that tools are just one component of infrastructure, the slope of the curve seems appropriate, at least until a more formal derivation is performed.

The curve relating Ada infrastructure to costs is steeper than the curve for non-Ada. The Effect of intensity of Ada infrastructure on cost (E_int_A_cst) slopes about twice as much for infrastructure more than the reference intensity index of 40, and three times as much for infrastructure less than the reference index. The cost of programming in Ada, then, is asserted to be more sensitive to infrastructural support than non-Ada programming. For low levels of infrastructure, and especially low levels of experience, Ada is assumed to be proportionately more expensive because of its complexity. There are many features, and learning enough to know which to use to program simply is an obstacle. Also, Ada programs are highly formatted, with many rules to follow just to get something to run correctly. Finally, many of the features that support highly proficient programmers in good practices may be confusing for programmers not accustomed to Ada. Programs using generics, packages, and structures with many nested procedures may do more harm than good for a programmer not accustomed to pieces of programs scattered all over the place.

On the high-infrastructure side, Ada costs are assumed to decline proportionately more than non-Ada costs for a given change in intensity of infrastructure. As usual, the basis is the design of Ada. All of the features that are so troublesome at low intensities of infrastructure become aids in controlling complexity at higher infrastructures. When experienced programmers master the features, the result is programs easier to structure, and easier to debug, since Ada's features support these activities better than most non-Ada languages. Having a complex language designed for experienced programmers implies a cost curve that descends more rapidly than for more typical languages.

Ada and non-Ada projects sectors calibration

Project starts. Total project starts (Total_prj_starts) is specified as an exogenous variable—one that, although time-varying, is not influenced by any variables in the system. Generally, the values were derived by combining the defined costs per project year with the EIA forecasts for total expenditures on software to yield a number of project starts per year (EIA 1980, EIA 1985). Mechanically, this was accomplished by first doing rough pencil-and-paper calculations to get the right magnitudes for the initial non-Ada project levels and the start rate. Then, a separate small model was used to get closer to numbers that would roughly duplicate the EIA figures. Finally, experiments with the model itself were used to derive a curve for starts per year.

The EIA forecast expenditures show smooth exponential-appearing growth, so one might expect the same behavior from project starts. The fitted curve for Total_prj_starts does indeed rise quickly from 1975 to 1980, and proportionately more quickly still from 1980 to 1985. Presumably, this surge resulted from microprocessors being suitable for embedded in a vastly wider group of weapons and systems. The curve flattens out from 1985 to 1990; indeed for simulated behavior to exactly match the EIA forecasts, the number of project starts would have to drop. In other words, just the continued starting of new projects at the 1985 rate is more than enough to explain the rise in the EIA figures. Rather than force the curve downward to fit the EIA numbers for years that haven't happened yet, the number of project starts just flattens out and rises slowly. The EIA figures come from bottom-up estimates of programs now in process, so the numbers would seem to exclude programs not yet approved, which would raise the actual expenditure curve beyond 1985, and thus be closer to the model behavior.

The behavior of Total yearly cost in the model is compared to the EIA forecasts in Appendix B.3 (plot number 19),. As is evident from the plot, the model-generated behavior lies within or close to the envelope defined by the two EIA forecasts. That the model matches the time-series is not remarkable, given that paraemters were adjusted in order for it to do so. But the match does show that the calibration *procedure* works.

The careful reader may have noted that Total yearly cost is measured in constant FY'86 dollars ("real" or "inflation-adjusted" dollars), whereas the EIA 1985-95 MCCR forecast numbers are in current ("inflated") dollars—the two sets of numbers are not really comparable. (This mismatch was not discovered until after the initial calibration and scenario analysis had been performed.) However, this mistake may not throw off the calibration as much as it first may appear. The EIA studies were apparently very conservative in tallying up future MCCR expenditures, for the real (i.e., inflation-adjusted) expenditure stream is nearly flat beyond 1990, presumably representing only those expenditures that are already in the works and therefore reasonably certain. But, as discussed at the beginning of Appendix A.9, the estimates appropriate for this study are not the most conservative, but the most likely estimates. Given the continuing advances of computer capabilities and applications, it would seem that the most likely course of MCCR spending would be continued gradual expansion, qualitatively more like the curve actually used for the calibration, the current dollar-inflated EIA forecast. So although the calibration procedure was based on incomplete information about the data, the outcome of the calibration should remain qualitatively similar when the model is recalibrated.

Language choice sector calibration

At present, the calibration of the language choice sector is described entirely within the equation description.

Ada and non-Ada infrastructure sectors calibration

Normal creation of intensity of Ada and non-Ada infrastructure. The normal creations of intensity of Ada and non-Ada infrastructures were estimated by first deriving the parameter for non-Ada from behavioral requirements, and then arguing for how the corresponding Ada parameter should differ from the non-Ada parameter.

The Normal creation of intensity of non-Ada infrastructure represents an abstract and highly aggregated process, as described in the equation description above. It is not possible to assign a value simply derived from known observations on any of the components of infrastructure, such as the rapidity with which experience changes a programmer's productivity, or the frequency of new Ada-related products introduced. Instead, a value can be derived from other values in the model and the behavior of the index variable, Intensity of non-Ada infrastructure, discussed in Section 5 of this report.

The Intensity of non-Ada infrastructure was defined to rise from a value of 35 in 1975 to 40 in 1985. Infrastructure is assumed to disappear through Obsolescence of Ada infrastructure at a rate of (Intensity of non-Ada infrastructure/Duration of intensity of non-Ada infrastructure) = (35/30) = 1.16. So Creation of intensity of non-Ada infrastructure must be at least 1.16 just to stay even with obsolescence.

But creation must also increase the intensity somewhat -- 5 intensity units between 1975 and 1985, or about 0.5 intensity units per year. The total creation, then, must be about 1.16 + 0.5 = 1.66 intensity units per year.

The heuristic computation just described is not precise. As the Intensity of non-Ada infrastructure increases between 1975 and 1985, the obsolescence likewise rises. Moreover, the inputs to the rate of Creation of non-Ada infrastructure such as Incompatibility of non-Ada infrastructure are not at their reference values during that period, so the creation rate will not equal the normal rate. Incompatibility of non-Ada infrastructure is defined to (and does) rise from its initial value of 40 in 1975 to its reference value of 50 in 1985. To properly account for such effects, the final parameter value was arrived at by simulating the model to ensure that the behavior of the levels matched the behavior implicit in the definitions of initial conditions and reference values. For the Normal creation of intensity of non-Ada infrastructure (Norm_cr_NA_infr), the value that works well turns out to be virtually the same as the value calculated above—1.6. The various effects that might change the value apparently more or less cancelled each other out.

With the normal creation for non-Ada computed, what can be said about the appropriate value for the corresponding Normal creation of intensity of Ada infrastructure? There are no clear reference values from which to compute. Probably the best that can be done is to argue that the Ada parameter should be greater than the non-Ada parameter, based on inherent differences between Ada and non-Ada.

One fundamental of the design of Ada that has had remarkably little disagreement is that Ada supports modern software engineering practices and encourages reuse of code better than any other common language. Ada constructs aid programmers in using structured programming, information hiding, reuse of coding through generics, packages, and so on. Other languages certainly permit such practices, but do not do as much as Ada to facilitate and encourage them. So Ada would seem to facilitate programmer education, which is one kind of creation of infrastructure intensity. Similarly, many languages permit building program libraries, but this process is not facilitated by the design of the language to the same extent it is in Ada; Ada, then, even with all other things being equal, creates

more incentives to create infrastructure. The normal creation of infrastructure should therefore be higher for Ada than for non-Ada languages; Norm_cr_int_A_infr is set at 2.4, or half again as great as for non-Ada.

Normal creation of incompatibility of Ada and non-Ada infrastructure. The computation of the normal creations of incompatibility of Ada and non-Ada infrastructures follows the same general format as the computations for normal creations of intensity: First, a non-Ada parameter value is computed, based on required behavior of the associated level. Then the value is refined through simulation testing. Finally, a value for the Ada parameter is argued on a priori grounds.

Incompatibility of non-Ada infrastructure is defined to rise from its initial 1975 value of 40 to its 1985 reference value of 50. In 1975, Loss of incompatibility of non-Ada infrastructure will be (Incompatibility of non-Ada infrastructure/Normal duration of incompatibility of non-Ada infrastructure) = (40/30) = 1.33 incompatibility units per year. Therefore, the Creation of incompatibility of non-Ada infrastructure must be at least 1.33 just to keep the level constant.

But the level of Incompatibility of non-Ada infrastructure should rise from 40 to 50 in ten years, or about 1.0 incompatibility unit per year. So the initial estimate for Normal creation of incompatibility of non-Ada infrastructure would be 1.33 + 1.0 = 2.33.

As with intensity, the inputs to the Incompatibility of non-Ada infrastructure do not stay constant between 1975 and 1985, so model simulation and parameter-tuning were used to ensure that the model behaved as it ought. The value of Normal creation of incompatibility of non-Ada infrastructure (Norm_cr_inco_NA) that produced the appropriate behavior was 3.0, not terribly far from the original, hand-computed estimate of 2.33.

Should the corresponding parameter for Ada, the Normal creation of incompatibility of Ada infrastructure, be more or less than the non-Ada parameter? The argument seems fairly clear: the Ada portion of the programming world has intrinsically less tendency to create incompatibility, since language is usually the major element in incompatibility and creation of dialects and so on is forbidden to the Ada side, but not the non-Ada side. And the matter of standardized language is the only fixed, fundamental, definitional difference between the Ada and non-Ada classifications. There are no inherent differences in tendency to create incompatibility in operating systems, tools, management styles, or whatever. So the Normal creation of incompatibility of Ada infrastructure (Norm_cr_inco_A_infr) is set at 0.75, one quarter of the non-Ada value.

Opposite polarity of impact on Ada and non-Ada of use and incentives. All three levels of infrastructure -- intensity, incompatibility, and coverage -- for both Ada and non-Ada are impacted by the Fraction of Ada use and Incentives to use Ada. Both use and incentives represent relative measures of Ada versus non-Ada. For Ada, high use and incentives mean high motivation to create infrastructure intensity, incompatibility, and coverage. For non-Ada programming, the opposite is true: high use of Ada and high incentives to use Ada are *disincentives* to accumulate intensity, incompatibility, and coverage. Therefore, the pairs of graphic functions have opposite slopes:

Ada variable has the opposite slope of the non-Ada variable

E_use_int_A_infr E_use_int_NA_infr
E_inc_int_A_infr E_inc_A_int_NA
E_use_inco_A_infr E_A_use_inco_NA_infr
E_inc_inco_A_infr E_inc_A_inco_NA
E_use_cov_A_infr E_use_cov_NA_infr
E_inc_cov_A_infr E_inc_cov_NA_infr

The Ada variables have been described in the equation description. The non-Ada variables all assume values of 1.0 for small values of use or incentives, on the assumption that when conditions work against Ada use, Ada use will be small, and creation of non-Ada infrastructure will not affected by how undesirable Ada use is. When Ada use or Ada incentives are high, creation of non-Ada infrastructure is inhibited, but not as seriously as Ada infrastructure is inhibited by low values, i.e. the slopes of the non-Ada variables are less than their corresponding Ada variables. This represents the assumption that by the time conditions for Ada use become highly favorable, the non-Ada languages will be evolving to fill the specialized needs not met as efffectively by Ada. Ada's potential success several years down the pike is less of a disadvantage to the accumulation non-Ada infrastructure of that time than the current use of non-Ada languages is a disadvantage to accumulation of Ada infrastructure.

APPENDIX B

MODEL LISTING, OUTPUT, AND POLICY LEVERS

Appendix B: Model Listing, Output, and Policy Levers

Appendix B contains supporting materials for the models used in this report: a listing, differences between the base scenario and others, both tabular and plotted output, and a summary of policy levers.

Appendix B.1 contains a complete equation listing of the base model, as described in Appendix A.

Appendix B.2 provides an index for the plots contained in Appendices B.3 through B.5, in the correct order and with full names and equation numbers for reference.

Appendix B.3 contains the output of the base scenario simulation, including both plots, and tables of data.

Appendix B.4 provides all output plots and tables for the scenario representing the adoption of a commercial environment as a standard APSE, and then gives the exact changes to the base model needed to create this scenario.

Appendix B.5 provides all output plots and tables for the scenario representing a policy of converting deployed Non-Ada programs undergoing major redevelopment into Ada, and then gives the exact model changes needed to create this scenario.

Appendix B.6 provides an overview of the policies that could be explored with the model and describes the levers in the model that would be used to represent them.

Appendix B.1: Model Listing

This appendix provides a complete equation listing of the base model as described in Appendix A. This model resides on the Macintosh floppy disks supplied along with this report as STELLA document SSM0.32 (Standardization Scenario Model release 0 version 32). STELLA equation listings are divided into three blocks: all the level variables are listed first, the second block lists the rates and converter variables which do not use graphic functions, and finally, the third block lists all the converter variables which use use graphic functions. Within each block variables are listed alphabetically.

The listing, then, begins with the level variables, symbolized by the small rectangles to the left of the equation proper:

□ Ada_dev_prj = Ada_dev_prj + Ada_dev_starts - Ada_dev_compl
INIT(Ada_dev_prj) = 0 {Ada development projects (projects)}

□ Ada_maint_prj = Ada_maint_prj + Ada_dev_compl - Ada_mn_prj_obsol + Conv_prj_compl
INIT(Ada_maint_prj) = 0 {Ada maintainance projects (projects)}

□ Conv_prj = Conv_prj - Conv_prj_compl + Conv_prj_starts
INIT(Conv_prj) = 0 {Conversion projects (projects)}

□ Cov_Ada_infr = Cov_Ada_infr + Ch_cov_Ada_infr
INIT(Cov_Ada_infr) = 0 {Coverage of Ada infrastructure (dimensionless)}

□ Cov_NA_infr = Cov_NA_infr + Ch_cov_NA_infr
INIT(Cov_NA_infr) = .85
{Coverage of non-Ada infrastructure (dimensionless)}

□ Incom_Ada_infra = Incom_Ada_infra - Loss_inco_Ada_infr + Crea_inco_Ada_infr
INIT(Incom_Ada_infra) = 2 {Incompatibility of Ada infrastructure (incompatability units)}

□ Incom_NA_infr = Incom_NA_infr - Loss_inco_NA_infr + Crea_inco_NA_infr
INIT(Incom_NA_infr) = 40 {Incompatibility of non-Ada infrastructure (incompatibility units)}

□ Inten_Ada_infra = Inten_Ada_infra + Crea_int_Ada_infr - Obsol_int_Ada_inf + Inj_GFE_Ada_infr
INIT(Inten_Ada_infra) = 0
{Intensity of Ada infrastructure (intensity units)}

□ Inten_NA_infra = Inten_NA_infra - Obsol_int_NA_infr + Crea_int_NA_infr
INIT(Inten_NA_infra) = 35
{Intensity of non-Ada infrastructure (intensity units)}

□ NonAda_dev_proj = NonAda_dev_proj + NA_dev_starts - NA_dev_compl
INIT(NonAda_dev_proj) = 150
{NonAda development projects (projects)}

□ NonAda_maint_prj = NonAda_maint_prj + NA_dev_compl -
NA_mn_prj_obsol - Conv_prj_starts
INIT(NonAda_maint_prj) = 90
{NonAda maintenance projects (projects)}

□ Perc_ince_use_Ada = Perc_ince_use_Ada + Ch_per_ince
INIT(Perc_ince_use_Ada) = -2
{Perceived incentive to use Ada (incentive units)}

□ Total_cost = Total_cost + Cst_yr_dis
INIT(Total_cost) = 0 {Total cost (dollars)}

○ Ada_dev_compl = Ada_dev_prj/Ada_dev_compl_time
{Ada development (project) completion time
(projects/year)}

○ Ada_dev_compl_time = 10 {Ada development (project) completion time (
years) }

○ Ada_dev_starts = Total_prj_starts*Fr_dev_starts_Ada
{Ada development (project) starts (projects/year)}

○ Ada_infr_init_inj = 20
{Ada infrastructure inititially injected (infrastructure units)}

○ Ada_mn_prj_obsol = Ada_maint_prj/Ada_prj_mn_time {Ada
maintainance project obsolescence (projects/year)}

○ Ada_prj_mn_time = 20 {Ada project maintainance time (years) -- the
time it takes for the system in which the technology is embedded to pass
out of useful service}

○ Add_GFE_Ada_infr = 0
{Additional GFE'd Ada infrastructure (infrastructure units)}

○ Add_inj_Ada_infr = PULSE(Add_GFE_Ada_infr,T_add_Ada_infr,
Inter_add_Ada_infr)
{Additional inection of Ada infrastructure (infrastructure units/year)}

○ Ch_cov_Ada_infr = (Ind_cov_A_infr-Cov_Ada_infr)/
(T_ch_cov_A_infr * E_pol_t_ch_cov_A)
{Change in coverage of Ada infrastructure (fraction/year)}

○ Ch_cov_NA_infr = (Ind_cov_NA_infr-Cov_NA_infr)/T_ch_cov_NA_infr
{Change in coverage of non-Ada infrastructure
(coverage units/year)}

○ Ch_per_ince = (Incentive_use_Ada - Perc_ince_use_Ada)
/Time_perc_ince
{Change in perceived incentives (to use Ada)
(incentive units/year)}

- Conv_compl_time = 2 {Conversion project completion time (years)}
- Conv_prj_compl = Conv_prj / Conv_compl_time
  {Conversion project completions (projects/year)}
- Conv_prj_starts = NonAda_maint_prj * Fr_conv_NA_mn_pr {Conversion project starts (projects/year)}
- Crea_inco_Ada_infr = Norm_cr_inco_A_infr*E_inc_inco_A_infr* E_int_inco_A_infr*E_use_inco_A_infr*E_pol_inco_A_infr {Creation of incompatibility of Ada infrastructure (incompatibility units/year)}
- Crea_inco_NA_infr = Norm_cr_inco_NA * E_int_inco_NA_infr * E_A_use_inco_NA * E_inc_A_inco_NA
  {Creation of incompatible Nonada infrastructure (incompatibility units/year)}
- Crea_int_Ada_infr = Norm_cr_int_A_infr*E_tech_cr_int* E_inc_int_Ada_infr*E_use_int_A_infr*E_inco_int_A_infr * E_pol_int_A_infr*E_rel_infr_int_Ada
  {Creation. of intens. of Ada infrastr. (infr. units/yr.)}
- Crea_int_NA_infr = Norm_cr_int_NA_infr *E_tech_cr_int* E_use_int_NA_infr * E_inco_int_NA_infr * E_inc_A_int_NA * E_rel_infr_int_NA
  {Creation of intensity of Non-Ada infrastructure (intensity units/year)}
- Cst_prj_yr_Ada_dev = Ref_cst_Ada_dev*E_Ada_cost
  {Cost per project-year of Ada development projects (dollars/project/year)}
- Cst_prj_yr_Ada_mn = Ref_cst_Ada_mn*E_Ada_cost
  {Cost per project-year for Ada maintenance projects (dollars/year)}
- Cst_prj_yr_Conv = Cst_prj_yr_Ada_dev * Ratio_conv_dev_cst
  {Cost per project-year for conversions (dollars/year)}
- Cst_prj_yr_NA_dev = Ref_cst_NA_dev*E_NA_cst
  {Cost per project-year for NonAda development projects (dollars/year)}
- Cst_prj_yr_NA_mn = Ref_cst_NA_mn*E_NA_cst
  {Cost per project-year for NonAda maintenance (dollars/year)}

- Cst_yr_Ada_dev = Ada_dev_prj*Cst_prj_yr_Ada_dev
  {Cost per year for Ada development projects
  (dollars/year)}
- Cst_yr_Ada_mn = Ada_maint_prj*Cst_prj_yr_Ada_mn {Cost per year for Ada maintenace projects (dollars/year)}
- Cst_yr_dis = Total_yearly_cost*Discount_index
  {Cost per year discounted (dollars/year)}
- Cst_Yr_NA_dev = NonAda_dev_proj*Cst_prj_yr_NA_dev
  {Cost per year for NonAda development projects (dollars/year)}
- Cst_yr_NA_mn = NonAda_maint_prj*Cst_prj_yr_NA_mn
  {Cost per year for NonAda maintenance projects (dollars/year)}
- Discount_index = IF TIME ≤ Start_yr_cst_accum THEN 0 ELSE EXP(-Discount_rate*(TIME - Start_yr_cst_accum))
  {Discount index (dimensionless)}
- Discount_rate = 0 {Discount rate (fraction/year)}
- Dura_int_Ada_infr = 30 {Duration of intensity of Ada infrastructure (years)}
- Dura_int_NA_infr = 30 {Duration of intensity of NonAda infrastructure (years)}
- E_Ada_cost = E_int_Ada_cst*E_inco_A_cst
  {Effects on Ada costs (dimensionless)}
- E_ind_cov_A_infr = E_inco_cov_A_infr*E_inc_cov_A_infr*
  E_use_cov_A_infr {Effects on indicated coverage of Ada infrastructure}
- E_NA_cst = E_int_NA_cst*E_inco_NA_cst
  {Effects on NonAda costs (dimensionless)}
- E_tech_cr_int = EXP((year - Start_yr_cst_accum) * Rate_tech_prog) {Effect of technology on creation of
  intensity of infrastructure (dimensionless)}
- Fr_Ada_prj = Total_Ada_prj/Total_projects
  {Fraction of Ada projects (dimensionless)}
- Fr_dev_starts_Ada = Nat_fr_Ada_starts * E_target_on_starts
  {Fraction of development (project) starts in Ada
  (dimensionless)}
- Incentive_use_Ada = Ince_rel_int_infr + Ince_rel_cov_infr + Ince_pol
  {Incentives to use Ada (incentive units)}
- Ind_cov_A_infr = IF (Sw_transp_A_cov=1) THEN 1 ELSE E_ind_cov_A_infr {Indicated coverage of infrastructure units (infrastructure units)}
- Ind_cov_NA_infr = E_use_cov_NA_infr*E_inc_cov_NA_infr*
  E_inco_cov_NA_infr {Indicated coverage of
  non-Ada infrastructure (dimensionless)}
- Init_inj_Ada_infr = PULSE(Ada_infr_Init_inj,1981,1e11)
  {Initial injection of Ada infrastructure (infrastructure units/year); the one-time construction of a few initial compilers, loaders, etc.}

- Inj_GFE_Ada_infr = Init_inj_Ada_infr + Add_inj_Ada_infr
  {Injection of GFE (government-furnished equipment) for Ada infrastructure (infrastructure units/year)}
- Inter_add_Ada_infr = 1e11 {Interval to add Ada infrastructure (years)}
- Loss_inco_Ada_infr = Incom_Ada_infra/
  (Norm_dur_inco_A_inf * E_pol_du_inco_A_inf)
  {Loss of incompatable Ada infrastructure (incompatability units/year)}
- Loss_inco_NA_infr = Incom_NA_infr / Norm_dur_incom_NA {Loss of incompatibility of non-
  Ada infrastructure (incompatibility units/year)}
- NA_dev_compl = NonAda_dev_proj/NA_dev_compl_time
  {Non-Ada development (project) completions (projects/year)}
- NA_dev_compl_time = 10
  {NonAda development (project) completion time (years)}
- NA_dev_starts = Total_prj_starts * (1-Fr_dev_starts_Ada)
  {Non-Ada development (project) starts (projects/year)}
- NA_mn_prj_obsol = NonAda_maint_prj/NA_proj_mn_time {NonAda maintainance project obsolescence (projects/year)}
- NA_proj_mn_time = 20 {NonAda project maintainance time (years)}
- Norm_cr_inco_A_infr = .75
  {Normal creation of incompatibility of Ada infrastructure (fraction)}
- Norm_cr_inco_NA = 3 {Normal creation of incompatibility of non-Ada infrastructure (incompatibility units/year)}
- Norm_cr_int_A_infr = 2.4
  {Normal creation of intensity of Ada infrastructure (intensity units/year)}
- Norm_cr_int_NA_infr = 1.6
  {Normal creation of intensity of non-Ada infrastructure (intensity units/year)}
- Norm_dur_incom_NA = 30 {Normal duration of incompatibility of non-Ada infrastructure (years)}
- Norm_dur_inco_A_inf = 30 {Normal duration of incompatibility of Ada infrastructure (years)}
- Obsol_int_Ada_inf = Inten_Ada_infra/Dura_int_Ada_infr
  {Obsolesence of intensity of Ada infrastructure (units/year)}
- Obsol_int_NA_infr = Inten_NA_infra / Dura_int_NA_infr
  {Obsolescence of intensity of non-Ada infrastructure (intensity units/year)}

- Rate_tech_prog = 0
  {Rate of technological progress (fraction/year)}
- Ratio_conv_dev_cst = 1
  {Ratio of conversion to development costs (dimensionless)}
- Ref_cst_Ada_dev = 5E6
  {Reference cost for Ada development projects
  (dollars/project/year)}
- Ref_cst_Ada_mn = 2e6 {Reference cost for Ada maintenance projects (Dollars/project/year)}
- Ref_cst_NA_dev = 6E6 {Reference cost for NonAda development projects (dollars/year)}
- Ref_cst_NA_mn = 3e6 {Reference cost for Non Ada maintenance projects (dollars/year)}
- Rel_cov_Ada_infr = Cov_Ada_infr/Cov_NA_infr
  {Relative coverage of Ada infrastructure (dimensionless)}
- Rel_int_Ada_infr = Inten_Ada_infra/Inten_NA_infra
  {Relative intensity of Ada infrastructure (dimensionless)}
- Start_yr_cst_accum = 1986
  {Starting year for cost accumulation (year)}
- Target_Ada_rel_nat = Targ_fr_Ada_starts/Nat_fr_Ada_starts {Target for Ada (starts) relative to natural (fraction) (fraction)}
- Time_perc_ince = 2 {Time to perceive incentives (to use Ada) (years)}
- Total_Ada_prj = Ada_dev_prj+(Ada_maint_prj*W_Ada_mn_prj)+(Conv_prj W_conv_prj)
  {Total Ada projects (projects)}
- Total_NA_prj = NonAda_dev_proj+((NonAda_maint_prj-Conv_prj)* w_NA_mn_prj)
  {Total NonAda projects (projects)}
- Total_projects = Total_Ada_prj+Total_NA_prj {Total projects (projects)}
- Total_yearly_cost = Tot_yr_cst_NA+Tot_yr_cst_conv+Tot_yr_cst_Ada { Total yearly cost (dollars/year)}
- Tot_yr_cst_Ada = Cst_yr_Ada_mn+Cst_yr_Ada_dev
  {Total yearly cost of Ada projects (dollars/year)}
- Tot_yr_cst_conv = Conv_prj*Cst_prj_yr_Conv
  {Total yearly cost conversions (dollars/year)}
- Tot_yr_cst_NA = Cst_yr_NA_mn+Cst_Yr_NA_dev
  {Total yearly cost of Nonada projects (dollars/year)}
- T_add_Ada_infr = 1990
  {Time for additional Ada infrastructure (year)}
- T_ch_cov_A_infr = 5
  {Time to change coverage of Ada infrastructure (years)}

○ T_ch_cov_NA_infr = 5 {Time to change converage of
non-Ada infrastructure (years)}
○ W_Ada_mn_prj = .5 {Weight for Ada maintainance projects (dimensionless)
○ W_conv_prj = 1 {Weight for conversion upgrades
(dimensionless)}
○ w_NA_mn_prj = .5
○ year = TIME {years (years)}
⊘ 1990.000 -> 3.210e+10
⊘ EIA_851995.000 -> 3.566e+10
⊘ E_A_use_inco_NA = graph(Fr_Ada_prj)
0.0 -> 0.995
0.100 -> 0.985
0.200 -> 0.955
0.300 -> 0.930
0.400 -> 0.885
0.500 -> 0.780
0.600 -> 0.700
0.700 -> 0.630
0.800 -> 0.570
0.900 -> 0.530
1.000 -> 0.500
⊘ E_inco_A_cst = graph(Incom_Ada_infra)
0.0 -> 0.600
10.000 -> 0.630
20.000 -> 0.650
30.000 -> 0.720
40.000 -> 0.830
50.000 -> 1.000
60.000 -> 1.190
70.000 -> 1.350
80.000 -> 1.540
90.000 -> 1.650
100.000 -> 1.730

E_inco_cov_A_infr = graph(Incom_Ada_infra)
0.0 -> 1.000
10.000 -> 0.975
20.000 -> 0.940
30.000 -> 0.900
40.000 -> 0.870
50.000 -> 0.850
60.000 -> 0.830
70.000 -> 0.820
80.000 -> 0.805
90.000 -> 0.795
100.000 -> 0.785

E_inco_cov_NA_infr = graph(Incom_NA_infr)
0.0 -> 1.000
10.000 -> 0.975
20.000 -> 0.940
30.000 -> 0.900
40.000 -> 0.870
50.000 -> 0.850
60.000 -> 0.830
70.000 -> 0.820
80.000 -> 0.805
90.000 -> 0.795
100.000 -> 0.785

E_inco_A_cst = graph(Incom_Ada_infra)
0.0 -> 0.600
10.000 -> 0.630
20.000 -> 0.650
30.000 -> 0.720
40.000 -> 0.830
50.000 -> 1.000
60.000 -> 1.190
70.000 -> 1.350
80.000 -> 1.540
90.000 -> 1.650
100.000 -> 1.730

E_inco_cov_A_infr = graph(Incom_Ada_infra)
0.0 -> 1.000
10.000 -> 0.975
20.000 -> 0.940
30.000 -> 0.900
40.000 -> 0.870
50.000 -> 0.850
60.000 -> 0.830
70.000 -> 0.820
80.000 -> 0.805
90.000 -> 0.795
100.000 -> 0.785

E_inco_cov_NA_infr = graph(Incom_NA_infr)
0.0 -> 1.000
10.000 -> 0.975
20.000 -> 0.940
30.000 -> 0.900
40.000 -> 0.870
50.000 -> 0.850
60.000 -> 0.830
70.000 -> 0.820
80.000 -> 0.805
90.000 -> 0.795
100.000 -> 0.785

E_inco_int_A_infr = graph(Incom_Ada_infra)
0.0 -> 3.000
10.000 -> 2.415
20.000 -> 1.965
30.000 -> 1.635
40.000 -> 1.290
50.000 -> 1.000
60.000 -> 0.675
70.000 -> 0.450
80.000 -> 0.285
90.000 -> 0.180
100.000 -> 0.135

⊘ E_inco_Int_NA_Infr = graph(Incom_NA_Infr)
0.0 -> 3.000
10.000 -> 2.415
20.000 -> 1.965
30.000 -> 1.635
40.000 -> 1.290
50.000 -> 1.000
60.000 -> 0.675
70.000 -> 0.450
80.000 -> 0.285
90.000 -> 0.180
100.000 -> 0.135

⊘ E_inco_NA_cst = graph(Incom_NA_Infr)
0.0 -> 0.600
10.000 -> 0.630
20.000 -> 0.650
30.000 -> 0.720
40.000 -> 0.830
50.000 -> 1.000
60.000 -> 1.190
70.000 -> 1.350
80.000 -> 1.540
90.000 -> 1.650
100.000 -> 1.730

⊘ E_inc_A_inco_NA = graph(Perc_ince_use_Ada)
-1.000 -> 1.000
-0.800 -> 1.000
-0.600 -> 1.000
-0.400 -> 1.000
-0.200 -> 1.000
0.0 -> 1.000
0.200 -> 0.990
0.400 -> 0.970
0.600 -> 0.930
0.800 -> 0.890
1.000 -> 0.820

⊘ E_inc_A_int_NA = graph(Perc_ince_use_Ada)
-1.000 -> 1.120
-0.800 -> 1.110
-0.600 -> 1.100
-0.400 -> 1.070
-0.200 -> 1.050
0.0 -> 1.000
0.200 -> 0.920
0.400 -> 0.840
0.600 -> 0.750
0.800 -> 0.680
1.000 -> 0.600

⊘ E_inc_cov_A_infr = graph(Perc_ince_use_Ada)
-1.000 -> 0.440
-0.800 -> 0.655
-0.600 -> 0.780
-0.400 -> 0.875
-0.200 -> 0.965
0.0 -> 1.000
0.200 -> 1.000
0.400 -> 1.000
0.600 -> 1.000
0.800 -> 1.000
1.000 -> 1.000

⊘ E_inc_cov_NA_infr = graph(Perc_ince_use_Ada)
-1.000 -> 1.000
-0.800 -> 1.000
-0.600 -> 1.000
-0.400 -> 1.000
-0.200 -> 1.000
0.0 -> 1.000
0.200 -> 0.980
0.400 -> 0.910
0.600 -> 0.870
0.800 -> 0.830
1.000 -> 0.800

⊘ E_inc_inco_A_infr = graph(Perc_ince_use_Ada)
-1.000 -> 0.240
-0.800 -> 0.290
-0.600 -> 0.360
-0.400 -> 0.450
-0.200 -> 0.620
0.0 -> 1.000
0.200 -> 1.280
0.400 -> 1.530
0.600 -> 1.700
0.800 -> 1.800
1.000 -> 1.900

⊘ E_inc_int_Ada_infr = graph(Perc_ince_use_Ada)
-1.000 -> 0.380
-0.800 -> 0.400
-0.600 -> 0.460
-0.400 -> 0.560
-0.200 -> 0.700
0.0 -> 1.000
0.200 -> 1.380
0.400 -> 1.680
0.600 -> 1.900
0.800 -> 2.000
1.000 -> 2.020

⊘ E_int_Ada_cst = graph(Inten_Ada_infra)
0.0 -> 3.000
10.000 -> 2.850
20.000 -> 2.565
30.000 -> 2.010
40.000 -> 1.000
50.000 -> 0.600
60.000 -> 0.465
70.000 -> 0.360
80.000 -> 0.270
90.000 -> 0.200
100.000 -> 0.100

E_int_inco_A_infr = graph(Inten_Ada_infra)
0.0 -> 1.000
20.000 -> 0.980
40.000 -> 0.950
60.000 -> 0.910
80.000 -> 0.840
100.000 -> 0.735
120.000 -> 0.640
140.000 -> 0.550
160.000 -> 0.470
180.000 -> 0.435
200.000 -> 0.410

E_int_inco_NA_infr = graph(Inten_NA_infra)
0.0 -> 1.000
20.000 -> 0.980
40.000 -> 0.850
60.000 -> 0.800
80.000 -> 0.650
100.000 -> 0.560
120.000 -> 0.510
140.000 -> 0.445
160.000 -> 0.425
180.000 -> 0.405
200.000 -> 0.400

E_int_NA_cst = graph(Inten_NA_infra)
0.0 -> 1.970
10.000 -> 1.530
20.000 -> 1.270
30.000 -> 1.120
40.000 -> 1.000
50.000 -> 0.880
60.000 -> 0.760
70.000 -> 0.630
80.000 -> 0.520
90.000 -> 0.390
100.000 -> 0.300

⊘ E_pol_du_inco_A_inf = graph(year)
1970.000 -> 1.000
1975.000 -> 1.000
1980.000 -> 1.000
1985.000 -> 1.000
1990.000 -> 1.000
1995.000 -> 1.000
2000.000 -> 1.000
2005.000 -> 1.000
2010.000 -> 1.000
2015.000 -> 1.000
2020.000 -> 1.000

⊘ E_pol_inco_A_infr = graph(year)
1970.000 -> 1.000
1975.000 -> 1.000
1980.000 -> 1.000
1985.000 -> 1.000
1990.000 -> 1.000
1995.000 -> 1.000
2000.000 -> 1.000
2005.000 -> 1.000
2010.000 -> 1.000
2015.000 -> 1.000
2020.000 -> 1.000

⊘ E_pol_int_A_infr = graph(year)
1970.000 -> 1.000
1975.000 -> 1.000
1980.000 -> 1.000
1985.000 -> 1.000
1990.000 -> 1.000
1995.000 -> 1.000
2000.000 -> 1.000
2005.000 -> 1.000
2010.000 -> 1.000
2015.000 -> 1.000
2020.000 -> 1.000

E_pol_t_ch_cov_A = graph(year)
1970.000 -> 1.000
1975.000 -> 1.000
1980.000 -> 1.000
1985.000 -> 1.000
1990.000 -> 1.000
1995.000 -> 1.000
2000.000 -> 1.000
2005.000 -> 1.000
2010.000 -> 1.000
2015.000 -> 1.000
2020.000 -> 1.000

E_rel_infr_int_Ada = graph(Rel_int_Ada_infr)
0.0 -> 1.850
0.200 -> 1.740
0.400 -> 1.510
0.600 -> 1.240
0.800 -> 1.080
1.000 -> 1.000
1.200 -> 0.940
1.400 -> 0.900
1.600 -> 0.890
1.800 -> 0.870
2.000 -> 0.860

E_target_on_starts = graph(Target_Ada_rel_nat)
0.0 -> 1.000
0.500 -> 1.000
1.000 -> 1.000
1.500 -> 1.125
2.000 -> 1.325
2.500 -> 1.575
3.000 -> 1.950
3.500 -> 2.300
4.000 -> 2.675
4.500 -> 3.075
5.000 -> 3.500

⊘ E_use_cov_A_infr = graph(Fr_Ada_prj)
0.0 -> 0.0
0.100 -> 0.895
0.200 -> 0.970
0.300 -> 0.975
0.400 -> 0.980
0.500 -> 0.985
0.600 -> 0.990
0.700 -> 0.995
0.800 -> 1.000
0.900 -> 1.000
1.000 -> 1.000

⊘ E_use_cov_NA_infr = graph(Fr_Ada_prj)
0.0 -> 1.000
0.100 -> 1.000
0.200 -> 1.000
0.300 -> 0.990
0.400 -> 0.975
0.500 -> 0.965
0.600 -> 0.935
0.700 -> 0.880
0.800 -> 0.835
0.900 -> 0.710
1.000 -> 0.500

⊘ E_use_inco_A_infr = graph(Fr_Ada_prj)
0.0 -> 0.405
0.100 -> 0.445
0.200 -> 0.510
0.300 -> 0.655
0.400 -> 0.750
0.500 -> 0.815
0.600 -> 0.870
0.700 -> 0.905
0.800 -> 0.945
0.900 -> 0.980
1.000 -> 0.995

E_use_int_A_infr = graph(Fr_Ada_prj)
0.0 -> 0.050
0.100 -> 0.500
0.200 -> 0.770
0.300 -> 0.870
0.400 -> 0.920
0.500 -> 0.945
0.600 -> 0.965
0.700 -> 0.985
0.800 -> 0.990
0.900 -> 1.000
1.000 -> 1.000

E_use_int_NA_infr = graph(Fr_Ada_prj)
0.0 -> 1.000
0.100 -> 1.000
0.200 -> 0.990
0.300 -> 0.975
0.400 -> 0.965
0.500 -> 0.935
0.600 -> 0.915
0.700 -> 0.885
0.800 -> 0.830
0.900 -> 0.715
1.000 -> 0.500

Fr_conv_NA_mn_pr = graph(year)
1970.000 -> 0.0
1975.000 -> 0.0
1980.000 -> 0.0
1985.000 -> 0.0
1990.000 -> 0.0
1995.000 -> 0.0
2000.000 -> 0.0
2005.000 -> 0.0
2010.000 -> 0.0
2015.000 -> 0.0
2020.000 -> 0.0

⊘ Ince_pol = graph(year)
1975.000 -> 0.0
1979.000 -> 0.0
1983.000 -> 0.0
1987.000 -> 0.0
1991.000 -> 0.0
1995.000 -> 0.0
1999.000 -> 0.0
2003.000 -> 0.0
2007.000 -> 0.0
2011.000 -> 0.0
2015.000 -> 0.0

⊘ Ince_rel_cov_infr = graph(Rel_cov_Ada_infr)
0.0 -> -0.310
0.200 -> -0.215
0.400 -> -0.140
0.600 -> -0.080
0.800 -> -0.015
1.000 -> 0.050
1.200 -> 0.095
1.400 -> 0.125
1.600 -> 0.160
1.800 -> 0.180
2.000 -> 0.190

⊘ Ince_rel_int_infr = graph(Rel_int_Ada_infr)
0.0 -> -0.530
0.200 -> -0.400
0.400 -> -0.290
0.600 -> -0.190
0.800 -> -0.100
1.000 -> 0.0
1.200 -> 0.160
1.400 -> 0.340
1.600 -> 0.540
1.800 -> 0.720
2.000 -> 0.970

⊘ Nat_fr_Ada_starts = graph(Perc_Ince_use_Ada)
-1.000 -> 5.000e-3
-0.800 -> 0.025
-0.600 -> 0.055
-0.400 -> 0.120
-0.200 -> 0.260
0.0 -> 0.500
0.200 -> 0.680
0.400 -> 0.780
0.600 -> 0.825
0.800 -> 0.870
1.000 -> 0.910

⊘ Sw_transp_A_cov = graph(year)
1970.000 -> 0.0
1975.000 -> 0.0
1980.000 -> 0.0
1985.000 -> 0.0
1990.000 -> 0.0
1995.000 -> 0.0
2000.000 -> 0.0
2005.000 -> 0.0
2010.000 -> 0.0
2015.000 -> 0.0
2020.000 -> 0.0

⊘ Targ_fr_Ada_starts = graph(year)
1970.000 -> 0.0
1975.000 -> 0.0
1980.000 -> 0.0
1985.000 -> 0.0
1990.000 -> 0.500
1995.000 -> 0.500
2000.000 -> 0.500
2005.000 -> 0.500
2010.000 -> 0.500
2015.000 -> 0.500
2020.000 -> 0.500

Total_prj_starts = graph(year)
1970.000 -> 15.000
1975.000 -> 50.000
1980.000 -> 120.000
1985.000 -> 620.000
1990.000 -> 760.000
1995.000 -> 1020.000
2000.000 -> 1390.000
2005.000 -> 1915.000
2010.000 -> 2360.000
2015.000 -> 2870.000
2020.000 -> 3380.000

Appendix B.2: Guide to Variables Plotted

This appendix provides a guide to the multiple pages of simulation plots in Appendices B.3 through B.5. The plots in those appendices are numbered identically to the listing below, although the captions of plots are not given there for space reasons. The first column gives the variable names as they appear above the respective plots. The second column gives the full name of the variable and the equation number. The latter can be used to find quickly a description of the variable in Appendix A.

Plot 1. Four-curve briefing summary

Total_yearly_cost	Total yearly cost, #50
Tot_Ada_prj	Total Ada projects, #770
Conv_prj	Conversion projects, #600
Inten_Ada_infra	Intensity of Ada infrastructure, #1350

Plot 2. Infrastructure summary

Inten_Ada_infra	Intensity of Ada infrastructure, #1350
Inten_NA_infra	Intensity of non-Ada infrastructure, #2000
Incom_Ada_infra	Incompatibility of Ada infrastructure, #1600
Incom_NA_infra	Incompatibility of non-Ada infrastructure, #2150

Plot 3. Infrastructure and cost summary

Cov_Ada_infr	Coverage of Ada infrastructure, #1800
Cov_NA_infra	Coverage of non-Ada infrastructure, #2300
Cst_yr_Ada_dev	Cost per year for Ada development projects, #90
Cst_yr_NA_dev	Cost per year for non-Ada development projects, #270

Plot 4. Fraction project starts in Ada

Fr_dev_starts_Ada	Fraction of development project starts in Ada, #1000
Targ_fr_Ada_starts	Target fraction for Ada starts, #1030
Nat_fr_Ada_starts	Natural fraction of Ada starts, #1040
Incentive_use_Ada	Incentives to use Ada, #1200

Plot 5. Incentives to use Ada

Incentive_use_Ada	Incentives to use Ada, #1200
Ince_rel_cov_infr	Incentive from relative coverage of infrastructure, #1230
Ince_rel_int_infr	Incentive from relative intensity of infrastructure, #1210
Ince_pol	Incentive from policy, #1270

Plot 6. Projects

Ada_dev_proj	Ada development projects, #500
Non-Ada_dev_proj	Non-ada development projects, #820
Ada_maint_prj	Ada maintenance projects, #700
Non-Ada maint_prj	Non-ada maintenance projects, #880

Plot 7. Intensity of Ada infrastructure

E_rel_infr_int_A	Effect of relative infrastructure on intensity of Ada infrastructure, #1540
E_inc_int_A_infr	Effect of incentives on intensity of Ada infrastructure, #1530
E_inco_int_A_infr	Effect of incompatibility on intensity of Ada infrastructure, #1520
E_use_int_A_infr	Effect of relative use on intensity of Ada infrastructure, #1550

Plot 8. Intensity of non-Ada infrastructure

E_rel_infr_int_NA	Effect of relative infrastructure on intensity of non-Ada infrastructure, #2110
E_inco_int_NA_infr	Effect of incompatibility on intensity of non-Ada infrastructure, #2070
E_use_int_NA_infr	Effect of use on intensity of non-Ada infrastructure, #2090
E_inc_A_int_NA	Effect of incentives for Ada use on intensity of non-Ada infrastructure, #2080

Plot 9. Incompatibility of Ada infrastructure

E_use_inco_A_infr	Effect of use on incompatibility of Ada infrastructure, #1680
E_inc_inco_A_infr	Effect of incentives on incompatibility of Ada infrastructure, #1690
E_int_inco_A_infr	Effect of intensity on incompatibility of Ada infrastructure, #1710
E_pol_inco_A_infr	Effect of policy on incompatibility of Ada infrastructure, #1720

Plot 10. Incompatibility of non-Ada infrastructure

E_A_use_inco_NA	Effect of Ada use on incompatibility of non-Ada infrastructure, #2220
E_inc_A_inco_NA	Effect of incentives for Ada use on incompatibility of non-Ada infrastructure, #2210
E_int_inco_NA_infr	Effect of intensity on incompatibility of non-Ada infrastructure, #2230
Crea_inco_NA_infr	Creation of incompatibility of non-Ada infrastructure, #2180

Plot 11. Coverage of Ada infrastructure

Ind_cov_A_infr	Indicated coverage of Ada infrastructure, #1840
E_use_cov_A_infr	Effect of relative use on coverage of Ada infrastructure, #1890
E_inc_cov_A_infr	Effect of incentives on coverage of Ada infrastructure, #1910
E_inco_cov_A_infr	Effect of incompatibility of coverage of Ada infrastructure, #1920

Plot 12. Coverage of non-Ada infrastructure

Ind_cov_NA_infr	Indicated coverage of non-Ada infrastructure, #2330
E_use_cov_NA_infr	Effect of relative use on coverage of non-Ada infrastructure, #2340
E_inc_cov_NA_infr	Effect of incentives on coverage of non-Ada infrastructure, #2350.
E_inco_cov_NA_infr	Effect of incompatibility on coverage of non-Ada infrastructure, #2360

Plot 13. Incentives delay and technology

Incentive_use_Ada	Incentives to use Ada, #1200.
Perc_ince_use_Ada	Perceived incentives to use Ada, #1100
E_tech_cr_int	Effect of technology on creation of intensity, 1570
Fr_Ada_prj	Fraction of Ada projects, #750

Plot 14. Project starts and conversion projects

Conv_prj	Conversion projects, #600
Conv_prj_starts	Conversion project starts, #650
Ada_dev_starts	Ada development project starts, #510
NA_dev_starts	Non-Ada development project starts, #830

Plot 15. Total cost

Total Cost	Total_cost, #10
Total_yearly_cost	Total yearly cost, #50
Cst_yr_dis	Cost per year discounted, #20
Discount_index	Discount index, #30.

Plot 16. Costs per project-year

Cst_prj_yr_Ada_dev	Cost per project-year for Ada development projects, #100
Cst_prj_yr_NA_dev	Cost per project-year for non-Ada development projects, #280
Cst_prj_yr_Ada_mn	Cost per project-year for Ada maintenance projects, #210
Cst_prj_yr_NA_mn	Cost per project-year for non-Ada maintenance projects, #330

Plot 17. Ada costs

Tot_yr_cst_Ada	Total yearly cost of Ada projects, #80
E_int_Ada_cst	Effect of intensity of infrastructure on Ada project costs, #150
E_inco_A_cst	Effect of incompatibility of infrastructure on Ada project costs, #160.

Plot 18. Non-Ada costs

Tot_yr_cst_NA	Total yearly cost of non-Ada projects, #260
E_int_NA_cst	Effect of intensity of infrastructure on non-Ada cost, #360
E_inco_NA_cst	Effect of incompatibility of infrastructure on non-Ada cost, #370

Plot 19. EIA forecasts

Total_yearly_cost	Total yearly cost, #50
EIA_ECR_forec	EIA 1980-1990 embedded computer resources forecast, #60
EIA_MCCR_forec	EIA 1985-1995 mission-critical computer resources forecast, #70

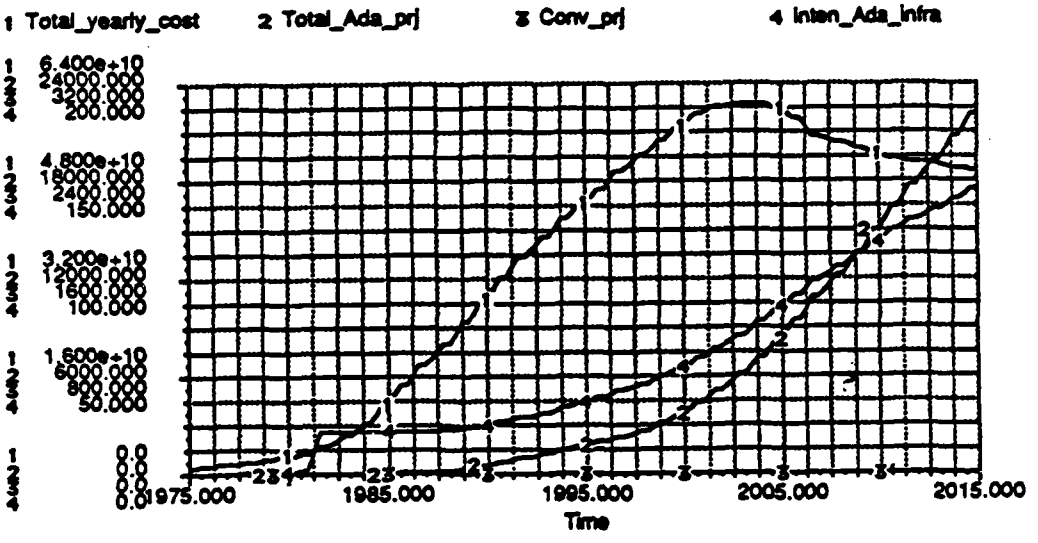
Plot 20. Two-curve briefing summary

Inten_Ada_infra	Intensity of Ada infrastructure, #1350
Total_yearly_cost	Total yearly cost, #50

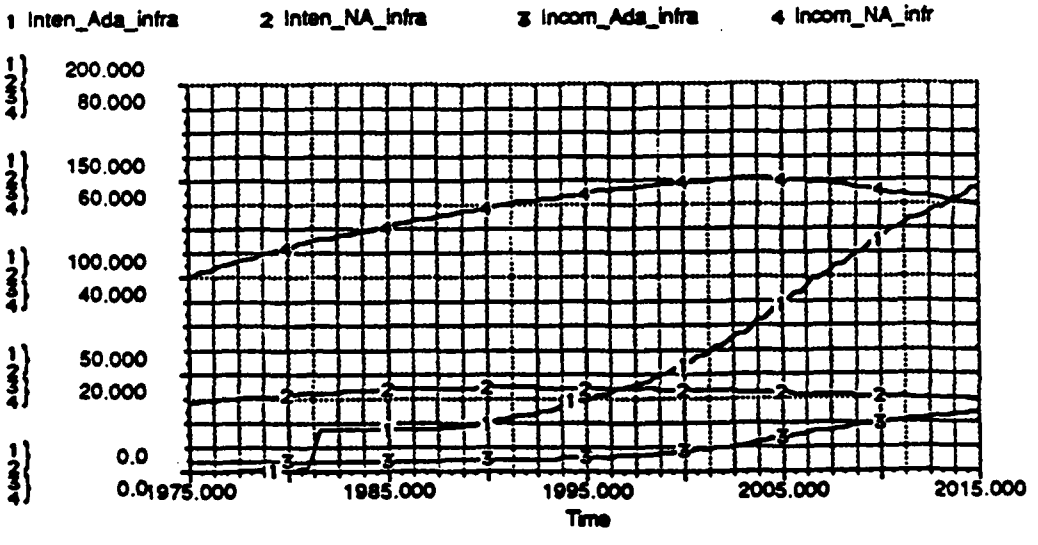
Appendix B.3: Complete Output for Base Scenario

This Appendix contains all of the output plots and tables for the base scenario produced by the Standardization Scenario Model, SSM0.32. That model as stored on the Macintosh floppy disk contains the plots and tables shown below, so they are available "on-line" as well. No changes to the equations listed in Appendix B.1 are required to produce this base scenario. Appendix B.2 "Guide to Variables Plotted" provides the full names and equation numbers for the plotted variables.

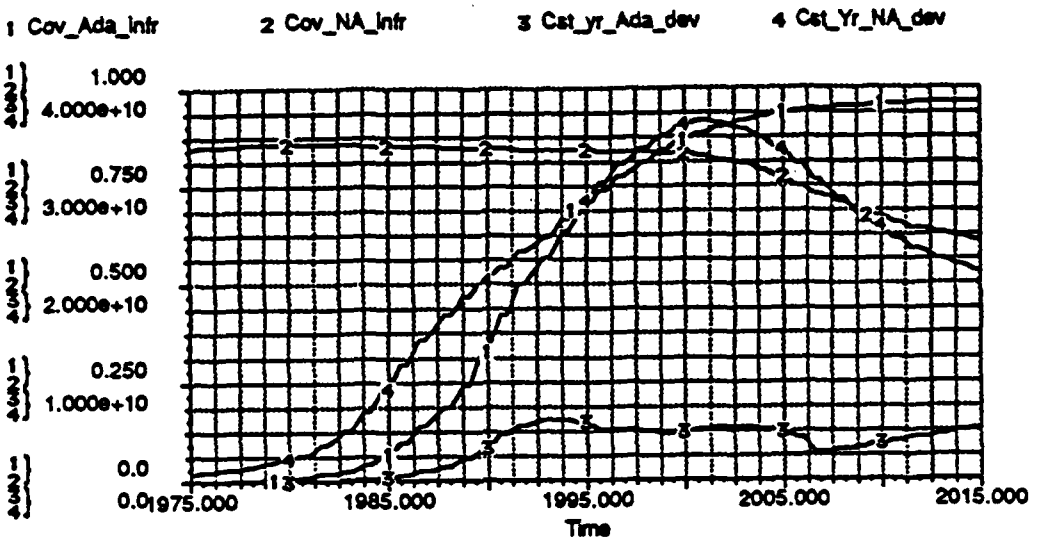
Plot 1



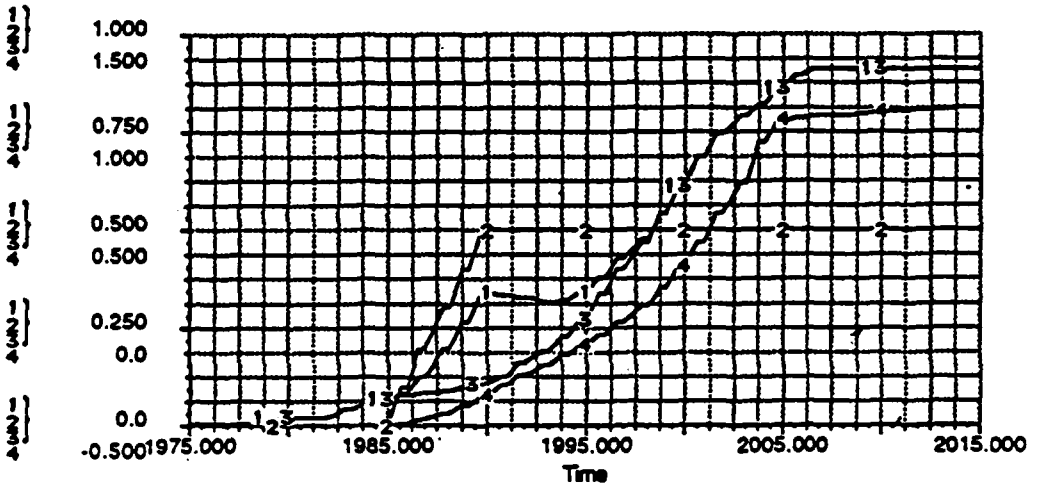
Plot 2



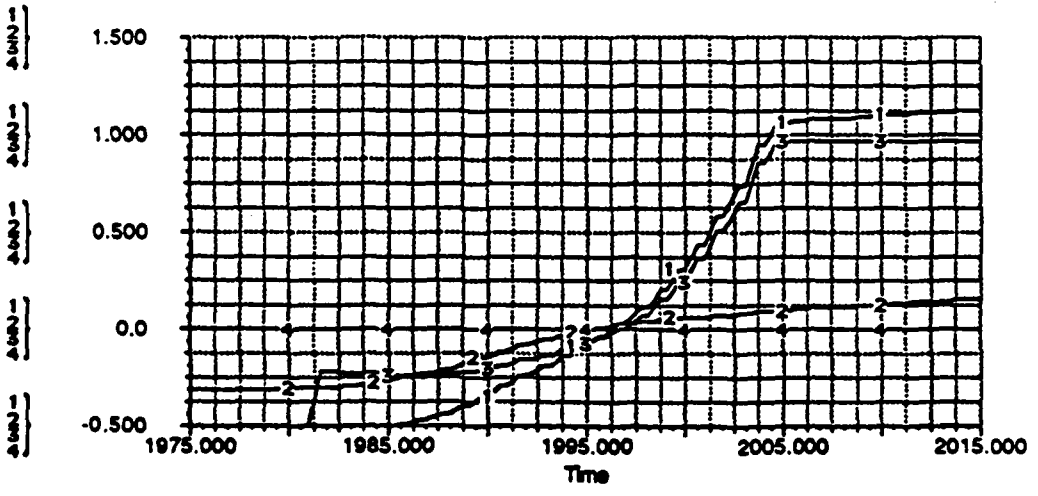
Plot 3



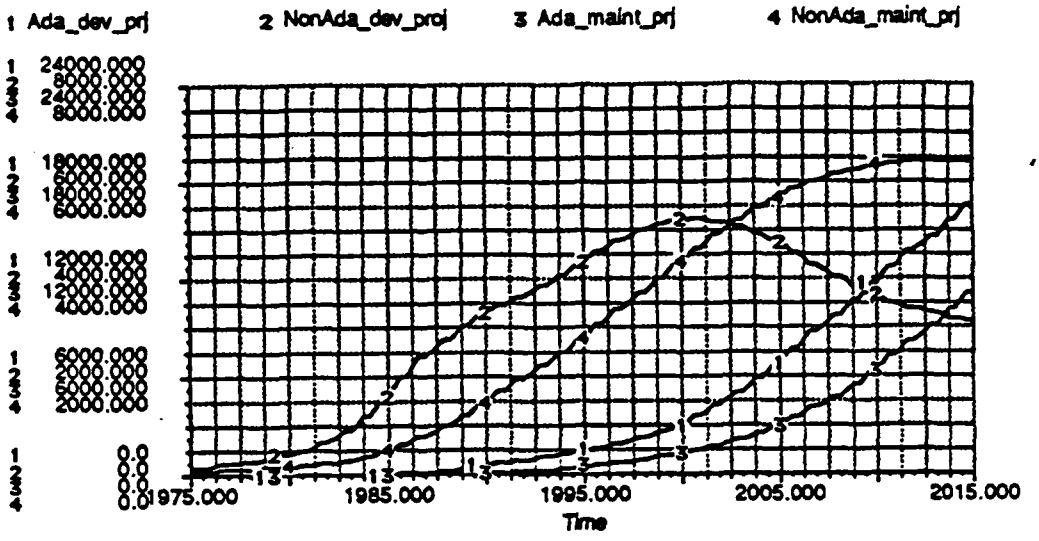
Plot 4



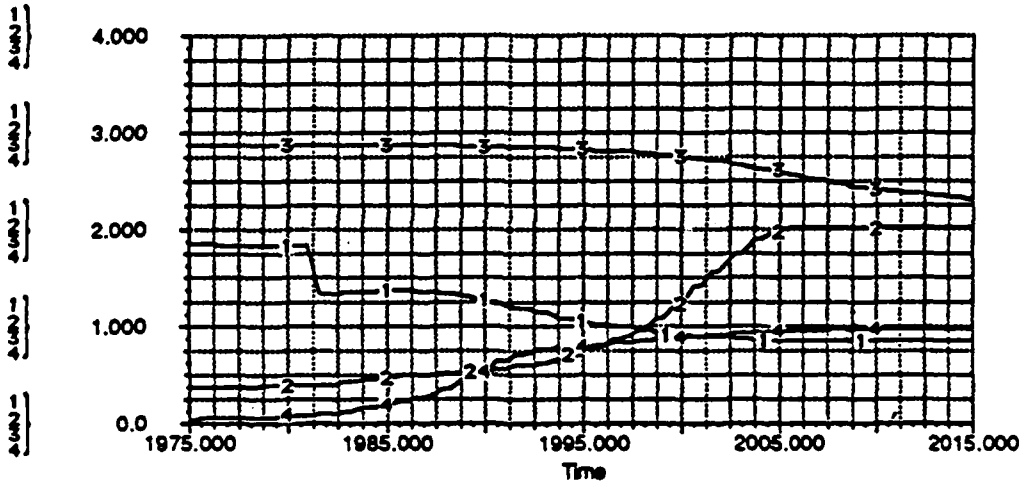
Plot 5



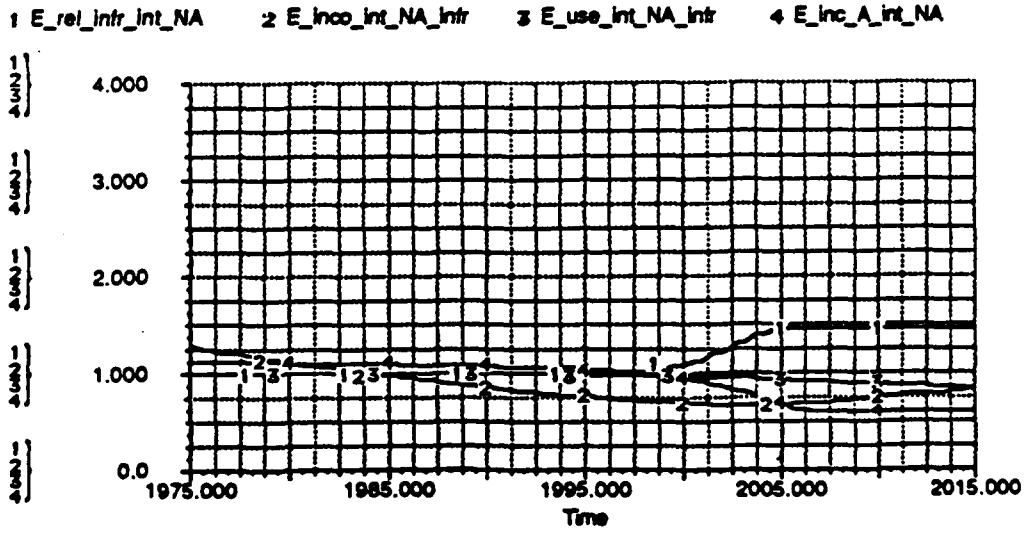
Plot 6



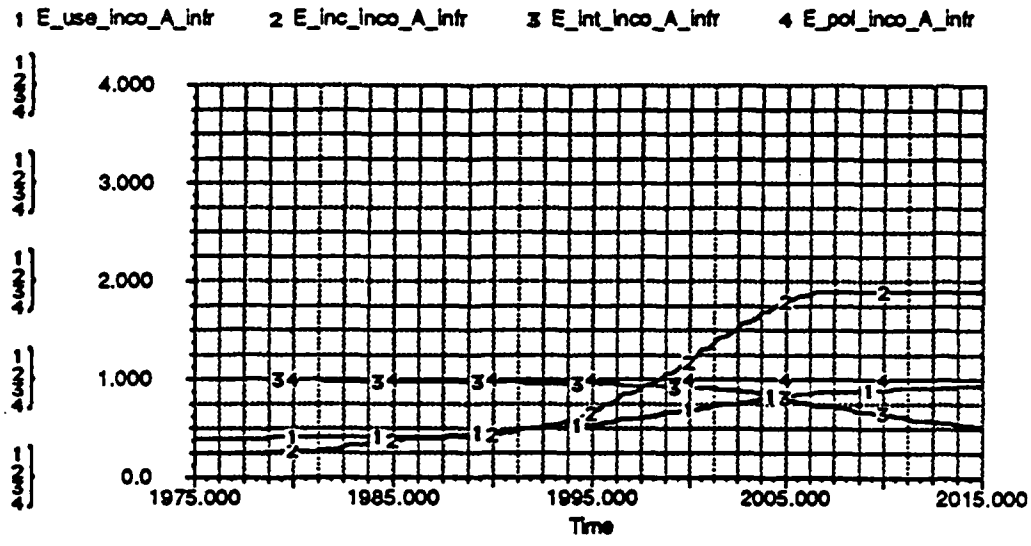
Plot 7



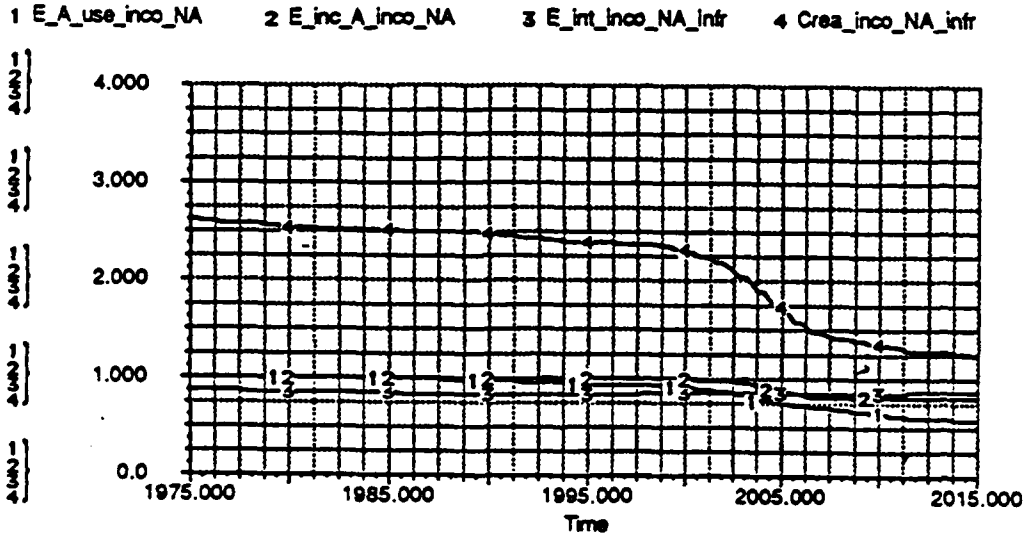
Plot 8



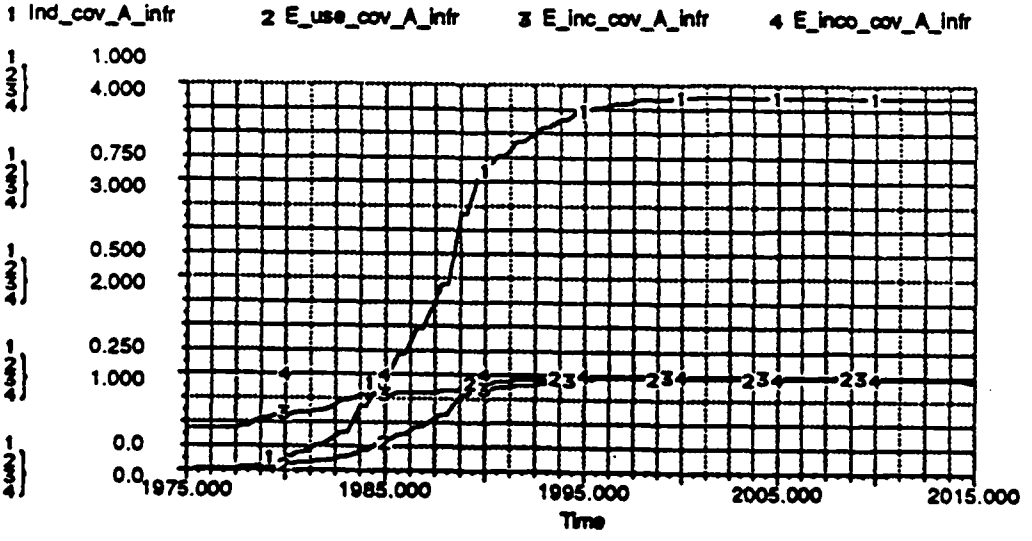
Plot 9



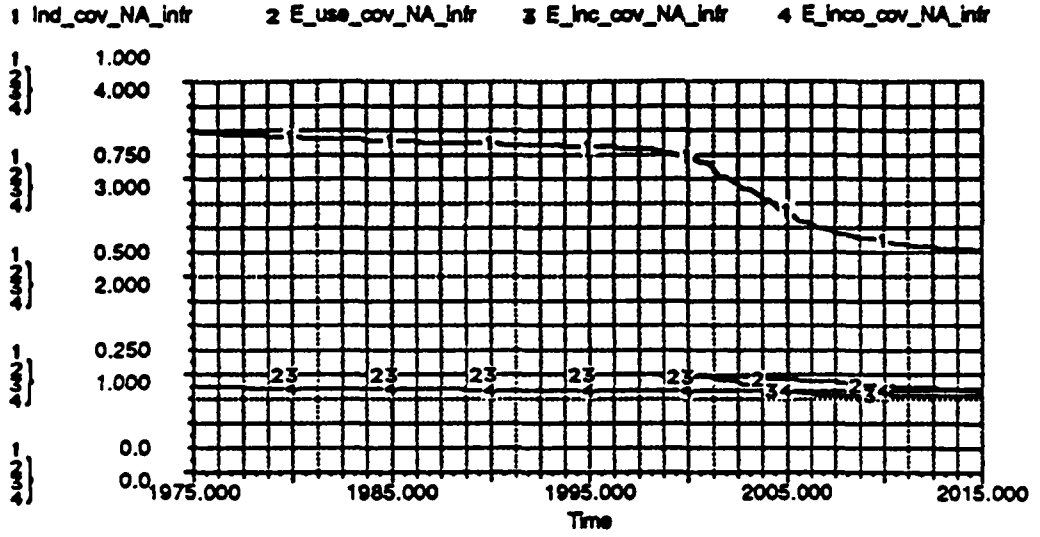
Plot 10



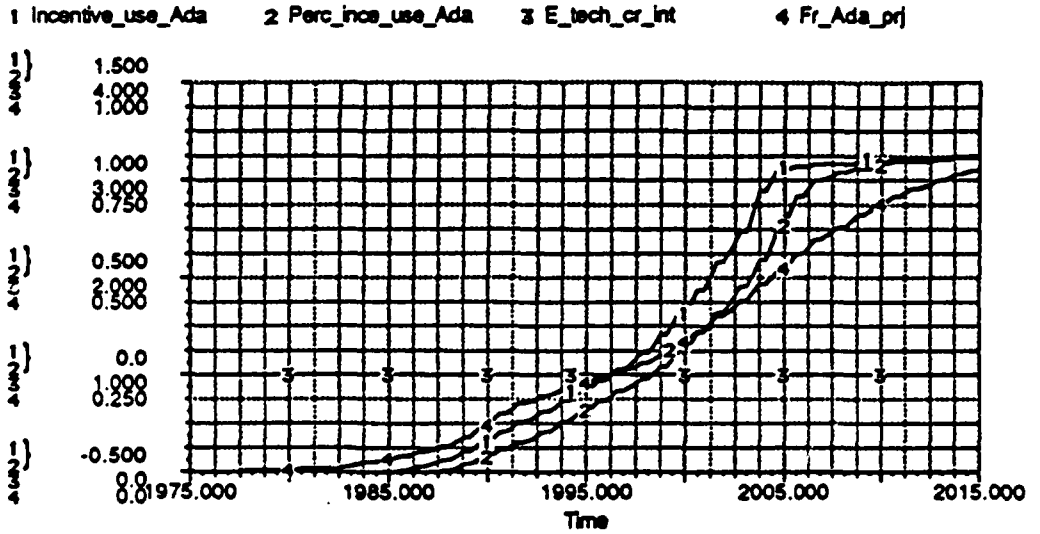
Plot 11



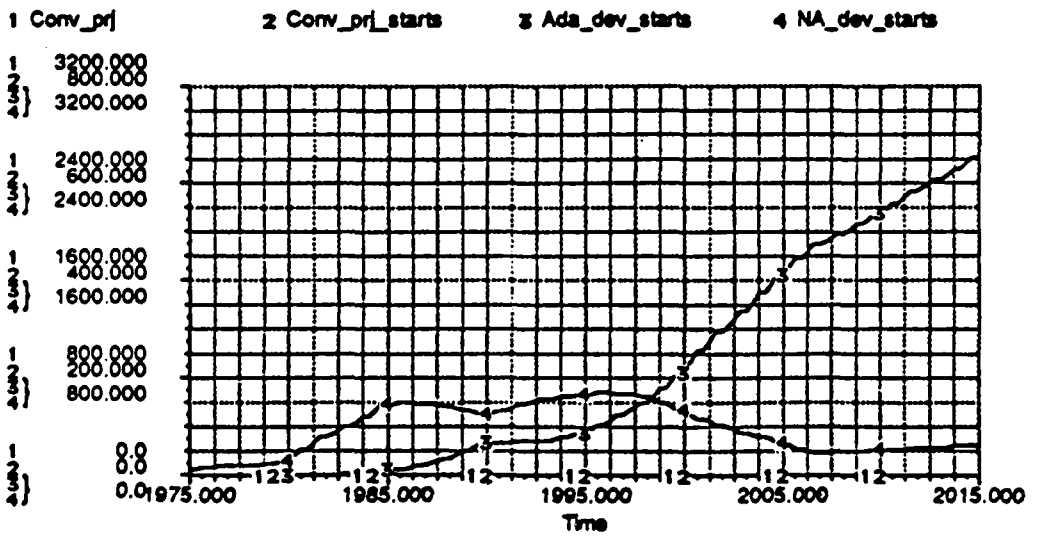
Plot 12



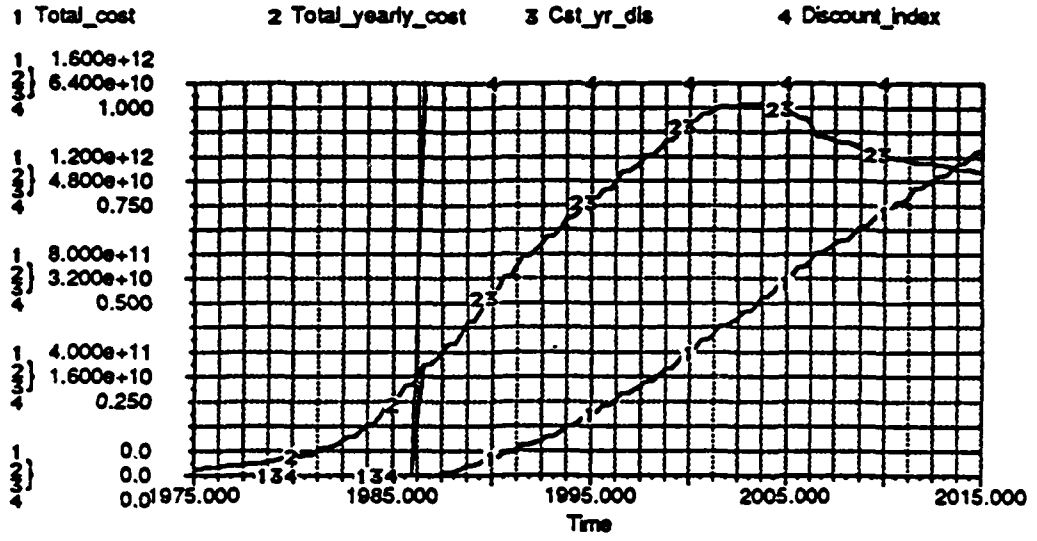
Plot 13



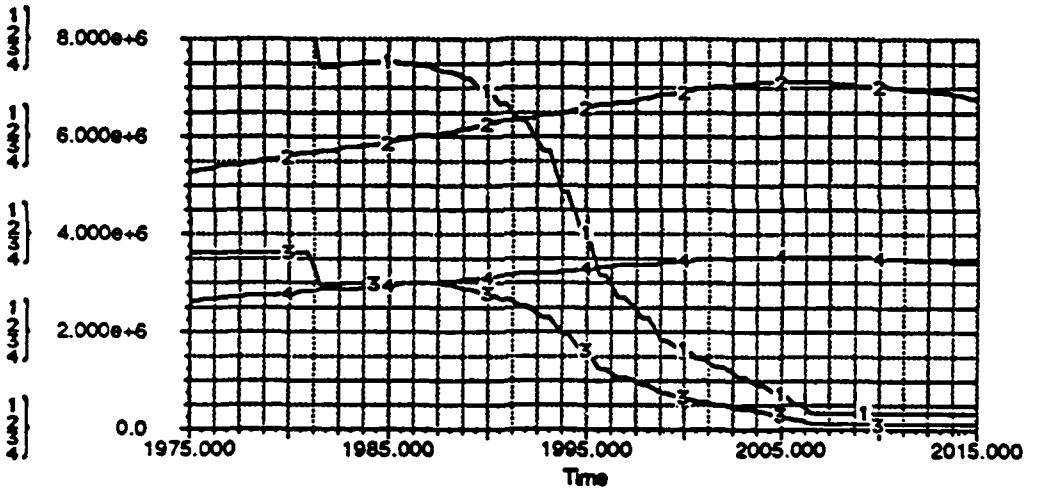
Plot 14



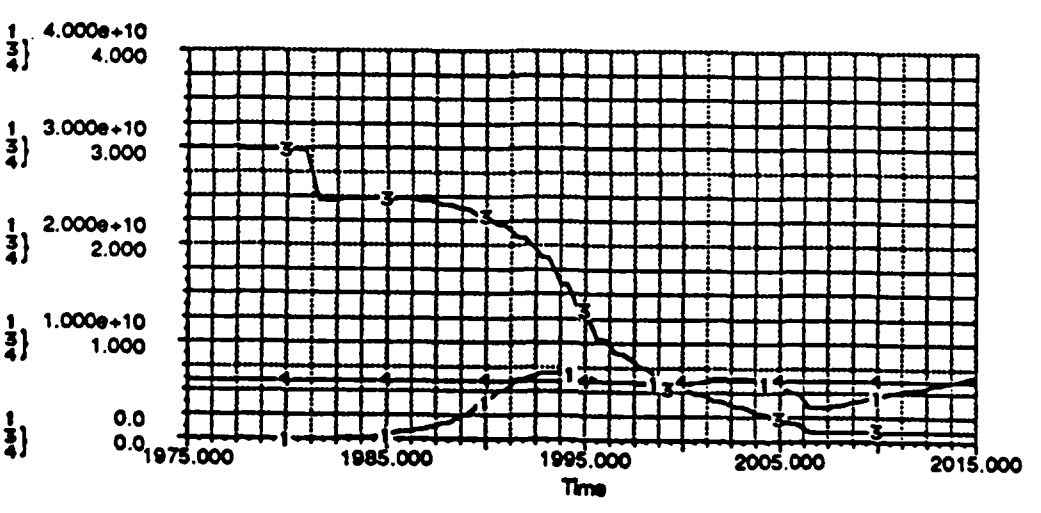
Plot 15



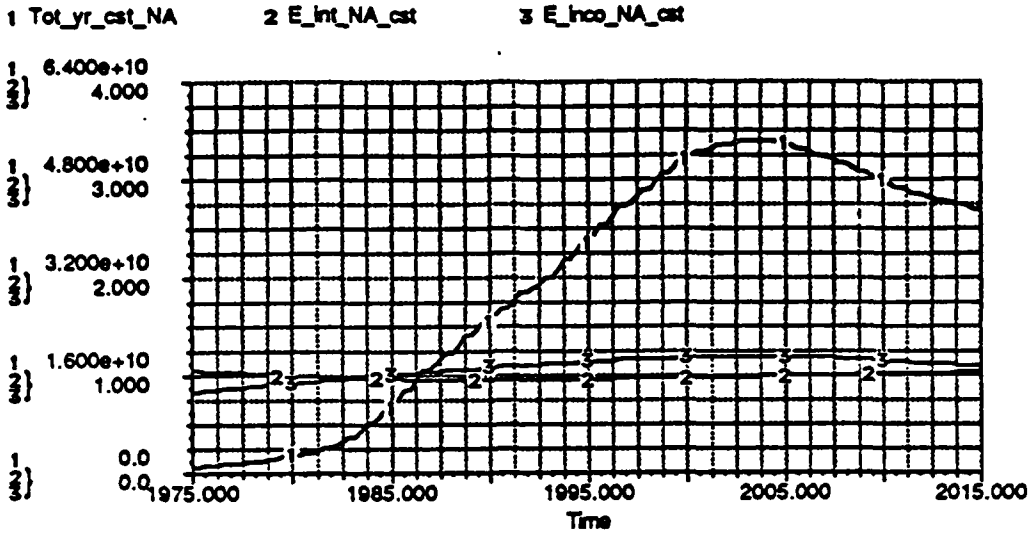
Plot 16



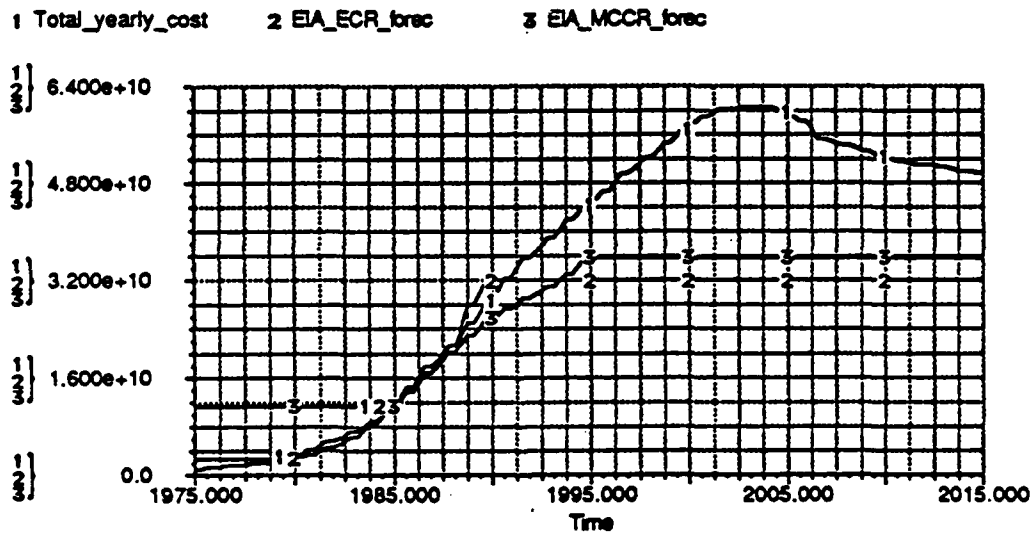
Plot 17



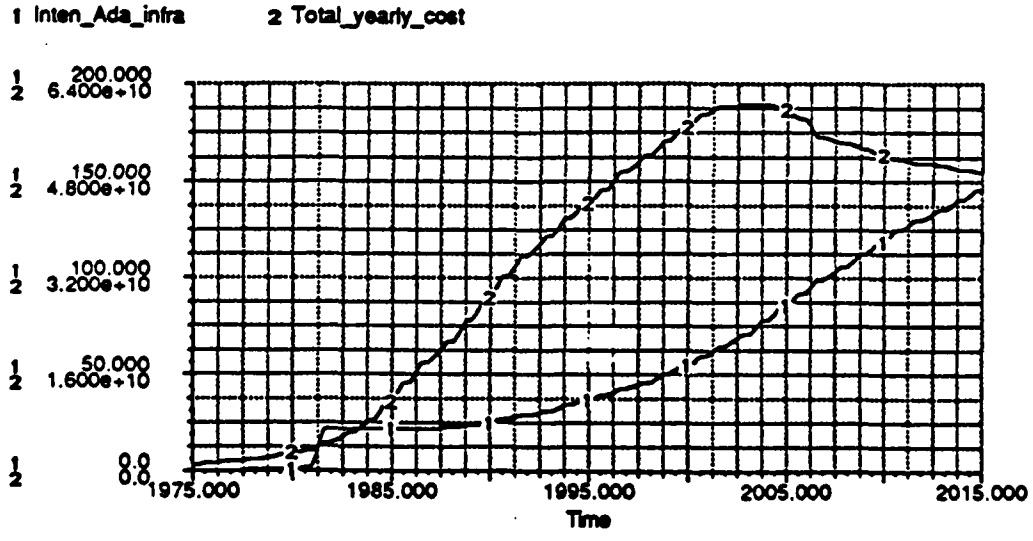
Plot 18



Plot 19



Plot 20

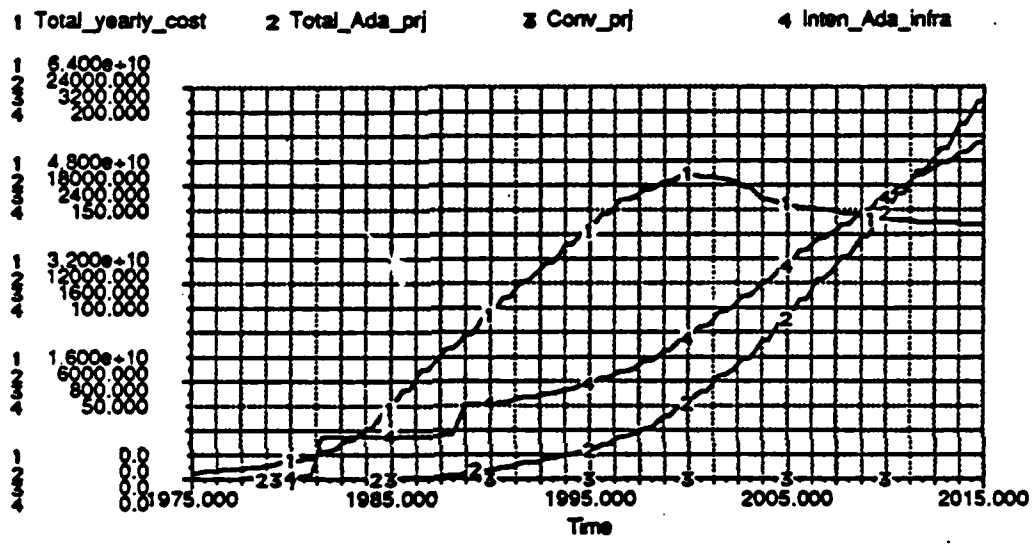


Time	Total_yearly_cost	Fr_Ada_prj	Total_cost	Rel_int_Ada_infr	Total_prj_starts
1975.000	1.029e+9	0.0	0.0	0.0	50.000
1980.000	2.867e+9	5.902e-3	0.0	0.034	120.000
1985.000	1.123e+10	0.033	0.0	0.506	620.000
1990.000	2.863e+10	0.117	6.440e+10	0.573	760.000
1995.000	4.447e+10	0.229	2.418e+11	0.856	1020.000
2000.000	5.705e+10	0.333	4.894e+11	1.301	1390.000
2005.000	5.982e+10	0.522	7.868e+11	2.101	1915.000
2010.000	5.246e+10	0.687	1.068e+12	3.091	2360.000
2015.000	4.976e+10	0.778	1.324e+12	3.954	2870.000

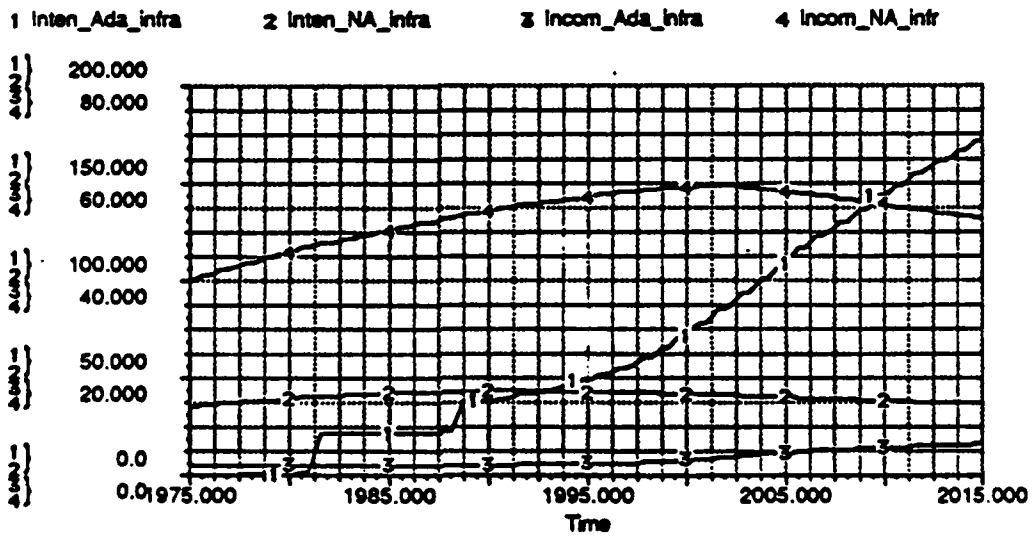
Appendix B.4: Complete Output for Commercial APSE Scenario

This appendix contains all of the output plots and tables for the scenario representing the adoption of a commercially available environment as the standard APSE, as an interim standard for all DOD Ada projects. This appendix then gives the equation changes to the base model that produce this scenario. These changes could be done on line to the base model. However, to ensure easy reproducibility of results, a model with the changes already made has been supplied along with this report on a Macintosh floppy disk as STELLA document CAPSEM0.32, (Commercial APSE Model Release 0 version 32, corresponding to the base model SSM0.32). That model stored on the floppy disk contains the output plot and tables as well as the equations and flow diagram.

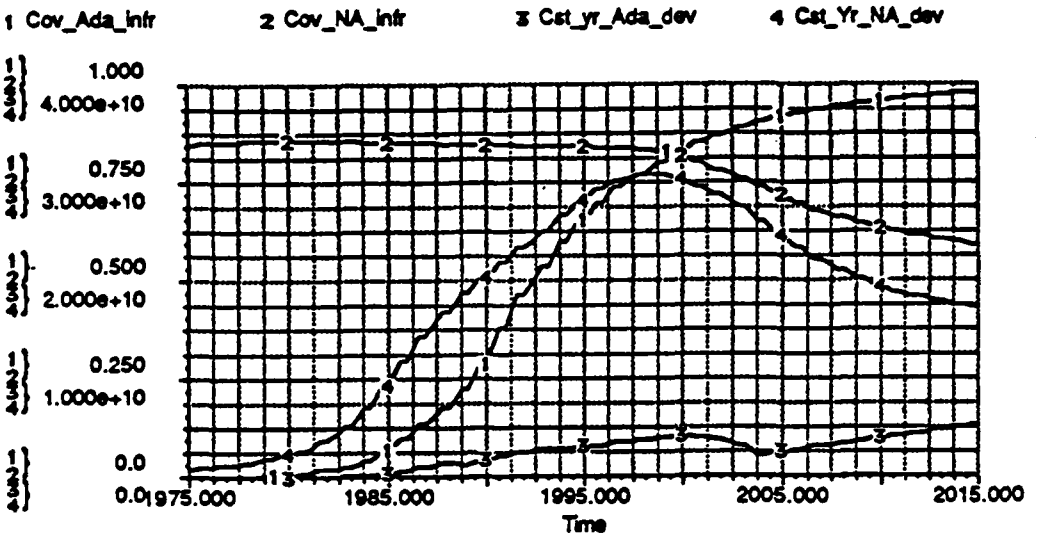
Plot 1



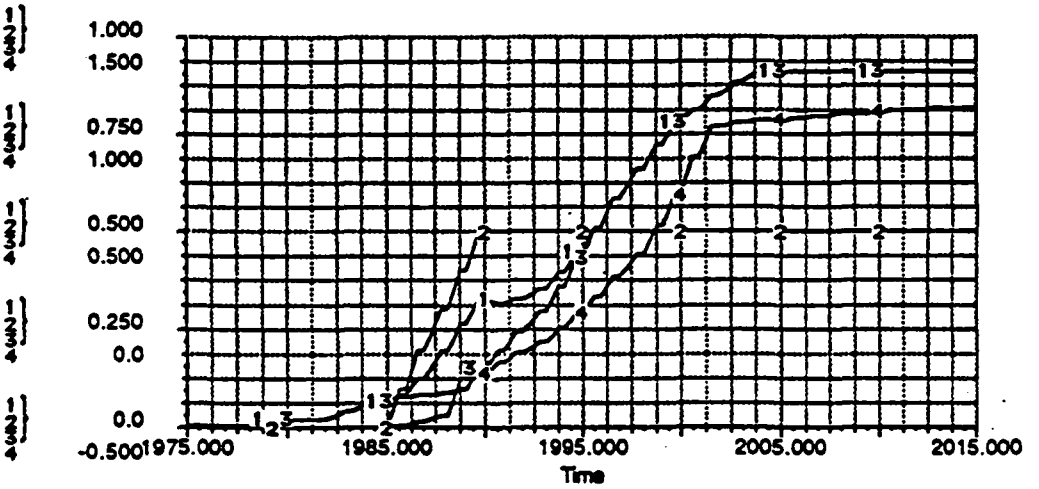
Plot 2



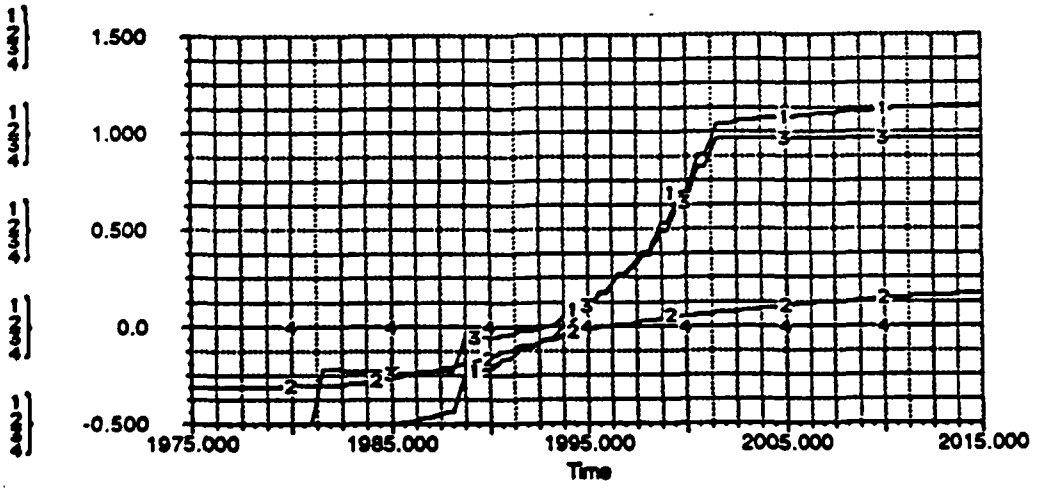
Plot 3



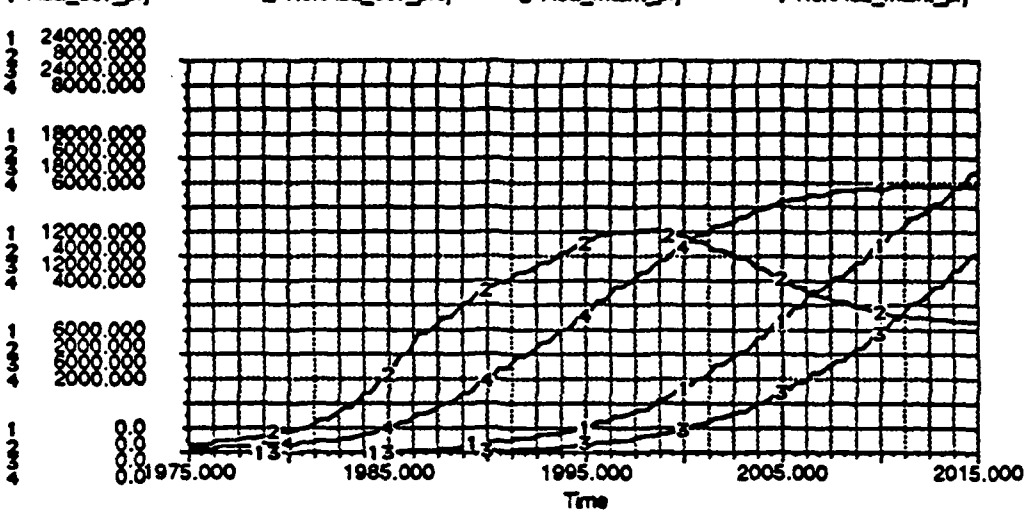
Plot 4



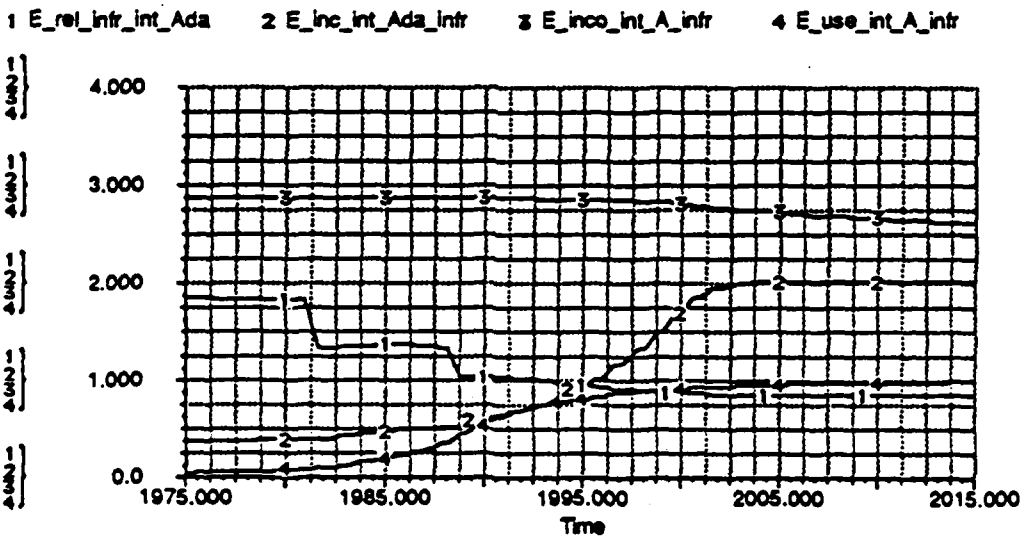
Plot 5



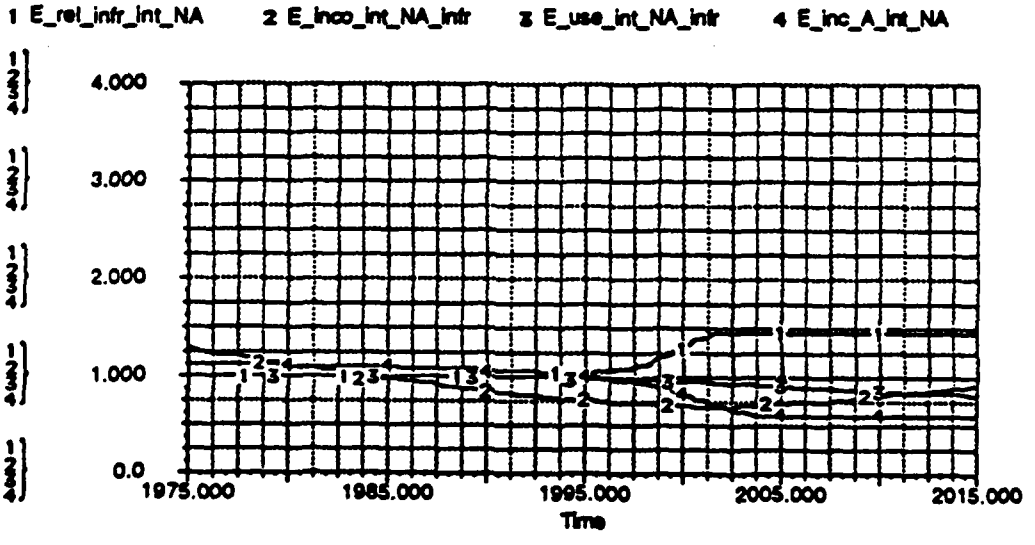
Plot 6



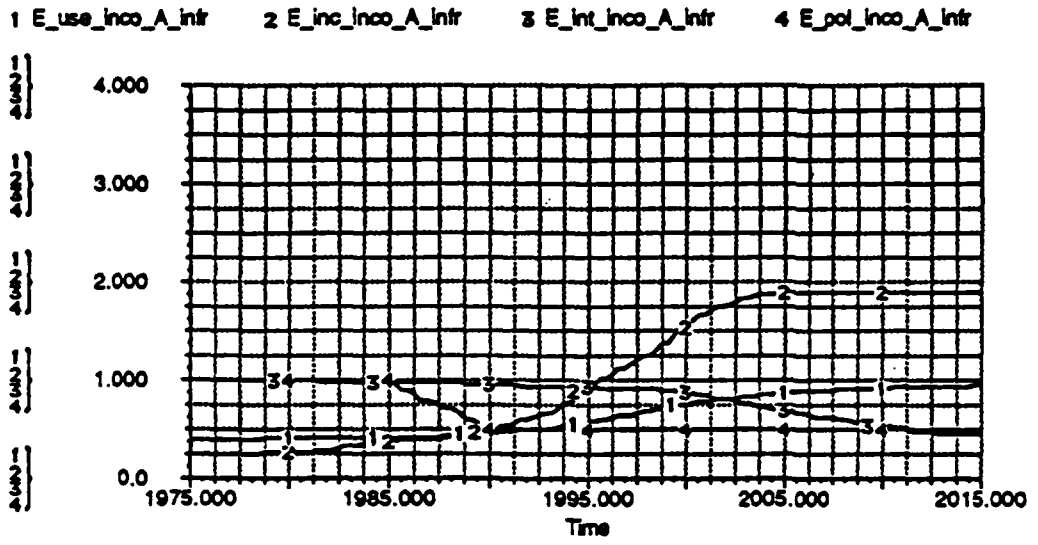
Plot 7



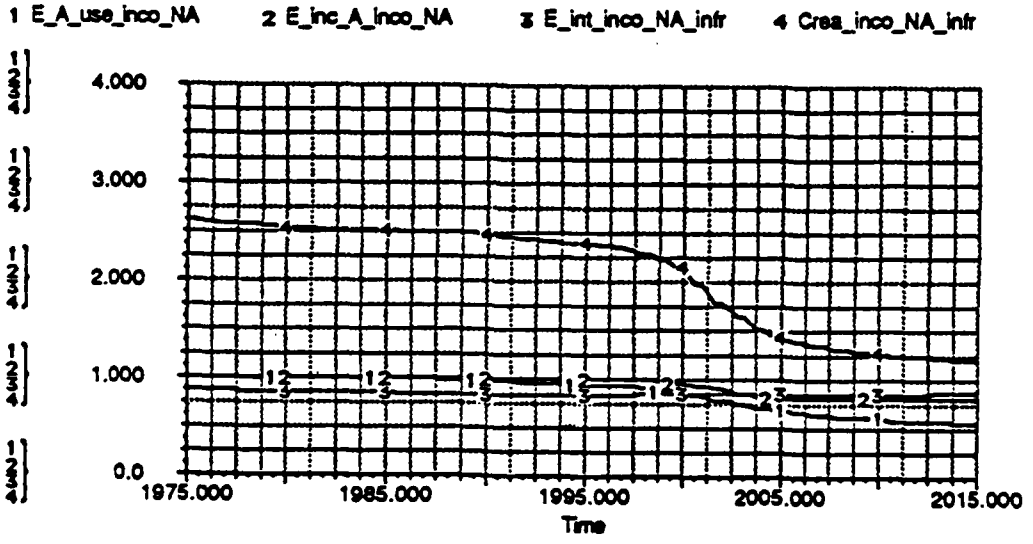
Plot 8



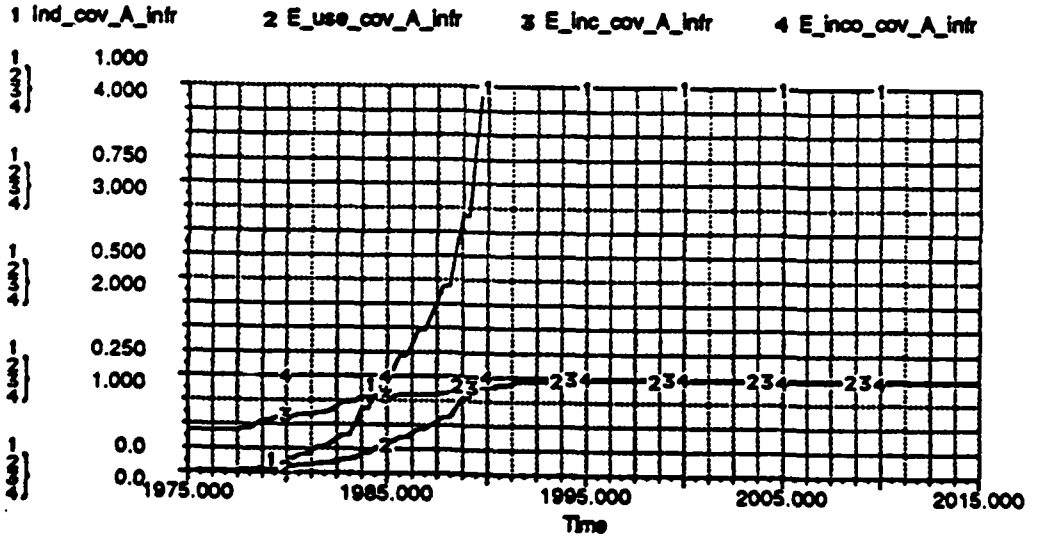
Plot 9



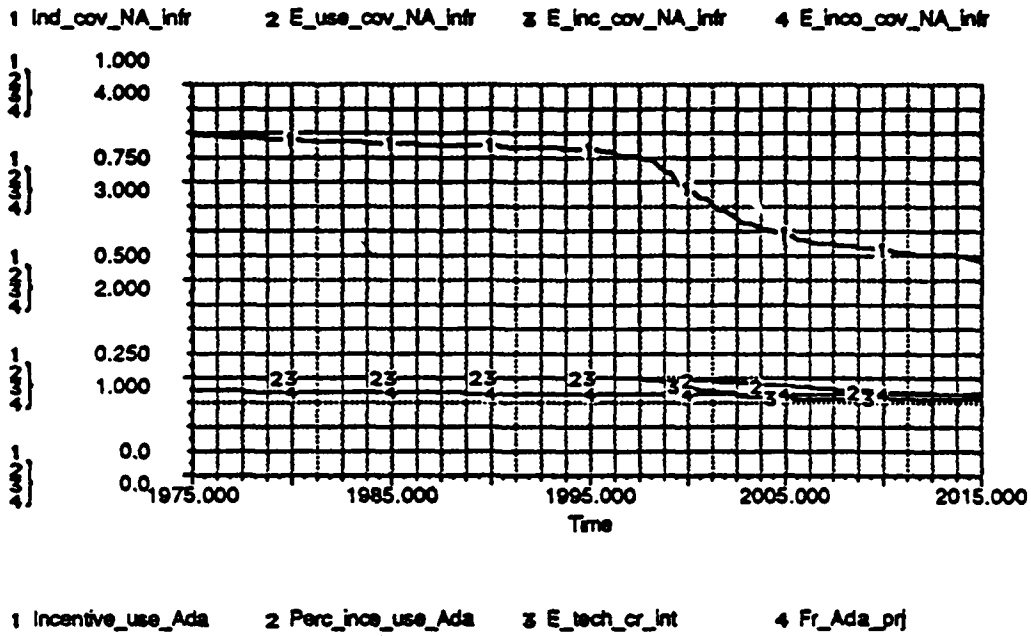
Plot 10



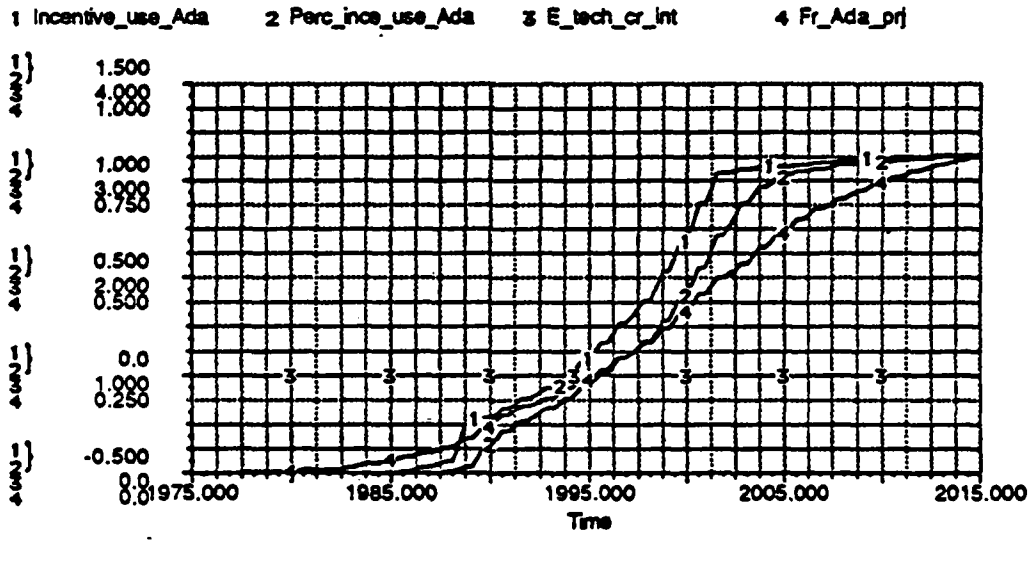
Plot 11



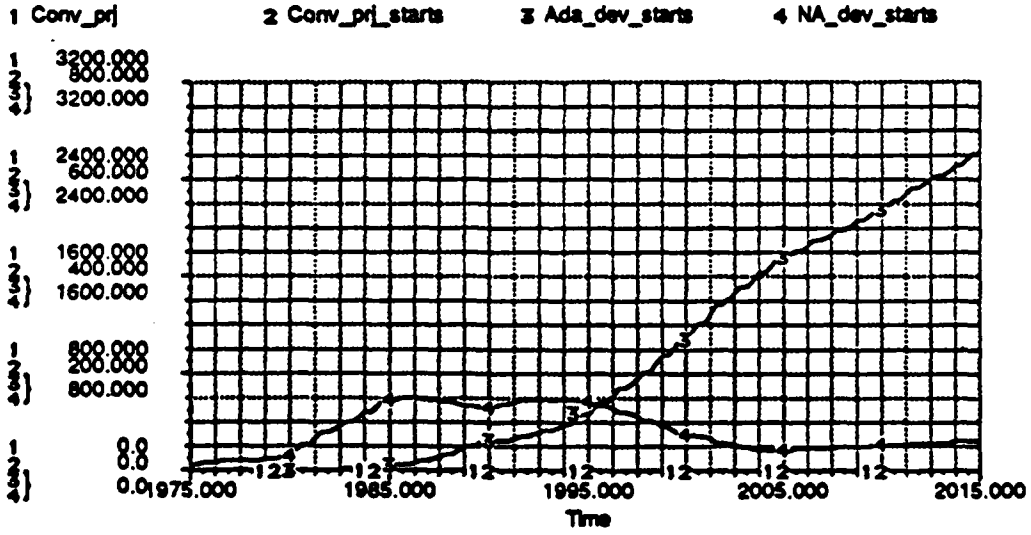
Plot 12



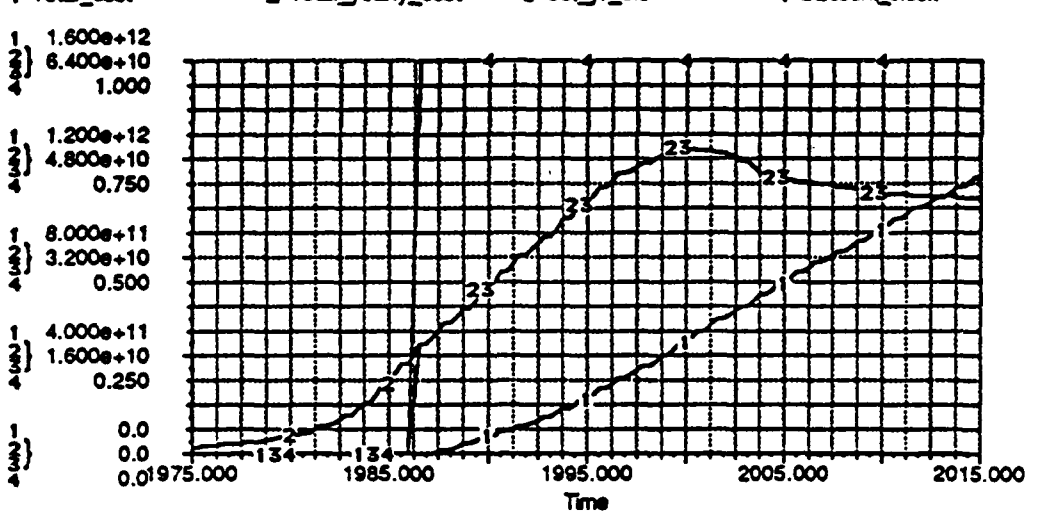
Plot 13



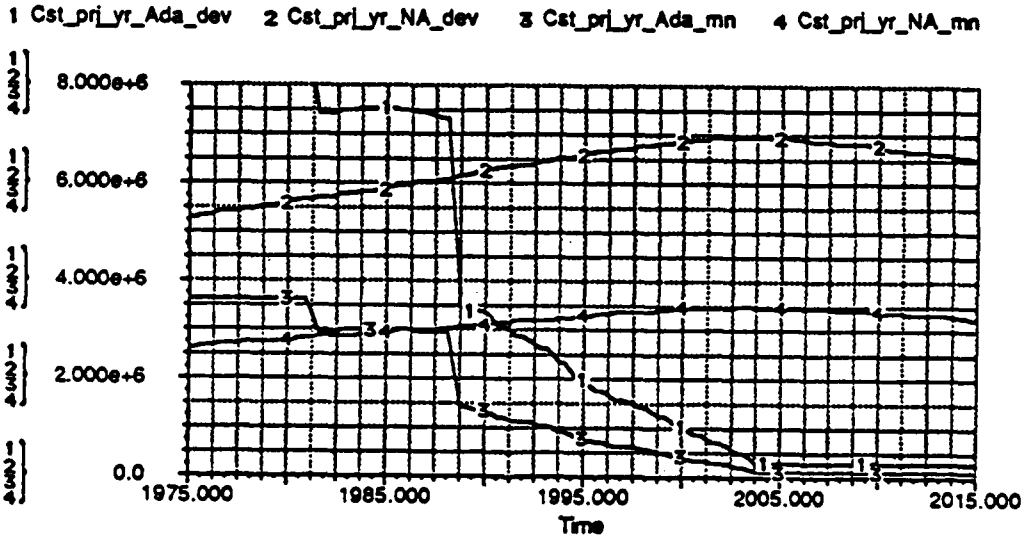
Plot 14



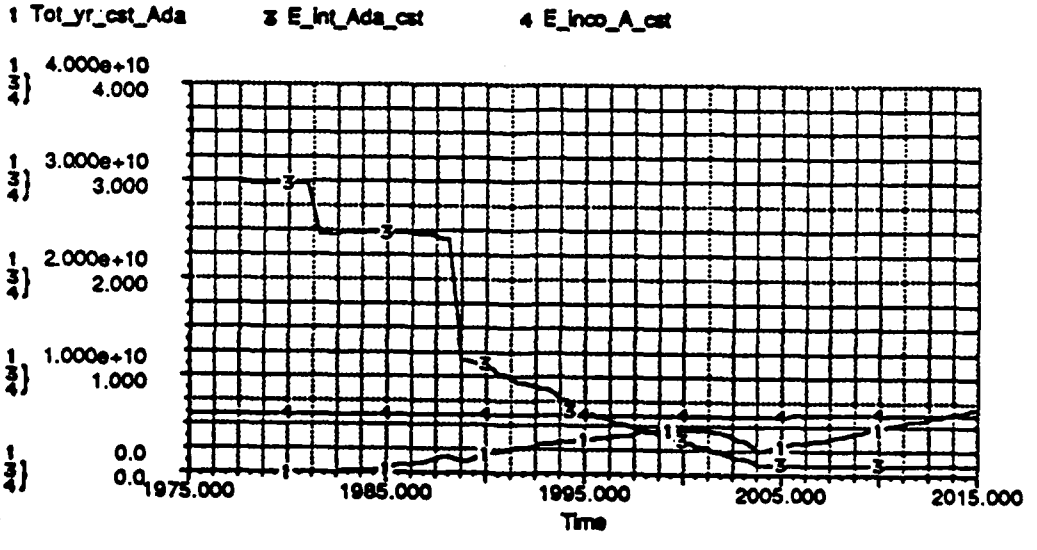
Plot 15



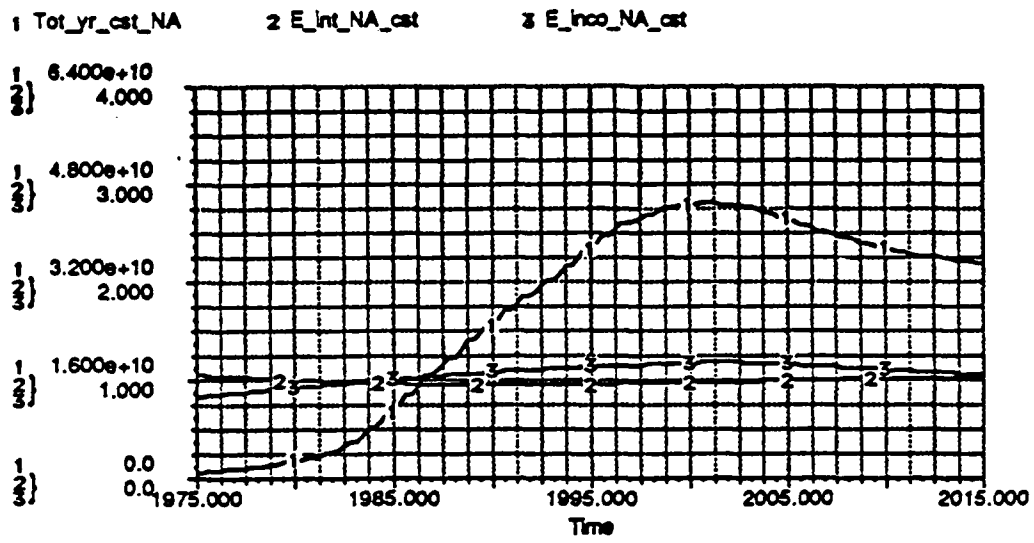
Plot 16



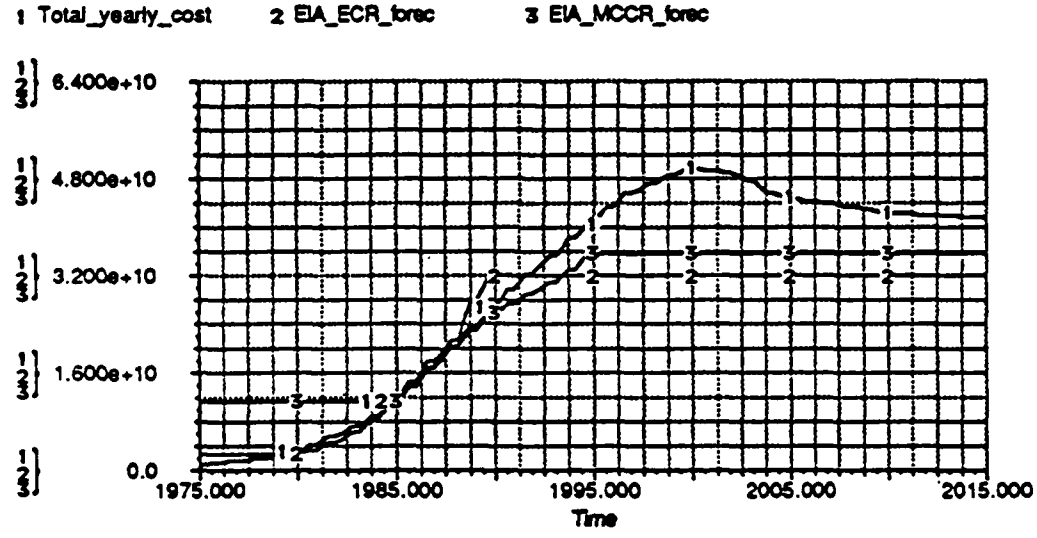
Plot 17



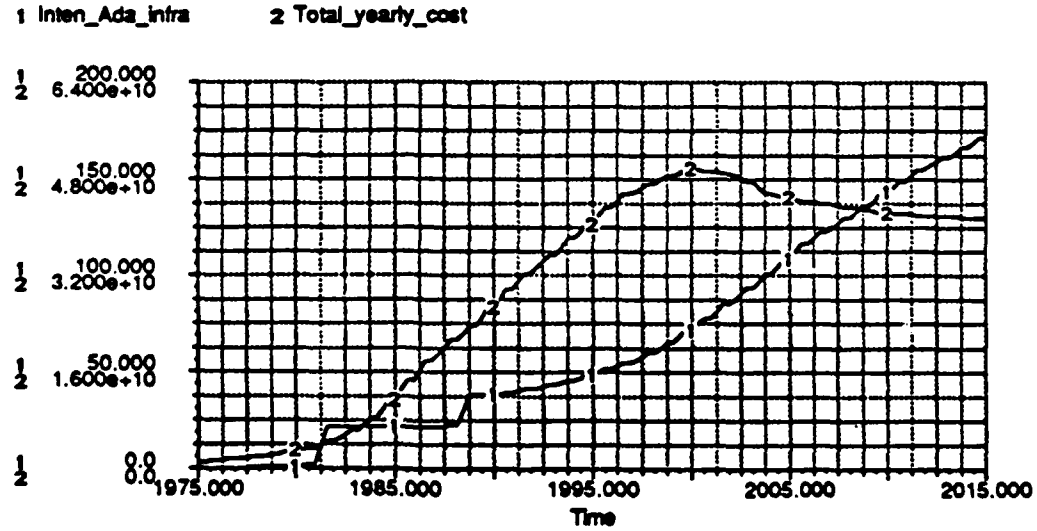
Plot 18



Plot 19



Plot 20



Time	Total_yearly_cost	Fr_Ada_prj	Total_cost	Rel_int_Ada_infr	Total_prj_starts
1975.000	1.029e+9	0.0	0.0	0.0	50.000
1980.000	2.867e+9	5.902e-3	0.0	0.034	120.000
1985.000	1.123e+10	0.033	0.0	0.506	620.000
1990.000	2.675e+10	0.118	6.311e+10	0.898	760.000
1995.000	4.075e+10	0.239	2.253e+11	1.133	1020.000
2000.000	4.950e+10	0.413	4.511e+11	1.707	1390.000
2005.000	4.513e+10	0.616	6.928e+11	2.703	1915.000
2010.000	4.248e+10	0.744	9.124e+11	3.728	2360.000
2015.000	4.179e+10	0.812	1.123e+12	4.625	2870.000

○ Add_GFE_Ada_infr = 15 {Additional GFE'd Ada infrastructure (infrastructure units)}

○ T_add_Ada_infr = 1988
{Time for additional Ada infrastructure (year)}

⊘ E_pol_inco_A_infr = graph(year)
1970.000 -> 1.000
1975.000 -> 1.000
1980.000 -> 1.000
1985.000 -> 1.000
1990.000 -> 0.500
1995.000 -> 0.500
2000.000 -> 0.500
2005.000 -> 0.500
2010.000 -> 0.500
2015.000 -> 0.500
2020.000 -> 0.500

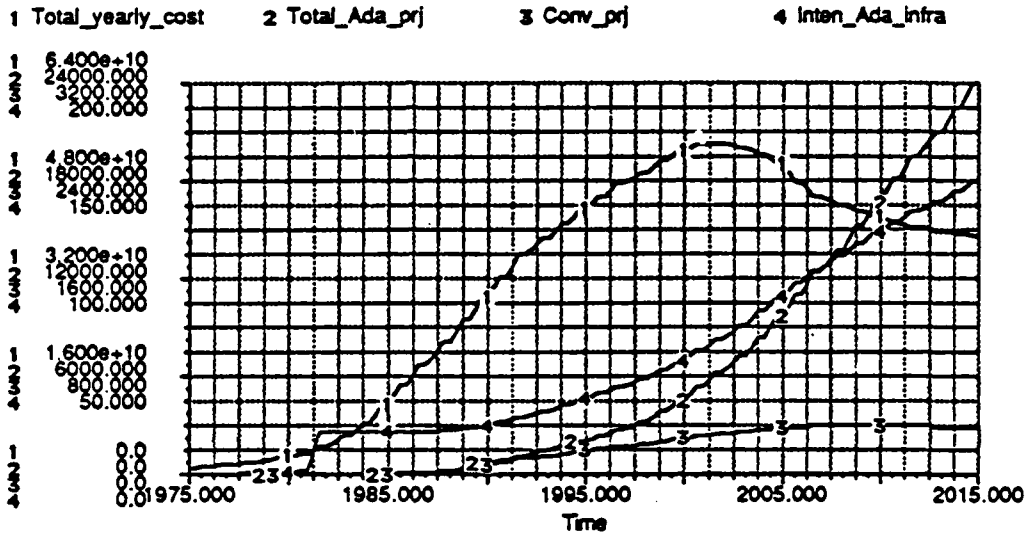
⊘ E_pol_t_ch_cov_A = graph(year)
1970.000 -> 1.000
1975.000 -> 1.000
1980.000 -> 1.000
1985.000 -> 1.000
1990.000 -> 1.500
1995.000 -> 1.500
2000.000 -> 1.500
2005.000 -> 1.500
2010.000 -> 1.500
2015.000 -> 1.500
2020.000 -> 1.500

⊘ Sw_transp_A_cov = graph(year)
1970.000 -> 0.0
1975.000 -> 0.0
1980.000 -> 0.0
1985.000 -> 0.0
1990.000 -> 1.000
1995.000 -> 1.000
2000.000 -> 1.000
2005.000 -> 1.000
2010.000 -> 1.000
2015.000 -> 1.000
2020.000 -> 1.000

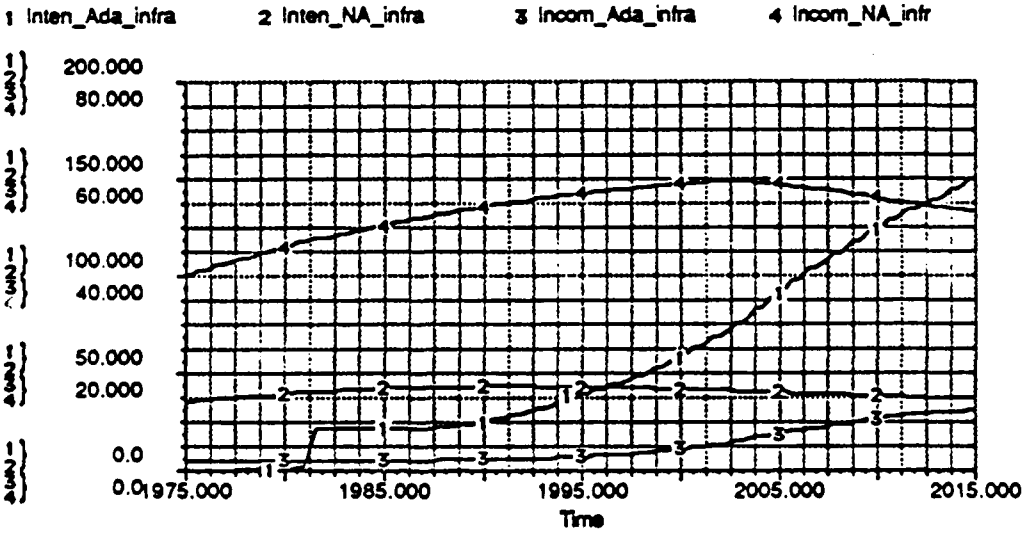
Appendix B.5: Complete Output for Conversion Scenario

This appendix contains all output plots and tables for the scenario representing gradual conversion of non-Ada programming being converted to Ada during maintenance. The changes to the base model done to represent this policy test (given here after the output) have been incorporated into an altered version of the base model which resides on a Macintosh floppy disk as a STELLA document named CONVM0.32 (Conversion Model, release 0, version 32, derived from the corresponding base model SSM0.32). As usual, the stored model contains all of the output information given here, in addition to the equations and flow diagram.

Plot 1

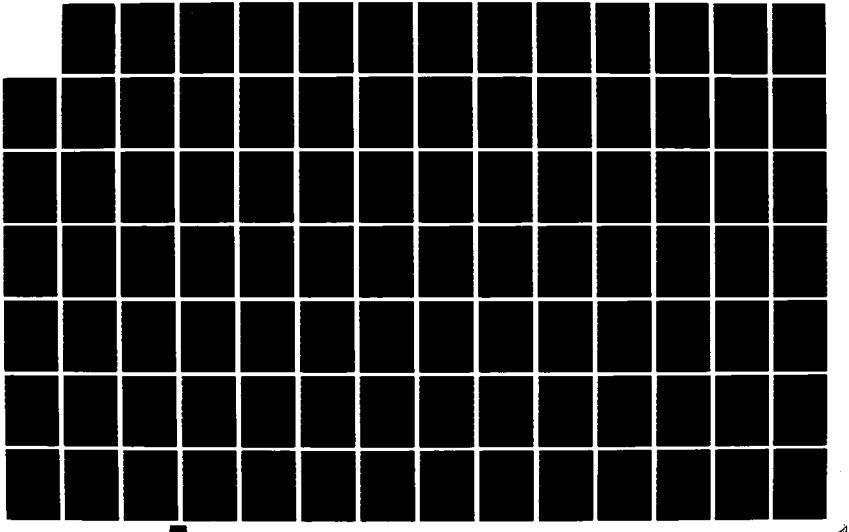


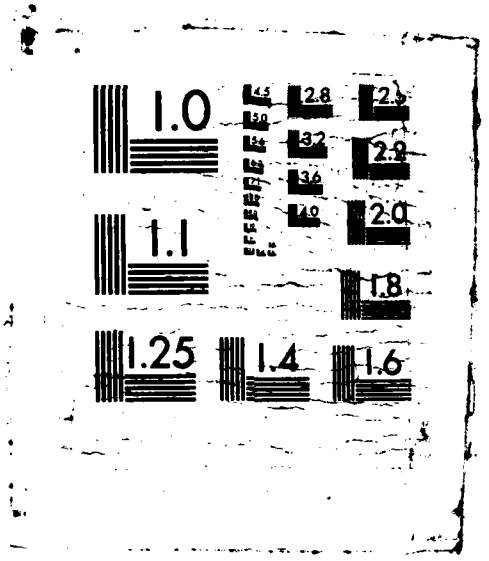
Plot 2



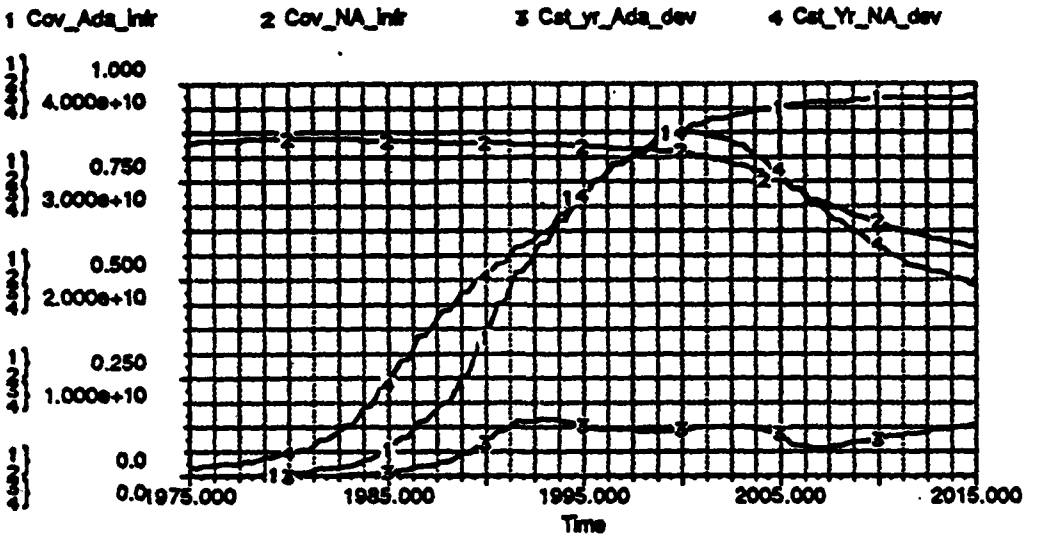
AD-A175 352 COST EFFECTIVENESS TRADEOFFS IN COMPUTER STANDARDIZATION AND TECHNOLOGY I..(U) INSTITUTE FOR DEFENSE ANALYSES ALEXANDRIA VA A A HOOK ET AL. JUN 86 IDA-P-1931 IDA/HQ-86-31052 MDA903-84-C-0031 F/G 9/2 3/4

UNCLASSIFIED NL

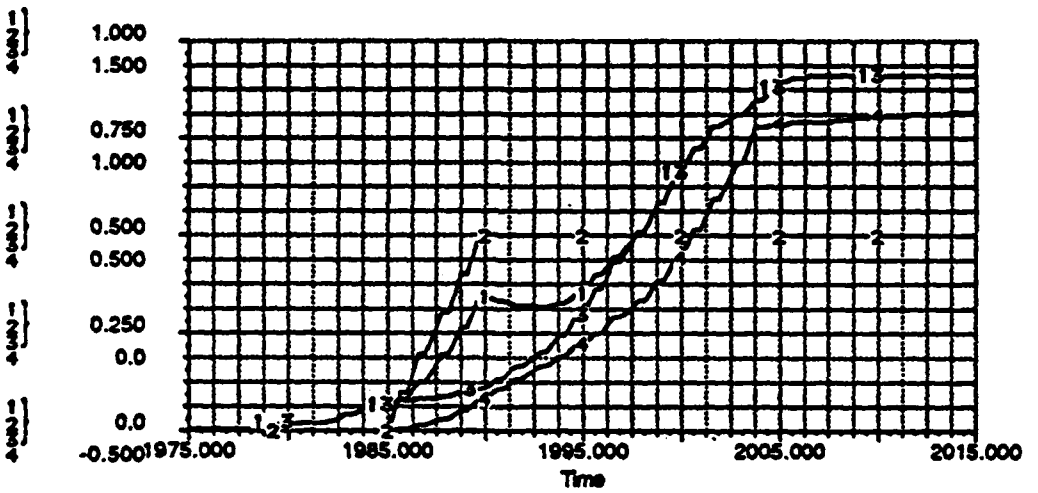



1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0

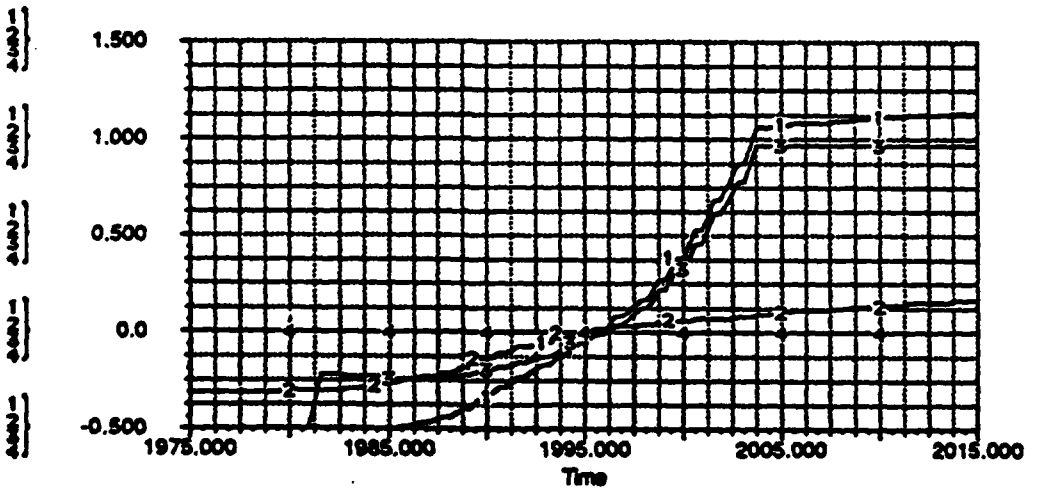
Plot 3



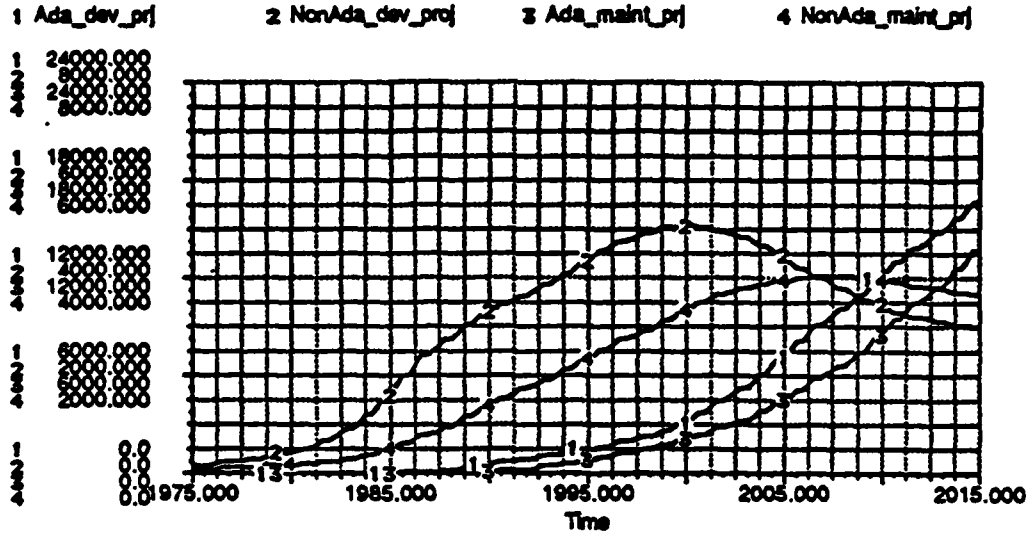
Plot 4



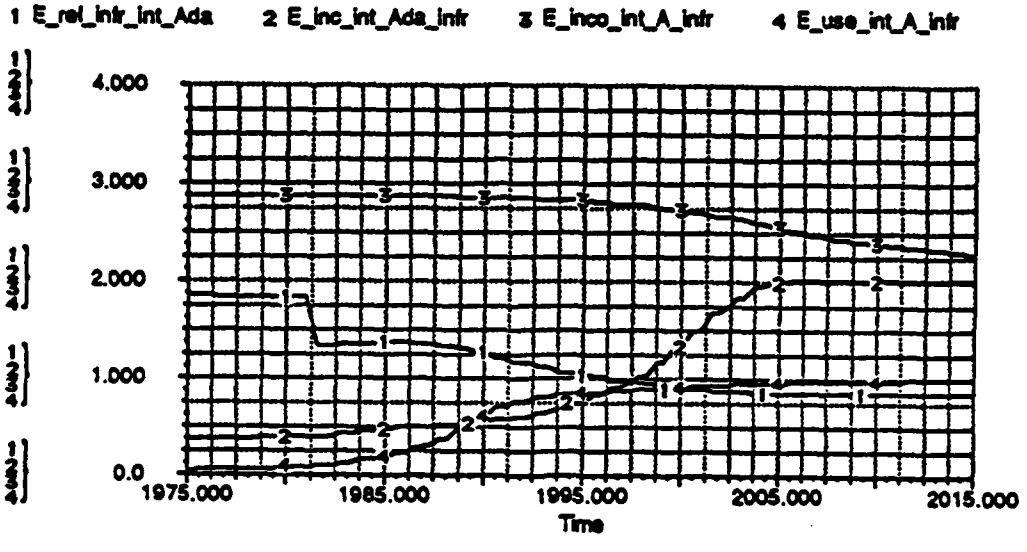
Plot 5



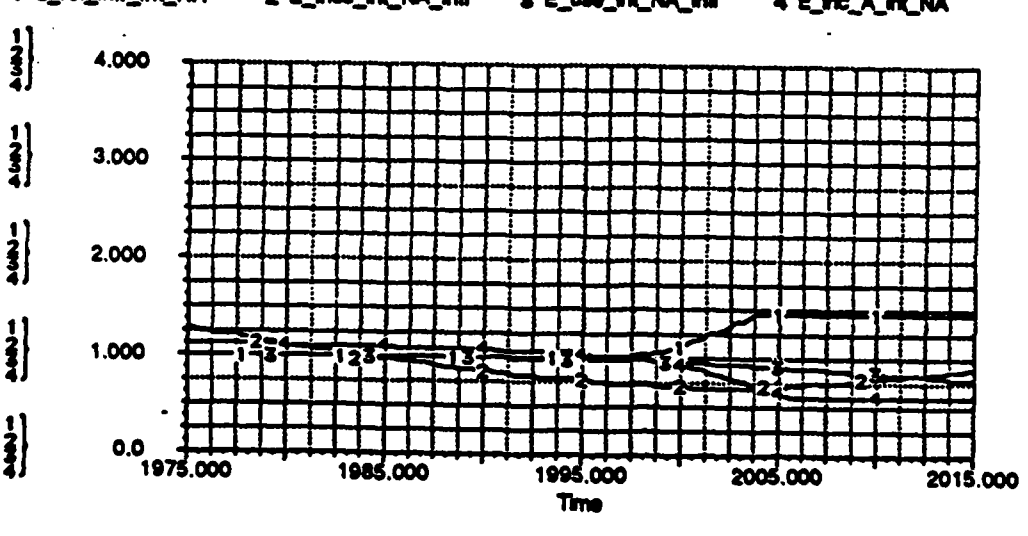
Plot 6



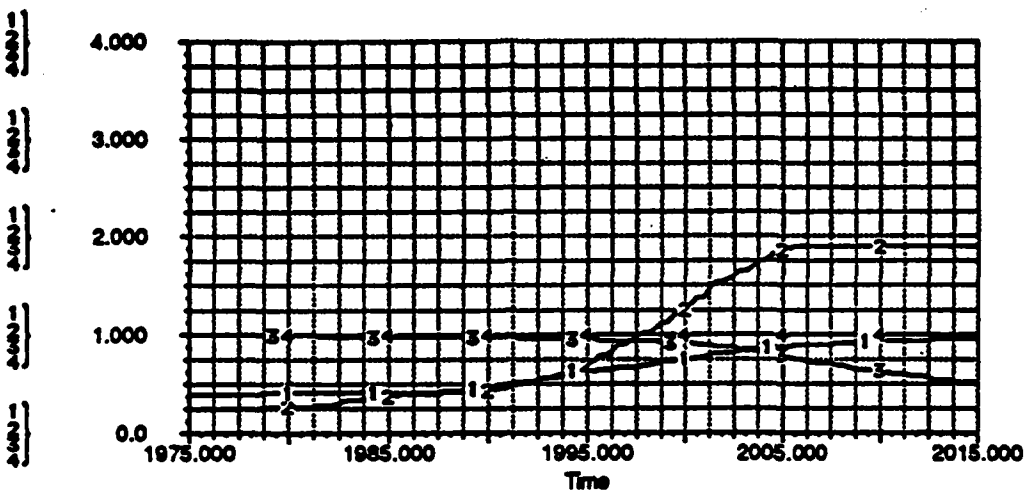
Plot 7



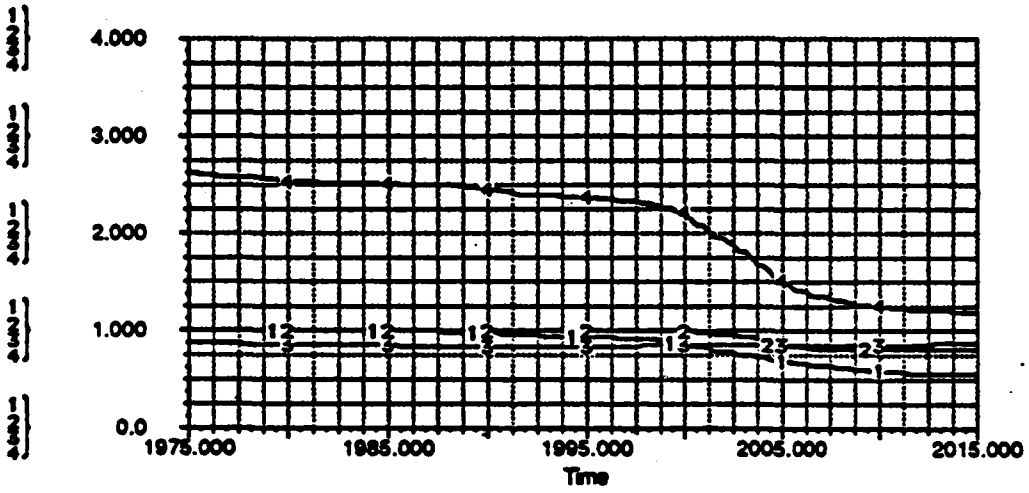
Plot 8



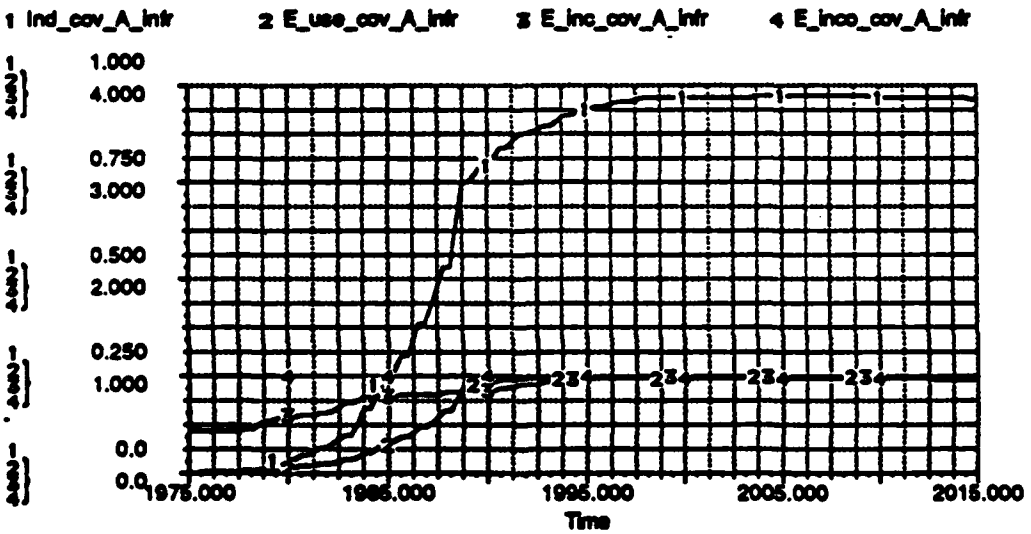
Plot 9



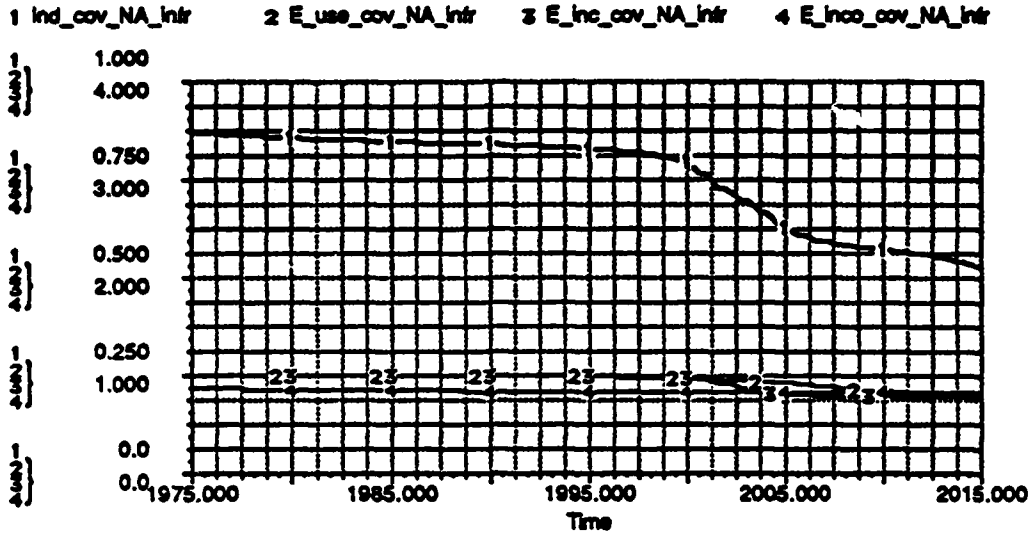
Plot 10



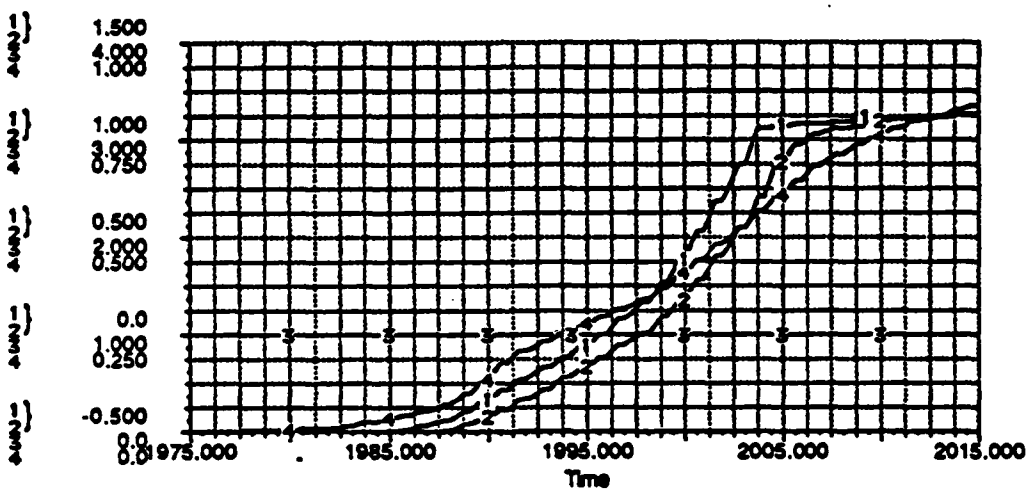
Plot 11



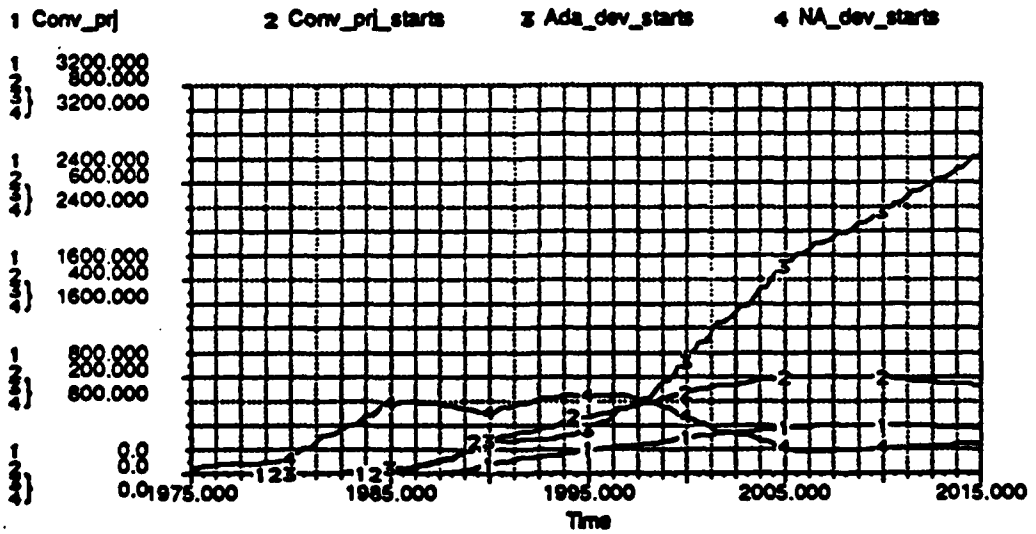
Plot 12



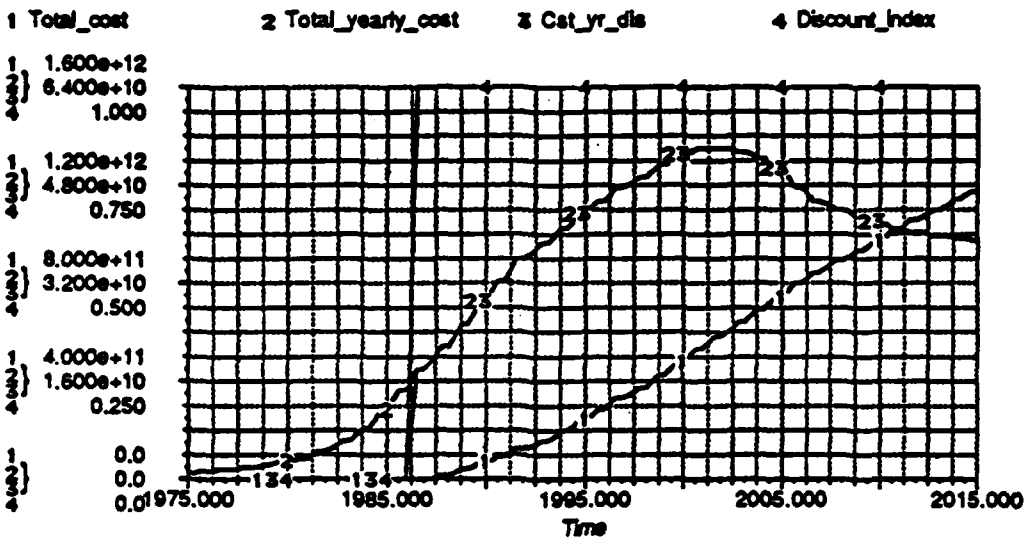
Plot 13



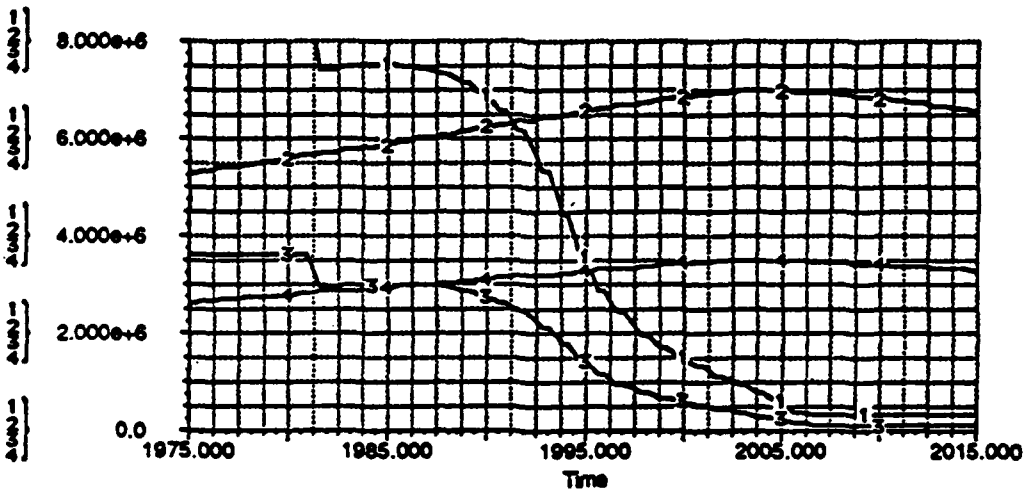
Plot 14



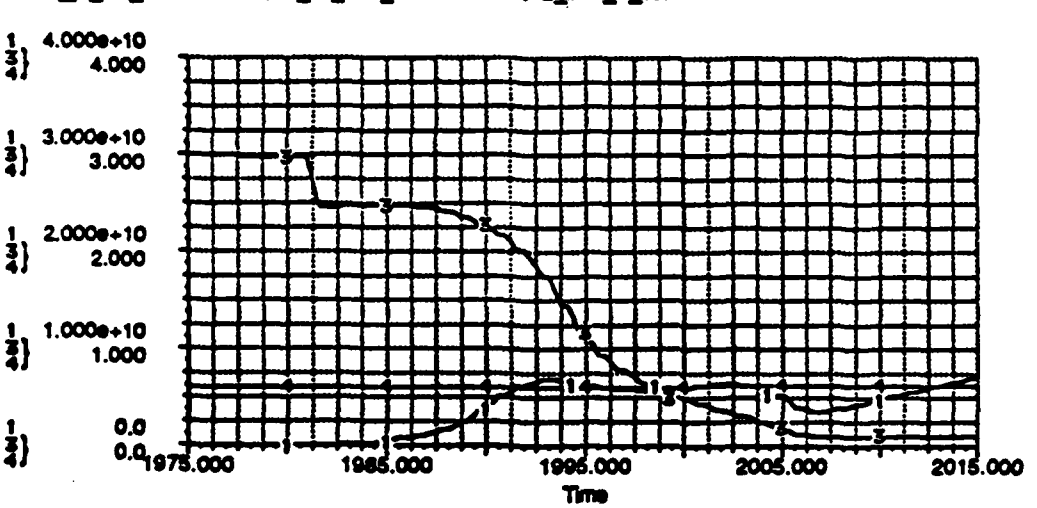
Plot 15



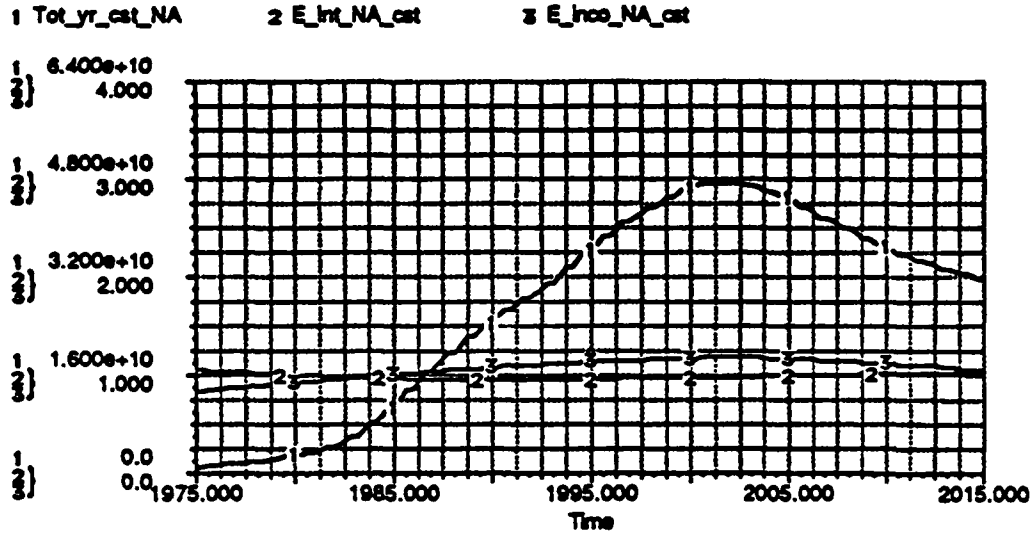
Plot 16



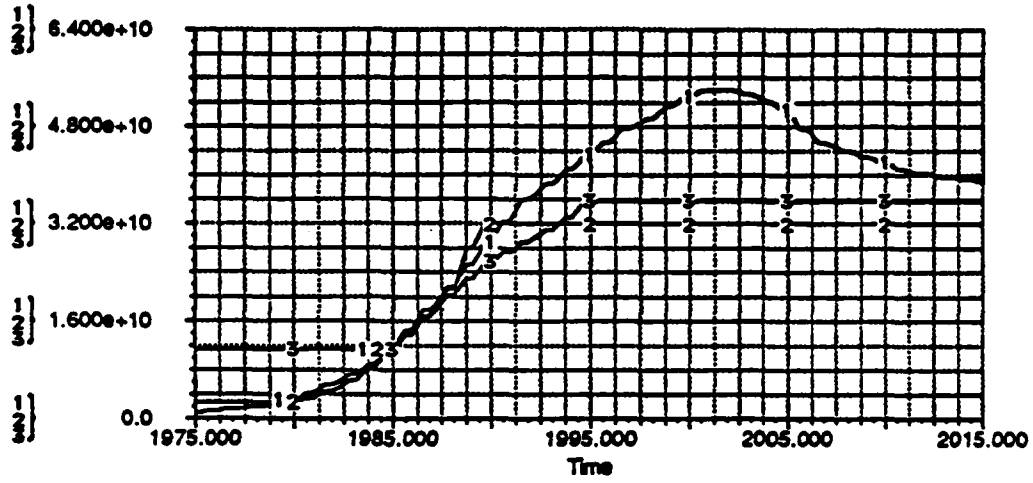
Plot 17



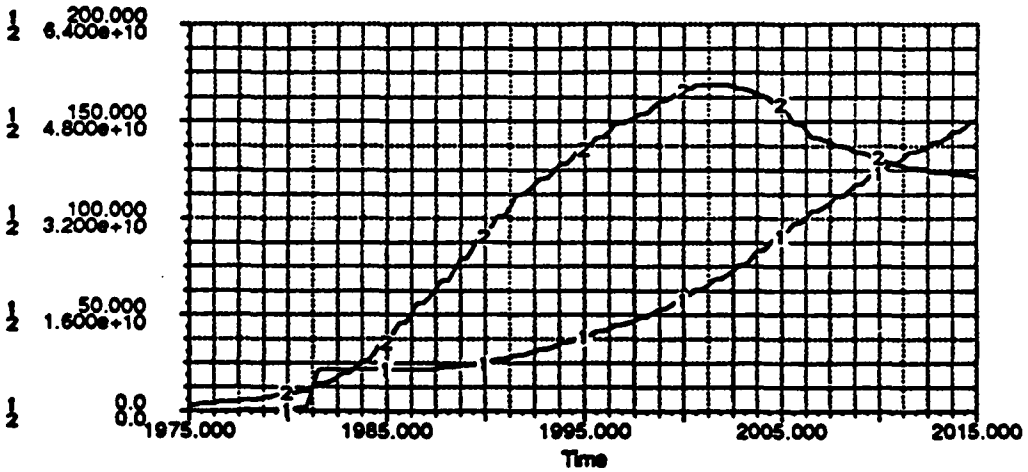
Plot 18



Plot 19



Plot 20



Time	Total yearly cost	Fr Ada prj	Total cost	Rel int Ada infr	Total prj starts
1975.000	1.029e+9	0.0	0.0	0.0	50.000
1980.000	2.867e+9	5.902e-3	0.0	0.034	120.000
1985.000	1.123e+10	0.033	0.0	0.506	620.000
1990.000	2.884e+10	0.135	6.466e+10	0.582	760.000
1995.000	4.321e+10	0.275	2.417e+11	0.892	1020.000
2000.000	5.303e+10	0.406	4.787e+11	1.382	1390.000
2005.000	5.066e+10	0.607	7.455e+11	2.233	1915.000
2010.000	4.202e+10	0.761	9.763e+11	3.204	2360.000
2015.000	3.885e+10	0.838	1.179e+12	4.033	2870.000

Fr_conv_NA_mn_pr=graph(year)
1970.000-> 0.0
1975.000-> 0.0
1980.000-> 0.0
1985.000-> 0.0
1990.000-> 0.050
1995.000-> 0.050
2000.000-> 0.050
2005.000-> 0.050
2010.000-> 0.050
2015.000-> 0.050
2020.000-> 0.050

Appendix B.6: Matrix of Policies and Levers

There are several ways to represent the effects of policies in models. In very detailed models with a relatively small number of policies to evaluate, one has available a single parameter that represents whether or not a given policy is to be in effect. In the study reported here, the model is relatively aggregated. However, there are numerous policies that need to be evaluated. Thus, a somewhat different means of representing policies is used. The model contains levers at important points in the system, such that the effects of any set of policies should be representable by changes in one or more of the levers. For example, one of the effects of introducing a Common Ada Interface Standard (CAIS) would be to reduce the amount of incompatibility in the infrastructure being created, all other things being equal—there would be much less incompatibility among operating systems and tool sets as a result of imposing a standard. This effect of a CAIS is represented by reducing the Effect of policy on incompatibility of Ada infrastructure (E_pol_inco_A_infr), which, other inputs being equal, reduces the Creation of incompatibility of Ada infrastructure (Cr_inco_Ada_infr).

The study has currently identified eight policies concerning standardization of programming support environments that should eventually be examined in scenario simulations. A scenario may have several or none of these policies in force, for they are not mutually exclusive. Some have been proposed in pairs, such as using SVID (UNIX System V) as an interim standard operating system-tool interface, followed by use of a DoD-developed interface (the CAIS) as the successor. The seven policies are described below.

There are 8 policies, and around 10 policy levers to represent their effects. One convenient way of summarizing both is shown in Figure B.6-1, which shows a matrix of 7 of the policies versus the policy levers. At this point in the study, policy analysis experiments on the proof-of-principle model have just begun, with the experiments described in Section 5 of this report. In the normal course of simulating and analyzing policy experiments, changes will be made in how policies are represented, so the entries in the matrix are far from final. They are included to enhance the value of the review process: they constitute a statement of the intended use of the model. The policy levers are described in Appendix A, and the policies themselves are as follows:

Mandated use of Ada

This policy is already in force as per the DeLauer memorandum. The policy of mandated use of the Ada language for mission-critical systems is in force in all simulations, except where explicitly noted. Also in force, however, are the pressures of expense and haste that can lead to waivers on Ada use for specific projects.

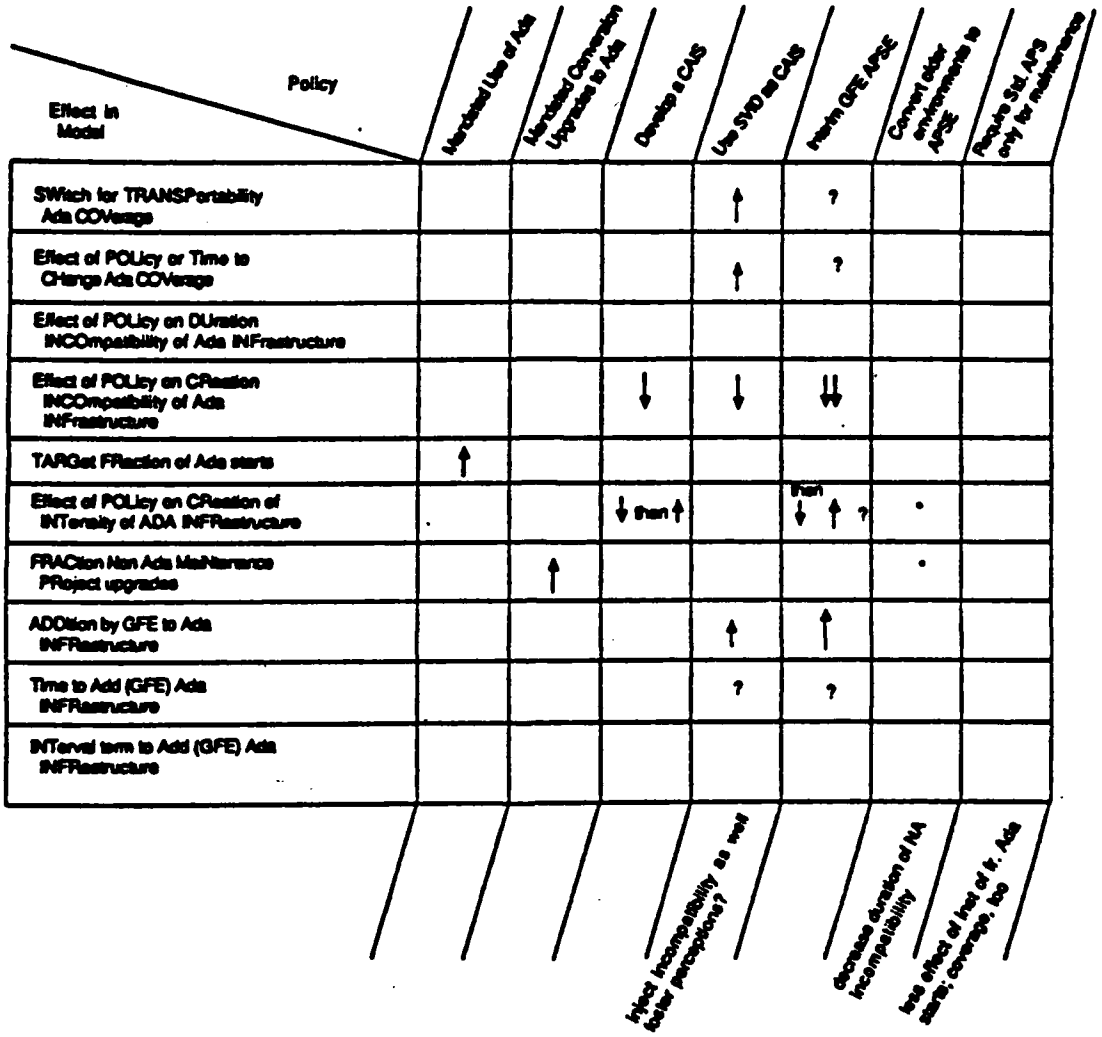


Figure B.6-1. Matrix of policies versus levers that represent such policies in the model.

Mandated conversions to Ada

As discussed in Appendix A.3, it is possible to do programming in the Ada language for deployed, operational systems that were originally programmed in other languages. Certainly the potential exists for major redevelopments, which tend to have mostly new code in any event. The potential also exists for doing piecemeal language conversion. One way is to write new subroutines in Ada when new subroutines are needed, then use a HOL-to-HOL translator to create HOL code in the same language as the rest of the system. The new subroutines can use existing linkers, loaders, and so on. This procedure is being used with CMS2. The other way to convert code piecemeal is to create Ada compilers that can link and load modules compatibly with modules sourced in the non-Ada language of the rest of the system. This is commonly done among FORTRAN, PL/I and other languages.

Although a policy of converting deployed systems to Ada is not prominent in current discussions, it is included here because it has potential to be a high-leverage policy. Simply because there are so many more systems in maintenance phase than there are being started in development, requiring use of Ada on a gradually-increasing number of existing systems offers the potential for rapidly increasing the rate at which Ada is used and thus internalized by the DoD programming community.

Develop a CAIS

The policy of developing and mandating a Common Ada Interface Standard, i.e. an operating system interface standard, is already gathering momentum. While no CAIS-based system yet exists, the effort to structure and specify such a system is well along.

Scenario simulations cannot speak to the technical efficacy of a given CAIS specification. What the model can evaluate (given assumptions about the technical merits of a standard, its development time and cost, and effect on productivity) is the impact on the evolution of DoD programming costs.

Use SVID as a CAIS

Until recently, there were no publicly available specifications for an operating system; there were only the actually-implemented OSs. These were proprietary, and therefore not appropriate to be standardized upon. However, AT&T has published a complete specification for its Unix V OS in the System V Interface Description (SVID). This has raised the possibility of using a UNIX-based standard for software products developed for, or delivered to, DoD.

Interim GFE APSE

An alternative approach to specifying a standard is to require the use of a particular programming support environment as government-furnished equipment (GFE). Again, the scenario simulations cannot speak to the technical desirability of a particular environment. Assuming reasonable parameters for cost, effectiveness, time to implement, and acceptance, the scenarios simulation the impact of such a policy on programming expenditures.

Convert older environments to APSE

Just as the DoD is plagued by a multiplicity of languages, it is plagued by a multiplicity of programming support environments. Just as with languages, there are a variety of excuses to customize an environment: a different target or host machine, a different language, or just passage of time creating desires for new features or changes to the old. One possible standardization strategy is to create an orderly migration process of programs on to a modern, standardized environment.

Although such a policy is not currently being discussed seriously, it is included here for further investigation, as it has potential to have very high leverage. For example, the survey being conducted as part of this study may reveal that a substantial portion of maintenance programming is being done in fairly old and primitive environments. Programs in those environments could be migrated to a modern standard environment with only the effort it takes to create a compiler, linker, loader, and debugger on the standard OS that is compatible with the original language. Such a migration would make available a wide array of programming support tools for maintenance, and make finding people experienced with the OS much easier. Moreover, adding maintenance programming to the list of users of the standard environment would make the market much larger for tool creation, and thus benefit all of DoD programming activities.

Require standard APSE only for maintenance

A variant on all policies that standardize an operating system, by specifying either a specific GFE'd product or an interface standard, is to impose such a requirement only on systems when they enter the deployment and maintenance phase. This would leave contractors and DoD organizations free to choose any appropriate environment for the development phase.

Standardize on small number of operating systems

Just as one step toward a common MCCR language was the specification of 7 approved languages for MCCR use, one step toward a common operating system and tool interface (and eventually tools) could be mandating a small number of commercially-available operating systems. A relatively small number of OSs could span most applications and requirements for MCCR programming. For example, one possible set is DEC's VMS, IBM's MVS, AT&T's UNIX V, and Softech's ALS/N. The first two are probably the most common OSs now used for MCCR programming, and the latter two should offer competing transportable OSs for nonIBM and DEC hardware. (This policy is not shown on the matrix)

APPENDIX C

AREAS FOR FURTHER INVESTIGATION

Appendix C: Areas for Further Investigation

One purpose of the present phase of this project is to develop a prototype model demonstrating the feasibility of the approach. Rapid prototyping requires a quick treatment of many areas that would otherwise warrant more detailed consideration. This appendix records such areas that were noted during the development process.

Appendix C.1 discusses improvements to the multivariable model calibration procedures.

Appendix C.2 discusses next steps in the policy analysis.

Appendix C.3 describes questions and improvements to the cost sector.

Appendix C.4 describes questions and improvements to the Ada and non-Ada projects sectors.

Appendix C.5 describes questions and improvements to the language choice sector.

Appendix C.6 describes questions and improvements to the Ada and non-Ada infrastructure sectors.

Appendix C.7 describes miscellaneous questions and improvements.

Appendix C.8 describes the procedures for creating this report from the simulation software on the Macintosh computer.

Most items are listed by model sector, and then prioritized within each sector. Three stars indicates the items that promise the most immediate benefits and should definitely be included in the next phase of this project. Two stars indicate that the area is possibly important. One star indicates an interesting but probably postponable question.

Appendix C.1: Multivariable Model Calibration

*** Resolve maintenance versus development costs data conflict

There is a statistic that circulates in software circles that 80 percent of the life cycle cost of software is maintenance. However, computing the development/maintenance balance from other figures gives quite different results. Suppose, as the model does, that development projects last ten years and maintenance activities go for another twenty years. Even if the yearly expenditure on software in maintenance equalled that in development, only 66 percent of the life cycle cost would be in maintenance. Either the model assumptions about "dwell time" are quite inaccurate, or the two computations are using incompatible cost measures. The origin of the 80 percent figure needs to be tracked down, and its supporting data and assumptions analyzed and reconciled with the other information.

*** Change EIA '85 to the real dollar equivalents

As discussed in Appendix A.9, the correct empirical time series of software costs to compare with Total yearly cost (equation #50) would be an inflation-adjusted version of the EIA forecasts (equations #60 and #70). The present EIA forecasts are in current (i.e., non-inflation adjusted) dollars.

** Add current-dollar comparisons

DoD instruction 7041.3 requires that multiyear economic analyses give results in both constant (inflation-adjusted) and current (actual budget) dollars. When both the EIA forecasts, Total yearly cost, and all its inputs are in constant dollars (as per the preceding item), an inflation index should be added so that such quantities can be viewed in current dollar terms also.

** Understand basis of EIA forecasts of total programming costs

The DoD Digital Data Processing Survey by the Electronic Industries Association is the major information source on the total magnitude of DoD programming effort. The information is used to calibrate the number of MCCR development project starts in the model. There is some evidence that the forecasts are very conservative in the spending estimates, more or less counting only projects for which budgetary authorization already exists. As discussed in the Appendix on multivariable model calibration (A.9), the estimates of future development project starts should be best estimates, not conservative estimates. In effect, the ground rules that created the EIA forecasts have to be known before it is possible to deviate from them correctly to calibrate the model.

Appendix C.2: Policy Analysis

*** Reformulate to accomodate all candidate policies

Appendix B.6 describes how to represent a broad array of policy questions within the model. Those representations were examples created rapidly, as part of the prototyping process. Their purpose was to demonstrate that something like the present model could be used for later, more thorough, policy analysis. The next step in policy analysis is to review the proposed representations thoroughly, reformulate the model as necessary, and then simulate the various policy scenarios.

*** Evaluate scenarios with contingency testing

When the full set of scenarios is tested and generally satisfactory, the next step is contingency/sensitivity testing on the most promising policies, as per DoD instructions.

** Use survey to see coverage of UNIX, VMS, and MVS

The viability of specific policy recommendations about operating systems (OSs), or at least tool interface specifications, hinges upon the current conditions regarding incompatibility of operating systems. Is UNIX a common enough OS among the organizations actually doing MCCR programming that it is a viable candidate for an interim CAIS? Or is the MCCR programming world heavily dominated by two or three OSs from major manufacturers (like VMS from DEC and MVS from IBM)? If so, then standardizing on the two or three current leaders would be a possibility, at least for an interim standard.

** Investigate policy viability

As the policy implications of the scenario simulations become clearer, there will be a need for research beyond the modeling effort, to investigate the political, technical, and commercial viability of the potentially recommendable policies. As part of that investigation, the costs of implementing the policies should be evaluated. For some policies, DoD will expend very little by comparison to expenditures on all MCCR software. For example, the Ada program costs, in dollars per year, much less than the billions to be spent on programming. But other policies may involve substantial expenditures, such as developing a GFE'd APSE. As another example, little is known about the general viability of migrating large programs to new operating systems or translating them into Ada from non-Ada languages. A few examples exist, and study of them should yield insights on the practicality of policies involving migration or translation.

*** Get incompatibility cost information from Fisher & Hook

(Fisher and Hook, 1986) do detailed calculations of the cost of language incompatibility. Such calculations should be able to yield appropriate slope values for the aggregated, more general effects in the model, the effects of incompatibility of Ada and non-Ada infrastructures on cost.

**** Tie model relationships into software science literature**

Although Phase II of Task T-4-215 involved a literature search, the primary aim of that search was information on Ada, its acceptance, and background. With the model structure now explicated, it constitutes a fairly complete specification of the relationships needing justification in order to validate the model's policy scenario outcomes. Another round of literature search would be appropriate, this time explicitly linked to model validation.

*** Get long-term trend in real per-line costs**

It ought to be possible to find information from previous studies that would give a trend in aggregate programming productivity over 10 or 20 years. This would allow validation of the assumptions made in intializing and assigning reference values for the non-Ada intensity and incompatibility levels. In brief, the assumptions are that intensity of infrastructure has risen somewhat over 1975 to 1985, and incompatibility has risen significantly. For further details see Appendix A.9, "Multivariable Model Calibration."

Appendix C.3: Cost Sector

*** Consider different shape for effects of infrastructure on cost

The present Effect of intensity of infrastructure on Ada project costs (equation #150) is an S-shaped function of intensity, with the steepest part in the middle, and less steep at both extremes of very little and very much infrastructure. (Boehm, 1981) argues for treatment of software cost as a classic production function, where the factors of production are the various components of the programming infrastructure. Under this view, the most realistic shape for the curve would be steepest at the extreme of no infrastructure, with cost declining as a function of intensity of infrastructure, but at an ever-decreasing rate. Such a curve would show the classic "diminishing returns to factor." This makes sense from a practical point of view, too: the tools that get used first are presumably the tools that are most useful and productive, with the more sophisticated tools making less difference in absolute terms (dollars per year saved).

*** Formally derive cost curves from COCOMO

The COCOMO model for software cost estimation (Boehm, 1983) explicitly represents the impact of most components of intensity of infrastructure on programming costs. It should be straightforward to correlate the meaning of the intensity of infrastructure indices with the inputs to the COCOMO cost drivers to deduce at least narrower bounds for the effect of intensity of Ada and non-Ada infrastructure on cost.

*** Consider curve for Incentive from relative intensity of infrastructure

The present curve for Incentive from relative intensity of infrastructure (equation #1210) gives an incentive of -0.53 at zero intensity of Ada infrastructure. That value should be -1.0, indicating complete unsuitability as a programming language.

*** Raise Target fraction for Ada starts

The target fraction for Ada starts (equation #1030) rises to 50 percent of new project starts targeted for Ada. This represents a fairly lax implementation of the DeLauer memorandum, and is probably too low; a figure closer to 100 percent would be more realistic.

An interesting alternative would be to experiment with defining the scope of the model as all DoD programming (not just MCCR), to explicitly represent Ada use in non-mandated applications.

** Use survey to develop realistic characterization of "projects"

It is sufficient to define what a standard project is by defining how much it costs to do per year in 1985 constant dollars with a given programming technology. But the survey and other sources may give a more multidimensional definition, including how many lines of a certain type of code it encompasses, how many people work on it in its various stages, and how many standard projects go into a major program like MIS, SDI, the B-1, an aircraft carrier, and so on. With such information, the definition would imply other characteristics, such as reference costs and duration of the development and maintenance phases. Also, although some information on the aggregate cost of software production per year exists, there is little hard information on how long projects and their payment obligations last. The survey will give a sampling of real projected lifespans for development and maintenance phases of projects.

*** Examine the correctness of the treatment of conversion costs**

It is unclear that the cost of conversion is exactly correct, in response to the following thought-experiment. Suppose Ada development costs were identical to non-Ada development costs (per project-year), and maintenance costs, especially the component representing redevelopment costs, had the appropriate relationship to development costs. Then turning on the conversion projects should make no difference to cost--the projects in the conversion level would merely be costed explicitly where before they were costed implicitly. It is unclear that the various time constants and costs, both actual and implicit, have the required relationships among them to make this so at present.

Appendix C.4: Ada and Non-Ada Projects Sectors

*** Consider measuring the Fraction Ada use by cost rather than by project

The present formulation uses a Fraction of Ada projects as an input to the process of creating infrastructure, representing both creation of experience and reusable programming, and the perception of a sizable market for programming tools and other forms of investment. At one point it was decided not to use cost because if Ada programming became very cheap relative to non-Ada, a fraction based on cost would "underestimate" the complexity, size, and amount of Ada programming under way.

But perhaps the issue should be decided in terms of how the Fraction of Ada projects is used, which is to influence the creation of infrastructure. If thousands of lines of code are generated at the touch of a button, this does not generate programmer experience, and may not generate reusable code. No one would perceive a profitable market for tools to increase efficiency. The real market would be where the money is being spent, which would be non-Ada programming. This might be a market mechanism that would hasten buildup of Ada infrastructure in the beginning and retard it near the end, by comparison to the present model. The mechanisms seem to be quite different from those captured in the effects of relative infrastructure.

*** Change weightings in Fraction Ada use to be consistent with costs

The present weightings for Ada projects are 1.0 for development projects and 0.5 for maintenance projects. The implicit assertion is that per year, maintaining a programming project takes 50 percent as much programming effort as it did during development. But in the cost sector, a maintenance project always costs 40 percent, not 50 percent, of what a development project costs per year. Given that the relative figures in the cost section are well-justified heuristically, if the present formulation is retained, the weighting for Ada maintenance projects whould be changed to 0.4.

** Correct the injection dates of first Ada compilers to 1983 from 1981

The dates of the insertion of infrastructure should be changed from the present 1981 to 1983, when the first Ada compilers actually were validated (see the description of equation #1390).

** Add free-market representation of language choice in upgrade section

Conversion upgrades promise to be a very important policy area, where some detail might be justified. In particular, It cannot be assumed that DoD has absolute leverage over the language conversion process, any more than it has absolute leverage over the original process of programming in a language. If it desired to be highly realistic about how much the DoD might be able to increase Ada programming over the late 80's and early 90's, it is necessary to look at the "free market" forces that might indicate when Ada should be used in major software upgrades, and how much the DoD can (and would want to) influence the economics of the process.

** Improve distinction between development and redevelopment

In addition to defining the average size and characterictics (especially cost) of our standard development project, a good operational definition is needed of when programming is a development project and when it is a major redevelopment, which is part of maintenance. This relates to the time to obsolesce maintenance projects. If people start

entirely over with every major redevelopement, then the average maintenance lifetime of the software is much less than that of the weapons or C^3I systems. If shorter-lived weapons or C^3I systems borrow software from previous systems, however, the lifetime of the software is longer than that of the average system in which it is embedded. This has important implications for the number and infrastructure available for non-Ada projects especially, and the desirability of policies, such as conversion or migration to standard operating systems, that deal directly with non-Ada programs.

*** Consider aggregating development & maintenance if costs are close**

If it turns out that the cost profile is roughly the same for both development and maintenance projects then the structure could be simplified by aggregating them into one projects level. The primary reason for treating them separatly is to represent the potential of different associated costs, as well as being more precise about when system obsolesce.

**** Determine if the best time for program translation is during upgrades**

Implicit in the choice of parameter values for Conversion project starts is the assumption that the time when programs are most cost-effectively translated is when they are being changed anyway, either in small ways during routine maintenance or in large ways during major redevelopments. Is this in fact true? An alternative assumption says that programs should be translated exactly, so that the full battery of tests (drawn from the old programming effort) run on the new code. Then, once the new code is working and well-documented, modify it, since modifications are supposed to be much easier in Ada. In scholarly research on the process of technological change in manufacturing, one of the big causes of failure to successfully change to new production technologies is trying for too much: simultaneous and large changes in both production technology (analogous to what language is used to program) and product technology (analogous to the program specification).

Perhaps there is a wide spectrum of appropriate procedures in practice. There is little point in translating a poorly-structured, poorly-documented or tested system whose requirements are dramatically changed anyway; any one of those three attributes in the extreme would make literal translation ineffective. Poor structuring (especially of assembly-language code) probably causes semi-automatic translation to become mostly manual. Poor documentation and testing implies that what the program is supposed to do is not very discoverable; in effect, the conversion team would have to write specifications and tests for the old system before it could be well-translated. It would be better to proceed straight to the new specifications and implement them in the new language, Ada.

But there must be situations where the non-Ada code is well-structured and well-documented. In such situations, translation followed by modification would seem the most efficient course. Even if the mission of the overall system had been changed, many of the specifications and test requirements would carry over directly from the non-Ada materials. Similarly, much of the code should be translatable nearly automatically.

*** Represent added complexity of upgraded software explicitly**

If in later phases of this research the definition of a project equates to a programming task of some defined level of complexity, then as requirements are increased in major upgrades/redevelopments, the model should show an increased number of projects in maintenance, i.e. more complex software to be maintained in the future. At present, this blowup in complexity and therefor presumably cost could be represented by the reference

cost per project of maintenance projects being higher than cost per line might indicate (because there would implicitly be more lines).

*** Settle on terminology of conversion or translation**

The model represents entities called conversion projects. They have so named rather than simply translation projects because more than simple translation from non-Ada to Ada may be involved. For the Navy piecemeal work, the new programming is actually in Ada, with translation from Ada to CMS2, not the reverse. And for shifting to Ada as part of redevelopment, there is mostly new programming going on, not translating. So the model uses a new term, conversions. But will that term lose more in obviousness than is gained in accuracy by not using simply "translations"?

Appendix C.5: Language Choice Sector

** Consider adding trend in Ada use

Given that incentives are supposedly representing perceptions about the future of Ada, perhaps a perceived trend in Ada use is a legitimate input. The "bandwagon effect" is certainly real.

* Reconsider input to incentives from incompatibility

In a situation where the infrastructure is low, there is an attractiveness to contractors if there is a choice between several Ada environments, each with its own strengths and weaknesses. If such an influence is represented in the model, it should have moderate gain for moderate incompatibility, but saturate quickly--vast numbers of incompatible environments should not hold vast attraction for users. The model should show a potential short-term attractiveness to diversity of choice. being overtaken in the long term by intensity of infrastructure.

The present formulation without an attractiveness of many infrastructures assumes that a variety of features is really represented by the intensity of infrastructure, and that having these features dispersed among a variety of incompatible systems is purely a disadvantage.

* Reexamine assumption of Ada fitness

The preliminary literature search done for this project revealed that the use of Ada and expansion of areas of application was growing rapidly, with many favorable reports from projects using it. A useful and interesting extension to this work might be a more detailed examination, now that some experience has been gained, of the suitability of the design of Ada, as well as its implementations.

Appendix C.6: Ada and Non-Ada Infrastructure Sectors

*** Explore differences in the creation of Ada and non-Ada infrastructure

In most simulations, Ada infrastructure gets to four times the non-Ada infrastructure, and 1/10 the cost. Are these numbers pluasible? Would the presence of advanced Ada infrastructure by itself reduce the creation of non-Ada incompatibility (and therefore allow more non-Ada infrastructure to accumulate)? There should be more analysis and discussion on how costs can get so different from one another.

** Consider eliminating coverage if it is redundant with intensity

It is unclear that having an explicit measure of coverage contributes to the model dynamics, for the same effects run through intensity of infrastructure: Intensity has the same inputs and effects the same things (and more) that coverage does. Coverage seems like a candidate for simplification if simplification is desired.

*** Add an injection incompatibility of infrastructure for SVID

At present, the adoption of SVID as an interim or permanent standard operating system leaves incompatibility of Ada infrastructure untouched, which is also the case in the commercial APSE scenario. Yet there are many sourcess of incompatibility within the UNIX world. There are two different shell languages, C and Bourne, user- or installation-defined utilities in the hundreds, and so on. It would be well to recognize such diversity (and therefore potential incompatibility) as one of the costs of an SVID standard. A model "policy lever" allowing the injection of incompatibility of Ada infrastructure would add to the flexibility and usefulness of the model.

*** Activate effect of technology on creation of infrastructure

The equation description indicates that there are real effects of technological change that exogenously impact the creation of infrastructure. As pointed out in the "Model calibration" section, the model parameters should not be set to make "conservative assumptions," because what is conservative for one policy question may be anything but conservative for another policy question. The proper choice fro the Rate of technological progress (equation #1580) therefore is not 0.0 but a best guess at how fast the ease of contributing to infrastructure is increasing. A value 3.0 percent per year or somewhat less seems appropriate. (For further discussion, see the text for equation #1580 in Appendix A.7.)

*** Use nonzero discount rate

Again in the spirit of using best guesses rather than attempting either conservative estimates or simplifying assumptions, the present model's representation of the comparison of present and future expenditures should be changed. Currently, the model treats them as comparable, with a discount rate of 0.0. A DoD instruction suggests 10 percent per year as a discount rate, but that seems perhaps colored too strongly by the interest rate experiences over the last 5 years. By financial theory, the discount rate should be a function of the expected real interest rate (if constant-dollar, i.e. inflation-adjusted flows are being discounted). 5 percent seems appropriate, being a compromise between the real interest rates of the last 4 years and the much lower average historical rate.

**** Consider reformulating infrastructure incompatibility as a coflow**

There is a subtle formulation issue related to the representation of incompatibility of Ada infrastructure. It might be better to use a coflow structure to more accurately capture the effect of large changes in the number of projects over the decades on the incompatibility of infrastructure. There is some incompatibility of Ada infrastructure right now because there are several environments, but for a number of projects that is small relative to what is to come. If a standard environment is mandated, during a time when vast numbers of new starts are using Ada, the large numbers of the new projects with the standard environment should rapidly overwhelm numerically the small number of projects done with the incompatibility environments, such that Ada incompatibility should disappear quite rapidly given a standard environment. If so, then a coflow structure would be better than the present formulation, which does not take into account major differences in numbers of new projects in determining incompatibility.

**** Reconsider effects of scale on creation of infrastructure**

The current formulation implicitly links the amount of programming (Ada and non-Ada) going and the infrastructure they create. There is an explicit influence of the fraction of Ada and non-Ada projects, but buried in this formulation there is an assumption about the absolute amount of programming going on as well. For some components of the infrastructure, absolute scale doesn't matter to the creation of average intensity. Programming experience is an example. But for other components, most notably software tools and program libraries, absolute scale does matter. Doubling the total amount of programming going on would increase the tool creation, purchase, and use, even if there wasn't an actual doubling. Over the course of the simulation, the scale of total programming efforts changes considerably, so it would be preferrable to have the scale issue better thought out.

Arguing for the present formulation, there is considerable infrastructure to be gained from university graduates utilizing more advanced programming techniques. The present formulation based on a constant flow of creation of infrastructure, modified to some extent by the amount of Ada programming, captures this idea.

**** Tie loss of incompatibility to that of intensity**

As the equation description explains, at least a portion of the loss of incompatibility of infrastructure is tied directly to the loss of intensity, which is in turn tied to the obsolescence of projects. Currently all of these rates of flow have independently-specifiable parameters, when perhaps they should all have the same parameter, or otherwise show the linking in a robust way. (With some parameter settings, the intensity can disappear but leave the incompability, or vice-versa.)

**** Reconsider the distinction between Ada and non-Ada infrastructures**

In the beginning, it seemed that Ada programming would go on only in PSEs that conformed at least in spirit to Stoneman, which meant that programming would be done in substantially new environments, different from the environments being used for non-Ada programming. But there are ever-increasing numbers of Ada programming projects under way. What environments are they using? Are there any particular constraints on what environments are permissible? Is the model correct in the way it represents the accumulation of infrastructure (which includes the APSEs) as quite seperate from non-Ada infrastructure?

**** Increase Natural fraction of Ada use.**

In the present model, if the Ada infrastructure intensity and coverage are equal to the corresponding values for non-Ada, 50 percent of new development project starts will use Ada. This number may be too low, if Ada is at all a good language in its own right. In addition, the natural fraction, even though it represents what language would be chosen in the absence of DoD pressure, must still represent the knowledge that Ada is a standardized language, and hence would have an advantage over many non-Ada languages.

**** Make "Waiting for CAIS" scenario more realistic**

There is a scenario where the changes representing standardization on one or more commercial APSEs are implemented 5 years later to represent a policy of waiting for the custom-designed CAIS. That simulation is optimistic, both by the mere 5-year wait, but also by creating as much new infrastructure as standardizing on existing PSEs. A more realistic scenario would inject considerably less infrastructure, for there would be virtually no experience implementing or using a custom-designed GAIS that is quite different from any other operating system interface.

**** Use survey to calibrate current incompatibility and intensity**

To know how much improvement in productivity is available from more intense infrastructure, perhaps brought about by standardization policies, data is needed on the present status of MCCR programming: what level of sophistication of tools, etc. is actually out and in use, and how incompatible the tools, operating systems, and languages are.

If the survey reveals that a considerable amount of maintenance programming is being done in environments that are very impoverished, that implies considerable leverage for a policy of transferring existing programs to a better-equipped PSE for further maintenance, maybe even well before (if ever) a program is translated into Ada.

*** Consider representing Non-Dod programming explicitly**

The model obviously details DoD programming; the representation of non-DoD programming is left implicit. Future models may offer additional insights by explicitly representing non-DoD programming. But at present, there is no overwhelming hypothesis as to why the dynamics of Ada use and Ada infrastructure development would proceed differently in the non-DoD world. In the absence of such an hypothesis, the model as it stands seems satisfactory in its implicit assumption that both non-Ada and Ada programming will occur in the non-DoD world, and contribute to development of Ada and non-Ada infrastructures responding to the same set of incentives as governs the DoD programming. If Ada were used *only* by the military, this situation would be represented in the model with a lower normal creation of infrastructure.

*** Improve formulation for coverage**

The present formulation of indicated coverage and actual coverage does not mirror the actual processes very closely. In real life, coverage expands based on perceived market for using a particular host/target combination. Then along come new combinations, which reduce the coverage. Combinations come along either because of new hardware, perhaps designed especially for the military, or newly-popular civilian hardware, such as the 8086 and 80286 chip sets made popular by the IBM PC. It should be possible for coverage to rise to nearly one, then fall if extension of coverage does not keep pace with new hardware developments. The present formulation allows no such "obsolescence" of coverage.

*** Consider renaming obsolescence of intensity as loss of intensity**

Obsolescence of intensity of Ada infrastructure now represents both the antiquation of infrastructure (being surpassed by newer technologies) and the continued loss of programmers etc.that are not obsolesced but merely retired or moved on to nonprogramming positions. If the dual meaning becomes confusing, perhaps Obsolescence of intensity ... should be renamed Loss of intensity ... where loss would more clearly represent both processes.

*** Consider seperating infrastructure for development and maintenance?**

It may be that intensity, coverage, and incompatibility for maintenance projects and new projects must be represented seperately. When the production technology of new software is standardized, all of the ancient systems currently deployed will remain unaffected. Their infrastructure is simply no longer added to, and may be decreased if people (programmers, managers, and teachers) are lured away to other languages and operating systems. As the model is presently structured, incompatibility is simply lost with a 30-year average lifetime.

There are various scenarios and policies that would make it desirable to keep track of the characteristics of the infrastructure for maintenance programming seperately from the infrastructure available for development programming. One such policy is permitting any development environment to be used, but specifying that a programming project be turned over to be maintained on a standardized environment. Although the effects of such policies can be simulated approximately with the present aggregate infrastructure, more detail might give more confidence in the simulated results.

Perhaps a reasonable interim step would be to structure a single aggregate set of infrastructure characteristics (intensity, incompatibility, and coverage) explicitly as coflows. Startup projects would have some level of these characteristics, and a separate part of the formulation could specify the extent to which existing projects (the aggregate of new and maintenance) can "retrofit" to get the (presumably higher) level of new infrastructure. This should allow more explicit linking of obsolescence of infrastructure with that of projects. Will it suffice to control the fading away of non-Ada infrastructure when Ada becomes very popular?

Appendix C.7: Miscellaneous Suggestions

** Produce a glossary with abbreviations, full names, and equation numbers

A fully indexed glossary of variable names, abbreviations, and equation numbers would be helpful. Even with a model of only moderate size, the task of finding out where a variable is, or what it means can be clumsy and time-consuming.

** Make a fold-up complete flow diagram

The current documentation contains a complete flow diagram, however, it is broken up into sectors, and scattered throughout the text. It is left to the reader to deduce the way the whole model fits together. The Stella software used in this project can produce a large (4'x5') and complete diagram of the whole model. For those with good eyes, the laser printer can shrink the flow diagram with lettering down to about the size of 4 8 1/2"x11" sheets. In either case, the equations would have to be hand-numbered. The major drawback is that the document becomes difficult to reproduce. Such a diagram is well worth the trouble for anyone actively using the model.

* Check reference conditions

Many formulations throughout the model rely on the "reference and multipliers" format, which is explained with equations #100, #1500, and #1660. This formulation defines a variable in terms of what it would be (simply equal to the reference value) under some set of reference conditions, modified by multiplicative factors representing the impact on the variable of conditions differing from the reference conditions. In principle each equation can define a different set of reference conditions, even two different reference values for the same input variable. In practice, it would be good to review the reference conditions and make them as consistent as possible, to allow a simpler presentation and reduce the possibility of human error in the calibration process.

Appendix C.8: Guide to Document Assembly

To facilitate later modification of this report, this appendix describes the process by which the report was produced, primarily on the Macintosh 512K personal computer. Microsoft Word produced the text, and contains many of the graphics as well. Most of the graphics were transferred electronically from the STELLA simulation package (Richmond 1985). Some graphics, however were transferred via hard copy. The following sections provide guidance on how to accomplish these various steps.

Guide to underlying disk files

The disk files used to create this report occupy several standard 400K Macintosh floppy disks. Each disk is labelled "T-4-215 Report disks x of n," where n is the total number of disks and the disk in question is the xth one. Each file name likewise contains the project number. Files for the various appendices are the most numerous; they can be identified quickly by the beginning characters, which are the letter of the appendix, a period, and the number of the section, e.g. "A.0." The report's text is contained in the following Microsoft Word documents:

A.0 #1 Appendix A intro. T-4-215
A.1 #1 SD term. & symb. T-4-215
A.2 #1 Standard Abbrev. T-4-215
A.3 #2 Cost Sector T-4-215
A.4 #2 Ada Proj. Sector T-4-215
A.5 #2 Non-Ada Proj. S. T-4-215
A.6 #2 Lang. Choice S. T-4-215
A.7 #2 Ada Infr. Sector T-4-215
A.8 #2 Non-Ada Infr. S. T-4-215
A.9 #2 Multiv.Mod.Calib. T-4-215
B.0 #1 Appendix B intro. T-4-215
B.1 #1 Model listing T-4-215
B.2 #1 Variables Plotted T-4-215
B.3 #1 Base Scen. T-4-215
B.4 #1 Comm. APSE Scen. T-4-215
B.5 #1 Conversion Scen. T-4-215
B.6 #1 Policy Matrix T-4-215
C.0-8 #1 Further Invest. T-4-215

The three primary scenarios (base, commercial APSE, and conversion) each have a stored model that without further alteration produces the plots and tables shown in Appendices B.3 through B.5. Those models appear respectively on the disk as STELLA documents:

SSM0.32 T-4-215
CAPSEM0.32 T-4-215
CONVM0.32 T-4-215

Two more models appear on the disks. They are identical to SSM0.32 T-4-215, the base model, except for the layout of the stored flow diagrams. The models above all have the same flow diagram, layed out to create large, wall-mounted flow charts printed on the Imagewriter dot-matrix printer. The additional models have flow charts layed out to create the figures in this report. The models are:

Figures for Appx. A T-4-215
Figure for Appx. B.1 T-4-215

Sections below discuss how to create graphics from the various models.

Electronic transfer: Equations and graphic functions into Word

STELLA displays information on the Macintosh's medium-quality bit-mapped graphics screen. The screen images of the model equations and the graphic functions were taken electronically and used in the Word documents.

The fundamentals of the process are straightforward. One runs STELLA and displays the desired picture (equations are shown by choosing "Display" from the main menu and the "Equations" menu item; graphic functions are displayed by double-clicking either the equation desired or its flow diagram symbol). Pressing the command, shift, and 3 keys simultaneously creates a MacPaint document on the disk. Using some means of selecting part of the screen image and putting it onto Macintosh's clipboard (more on that below), one runs Word and pastes the image into the document (either from the menu or command-v). Pasting automatically left-justifies the inserted figure with respect to the margin. To center the graphic function's graphs, either use the ruler to indent 1 1/2 inches, or click on the image to select it, click again on the border that appears, and drag it to a 1 1/2 inch indentation. There are two complications to this process: setting up to do large numbers of such images, and getting selected parts of the image onto the clipboard.

The system of using screen dumps is somewhat awkward for doing large numbers of screen dumps. Each time command-shift-3 is pressed, a disk file is created (containing the screen image in bit-mapped form) and named. The first file is named Screen0, the second is Screen1, and so on up to Screen9. If command-shift-3 is pressed when 10 screen dumps with those names already exist, the Macintosh will beep and nothing further happens. The ten files need to be renamed so that 10 more screen dumps can be made. The time required is prohibitive to simply quit from STELLA and return to the Finder (which is the user interface for the operating system that creates the well-known desktop). Instead, use the Apple Switcher program, which can load several applications into memory at once. (Word will run in 128K in the Switcher, but STELLA prefers about 200K; set these allocations when first entering the Switcher by choosing "Configure then Install" under the "Switcher" main menu item, and for each program enter the desired size into both the preferred memory size and the minimum memory size.)

With STELLA and the Finder loaded into the Switcher, one can make ten screen dumps in STELLA, switch to the Finder to rename them, and switch back to STELLA quite rapidly. Be sure to mark on a hard copy listing which screen dump documents show which equations. The whole procedure is best done with a hard disk, because storing the bit-mapped graphics fills 400K floppy disks fairly rapidly. (The models here will fill two floppies, without anything else on the disks, which is awkward when trying to run the system and an application, too.)

The second complication is getting the right parts of the screen images into the clipboard. The screen images record literally the whole screen, only part of which will be

the equation or graphic function that the report is supposed to contain. By far the best way is the Paint Grabber desk accessory, a widely available desk accessory that, in the midst of any application (Word, in this case), can open up a MacPaint document, select a rectangular portion of it, and place that on the clipboard. (An alternative is described below.) Paint Grabber is available from any Apple dealer, either off-the-shelf, or within a few days by ordering. See the Macintosh instruction manual on how to install desk accessories and install Paint Grabber on the system you will be using.

With the system containing Paint Grabber and the screen dumps available online, open up Word and find the location where something is to be inserted. Under the Apple symbol on the main menu, select Paint Grabber. A new main menu item will appear, "Art Thief." From that, choose "Open." An option box will appear with all of the MacPaint documents available from which to choose. Using the list of screen dump names, find the name of the MacPaint document that contains the equation or graph you want. Scroll to that name in the option box, click on it, and click on "Open." (Note: as for most all commercial software, subtle bugs remain in Paint Grabber; one has better luck selecting a file and using the Open button than just double clicking on the file.)

The screen image will appear, and you can click and drag to form a rectangular boundary around the part of the image you want. Under the "Edit" main menu item, choose "Cut" or Copy." (Note: the "Art Thief" main menu item also has a "Display Clipboard" and cut option called "Swipe." Using these sometimes puts things on Paint Grabber's clipboard, but you have to display that, select your selection again, and use the Edit menu's cut before Word's clipboard will hold the selection. These features are quircky but workable—just experiment.) With the right selection on the clipboard, click on the Word window to activate it, put the cursor at the location for pasting, and paste (either from the Edit menu, or with command-v).

If Paint Grabber is not available, one can use the Switcher to allow MacPaint and Word to operate closely. Make sure that both programs are looking at the same clipboard by choosing "Options" under the "Switcher" main menu item and putting an X in the "Always convert clipboard" box. MacPaint is more awkward to use than Paint Grabber, since MacPaint seems to be limited in how large an object can be put in its clipboard—equations are wider than the MacPaint window. There is a MacPaint-like program for larger bit-mapped documents called "Paint Cutter" that may be more suitable. Even so, Paint Grabber seems perfectly designed for the task at hand, and is the preferred method. It's much faster than the instructions above might make it appear, and is much faster and higher-quality than physically cutting and pasting originals.

Hard-copy transfer: Flow diagrams, plots, tables, and listings into document

Given that legible flow diagrams are larger than the Macintosh screen, these appendices use STELLA's facility for printing flow diagrams and portions thereof with a laserprinter, and use nonelectronic means to put the images into the report. This procedure takes advantage of the high-quality laser output. The flow diagrams in the two models, Figures for Appx. A T-4-215 and Figure for Appx. B.1 T-4-215, are arranged so that the laser printer will print each portion of the flow diagram on a separate page, and positioned in the middle of the page.

To laser print the flow diagrams, choose "Choose printer" from under the Apple symbol on the main menu, and choose the LaserWriter. (Make sure there is about 60K left on the disk for scratch space for the printing process.) Then display one of the two specially-prepared models in STELLA. Choose "Diagram" under the "Windows" main

menu item, and under the "Display" main menu item choose "show pages" to show where the boundaries between printed pages will lie. Under the "File" main menu item choose "Page setup" and specify reduction to 60 percent. You should see page boundaries coming in between parts of the flow diagram. Under the "Display" main menu item hoose "Full size" and "Hide pages." Under the "File" main menu item choose "Print." If you want only part of the flow diagram, specify the page or pages to print. Choose "Full view" under the "Display" main menu item to see the entirety of the model, but change back to "Full size" before printing, or the LaserWriter will create a cute little one-page diagram of the whole model. STELLA numbers pages like MacDraw: number one is the upper left corner page, number two is below it, and so on. So, for example, the top row of pages in the models here are pages numbered 1, 5, 9, 13, 17, and 21. Because of the way the models are layed out, pages 20, 21, 23, and 24 are blank.

Where the flow diagrams go,the Word document pages contain nothing but the heading above and the caption below. Laser print those, copy them on a Xerox machine or whatever, and then put the copies back in the paper feed of the copier. The copier is now set up to copy an image onto a page already showing the heading and the caption. Just copy the flow diagrams onto these pages in the correct order. In the stack of LaserWriter text output from Word, just replace the pages with the copies (which now show the flow diagram with caption on a properly numbered page) with the corresponding page numbers.

The procedure for putting plots, tables, and listings onto properly headed and page-numbered pages is exactly the same, except for how the originals are laser printed and assembled. Under "Windows" choose "Graph Pad" for plots or "Table," and then under "File" choose "Print Graph" or "Print Table" (only the correct option will appear). On the LaserWriter menu that follows the selection of printing, choose reduction of 78 percent for plots and 100 percent for tables.

Creating hard copy for equation listings is slightly more complex, because STELLA insists on not recognizing page boundaries. Obtain a printout by choosing "Equations" under "Windows" and then "Print Equations" under "File." (If the screen display shows blank, it is displaying the changes made recently, which are hopefully none. To get back to the full display of equations, select "Show All Equations" under "Display.") On the printout there will be equations whose top half is on one page and whose bottom half is on the next. Obtain a listing with these "page breaks" in different equations by going to the flow diagram and creating a fictitious level or two with names like "AAAA" that would put them at the beginning of the listing. Then print the equations again. Now there is enough material to cut and paste an equation listing onto 8 1/2 x 11 pages with proper margins. Doing the trick with the copier as described above will then put the listing on pages with headings and page numbers.

After comparing the relative effort for physical versus electronic cutting and pasting, the electronic procedure seems easier, more reliable, and gives higher-quality results. There is nothing to be done about flow diagrams, but they are the easiest output to physically prepare (there is no actual cutting or glueing). But the listings, plots, and tables should all be done with screen dumps if they are ever done again. The screen dumps that would make up the listing are all available from the process of pasting equations inot the text. According to the Addendum for Microsoft Word 1.05, the plots can be reduced in a controlled way as follows: do a screen dump and paste it into the Word document as described above. Then type Command-Shift-Y and then 7, which should reduce the figure to 70 percent of its original size. The screen display of resized screen dumps looks awful, but it comes out fine on a laser printer. Use 6 for 60 percent reduction, 8 for 80 percent, and so on. Command-shift-Y then 0 restores the figure to full size.

For fancier plots for the main body of the report or whatever, screen dump the plots, then open the documents with MacPaint to trim off the date or scaling numbers if they are too small to read. Put the whole thing on the clipboard (which should be doable from the page-size display in MacPaint), and open up MacDraw to put on nice labels. (They could be done in MacPaint, but they couldn't be repositioned and they wouldn't print out as well on the laser printer.) In MacDraw, use rectangles with invisible edges and filled with solid white to create a pure white background for the captions. If the text is "Grouped" (command-G or menu selection) with the rectangle, the text automatically rewraps itself to fit within the rectangle. Then put the MacDraw object in the clipboard and transfer to Word, and paste it in at whatever reduction is appropriate.

The figure showing the matrix of policy levers versus policies in Appendix B.6 is reduced from an original created in Audrey Hook's office on a Macintosh. The graphics in the main body of this report were artist-created from full-sized laser output of the models' plots.

Assembly, pagination, and printing

Pagination in the present draft is not well-done, for historical reasons. Most of the text in Appendix A originated in MacWrite, which divides paragraphs from one another with extra carriage returns. Word, by contrast, is set up to use a single carriage return to mark the end of a paragraph, and uses the paragraph formats menu to add an extra line between paragraphs. This difference becomes important when trying to arrange the text on pages such that no captions are orphaned at the bottom of the page, with the text starting on the next page. It is also desirable to have the short paragraph that precedes each model equation appear on the same page as the equation itself, or the graph, in the case of a graphic function. Word has facilities for keeping paragraphs together, but they are unreliable if there's an extra carriage return. They are especially unreliable when the following paragraph is a picture, to the point of crashing the system. Save the file before playing with pagination!

Cut the full-page spaces for figures and paste them at the end of the document before repaginating—Word cannot automatically put full-page figures after a full page of text, so if the document is repaginated with the figures in their old locations, quite a few nearly-blank pages will result. Also remove any other fixed page breaks in the document that do not start major sections. (These were inserted as a makeshift ways of having captions come out with their text.)

In retrospect, it would have been easier to go along with Word's system for paragraphs and pagination completely, just editing out all the extra carriage returns and using the option-command-click shortcut for copying the paragraph format that would keep the appropriate paragraphs and figures together.

Before reprinting a document, check the page layout (under "File") and the division layout (under "document"), since the right settings don't seem to have been saved with the documents consistently. The page layout for most of the documents is the default settings except for left margin of 1 inch and right margin of 1.5 inches. (For some reason, this is required to give 6 1/2 inches of text). The exceptions are Appendices B.3-B.5, which use a margin of 1/2 inch left and right, to accomodate labels for plots. Choose smoothing in the page setup menu to get the figures to look good, but do not choose font substitution. Even though the documents use a laser font throughout (such that there shouldn't be any font substitution), it somehow wipes out the underscores in variable names. Division layout uses all the default settings except for the running head starting 0.6 inches from the top. (0.75 was just too close to the text.)

APPENDIX D

QUESTIONAIRE

QUESTIONAIRE

PART I PROFILE OF INFRASTRUCTURE USED

General Directions for Completing:

There are two types of forms to be completed in the first part of this survey. The first type of form is labeled ATTACHMENT 1. It is the mechanism for collecting a data sample indicating the types of computers (host and target) and languages that are actually available and being used to develop/maintain MCCR software applications. The hardware, including operating systems, and languages are the minimal infrastructure for tools. This infrastructure also represents technical constraints that must be considered in future strategies for improvements in environments.

The second type of form is labeled ATTACHMENT 2. It is the mechanism for collecting a data sample indicating the tools routinely used for developing/maintaining MCCR applications written in the languages you identified on ATTACHMENT 1. The detailed directions for completing each attachment are stapled to each form.

Directions for Completing ATTACHMENT 1

Please complete ATTACHMENT 1 for each MCCR Program and for each software project (of that Program). ATTACHMENT 1 should be duplicated as continuation pages if more space is needed to list all the host/targets. There are two pages in ATTACHMENT 1. The first page has pre-printed languages at the top with space for one "other" language (e.g graphics language) which has a special purpose when used in combination with another language. The second page is space for you to identify languages used but not identified on page 1 (e.g. COBOL).

PROGRAM NAME _____

PROJECT NAME _____
STATUS: _____
(NOTE 3)

ATTACHMENT 1

HARDWARE USED		LANGUAGES USED								
Host Computers And Operating Systems (Note 1)	Target Computers	Ada %_____ (Note 2)	Jovial-73 %_____	Jovial-J3B %_____	CMS2 %_____	C %___	Fortran %_____	Pascal %_____	SPL/1 %___	OTHER %_____
A										
B										
C										
D										
E										
F										
G										

NOTE: 1. If you have several models of the same vendor's host and they all have the same operating system, identify a representative model (e.g. VAX7XX MVS 4.1)
2. Please estimate the percent of total work done in each language.
3. WRITE "D" for Development or "M" for Maintenance.

PROGRAM NAME _____ PROJECT NAME _____

ATTACHMENT 1

HARDWARE USED		LANGUAGES USED								
Host Computers And Operating Systems (Note 1)	Target Computers									
A										
B										
C										
D										
E										
F										
G										

NOTE: 1. If you have several models of the same vendor's host and they all have the same operating system, identify a representative model (e.g. VAX7XX MVS 4.1)
2. Please estimate the percent of total work done in each language.

Directions for Completing ATTACHMENT II

Please indicate the tools you use with a check mark. We have included nine forms (one for each language shown on ATTACHMENT 1); however, please duplicate and complete this form for any other language you use.

The following provides some guidelines on selecting the level of tools.

Meaning of LEVEL:

LEVEL I (BASIC) = tools available for microprocessors.

LEVEL II = Basic tools available for most minicomputers

LEVEL III = Tools avaiable on minicomputers and host computers which extend the operating system services.

LEVEL IV = Tools available on most large hosts (equivalent to Stoneman Minimal Ada Programming Support Environment (MAPSE)) as an extension of a virtual machine interface for programmers.

LEVEL V = Tools that may be available on large hosts (equivalent to Stoneman Ada Programming Support Environment (APSE)) which may include project specific tools.

The "cost driver" factors used by the COCOMO model can be derived from this classification of tools; therefore, we have selected this scheme for aggregating data that will be used in the Systems Dynamics Model under development.
(Reference: Boehm, Barry W.: Software Engineering Economics; Prentice-Hall, Inc, 1981 pp 458-466

PROGRAM NAME _____

ATTACHMENT 2
SOFTWARE TOOLS
LANGUAGE _____

PROJECT NAME _____

	CHECK THE ITEMS YOU HAVE AND ROUTINELY USE
LEVEL I (BASIC)	
ASSEMBLER	
LINKER	
MONITOR	
BATCH DEBUG AIDS	
LEVEL II	
HOL COMPILER	
MACRO ASSEMBLER	
OVERLAY LINKER	
LANGUAGE INDEPENDENT MONITOR	
BATCH SOURCE EDITOR	
LIBRARY AIDS	
DATA BASE AIDS	
LEVEL III	
TIMESHARING OPERATING SYSTEM	
DATA BASE MANAGEMENT SYSTEM	
OVERLAY LINKER	
INTERACTIVE DEBUG AIDS	
PROGRAMMING SUPPORT LIBRARY	
INTERACTIVE SOURCE EDITOR	

	CHECK THE ITEMS YOU HAVE AND ROUTINELY USE
LEVEL IV	
VIRTUAL MEMORY OPERATING SYSTEM	
DATA BASE DESIGN AID	
PROGRAM DESIGN LANGUAGE	
PERFORMANCE MEASUREMENT AND ANALYSIS AIDS	
PROGRAMMING SUPPORT LIBRARY WITH BASIC CM AIDS	
SET_USE ANALYZER	
PROGRAM FLOW AND TEST CASE ANALYZER	
TEXT EDITOR AND MANAGER	
LEVEL V	
PROGRAMMING SUPPORT LIBRARY WITH CM AIDS	
INTEGRATED DOCUMENTATION SYSTEM	
PROJECT CONTROL SYSTEM	
REQUIREMENTS SPECIFICATION LANGUAGE AND ANALYZER	
PROGRAM DESIGN TOOLS	
AUTOMATED VERIFICATION SYSTEM	
SPECIAL-PURPOSE TOOLS:	
CROSSCOMPILERS	
INSTRUCTION SET SIMULATORS	
DISPLAY FORMATTERS	
COMMUNICATIONS PROCESSING TOOLS	
DATA ENTRY CONTROL TOOLS	
CONVERSION AIDS	
OTHER	

PART II APPROXIMATION OF COMPATIBILITY AND COST

Directions for Completion:

In this part of the Questionaire, we need your approximation of factors that help quantify the magnitude of costs/benefits associated with current software development and maintenance practices.

1. How long do you normally use a tool? (i.e. the length of time in months/years between acquisition and replacement of a tool)_____months_____ years

2. The percentages you estimate below will give us an indication of the portability/re-usability of your tools. Please indicate percentages for the following (the total for all four cases should be 100%):

LANGUAGE SPECIFIC TOOLS:

Host Specific_____% Host Independent_____%

LANGUAGE INDEPENDENT TOOLS:

Host Specific_____% Host Independent_____%

3. The source of tools is also of interest. Please indicate by percentage the sources for the tools you use (the total for all four cases should be 100%):

Government Furnished_____%
Internally Developed_____%
Commercial Products_____%
Other (explain below)_____%

4. For each project you support, please estimate the "size" of the software by either the total lines of source code delivered to the customer/user OR the annual level of effort (programmers/systems analysts only) required to produce the software for which you are responsible. (If you can estimate both of these quantities, please provide them.)
Note: When estimating for development projects, estimate the expected size. If you count delivered source code by another unit of measure (e.g. words/statements) please label the units and the language.

PROJECT NAMES	LINES OF CODE	or	ANNUAL LEVEL OF EFFORT

(Note: Please provide a continuation sheet if required.)

PART III PERCEPTIONS OF THE "VALUE" OF TOOLS

Your answers to the following questions will be used to refine the perceptions of DoD with respect to cost factors which are difficult to quantify.

1. Have you kept any statistics that indicate how tools effect the productivity of application development or maintenance? Yes_____ No_____

a. If your answer is yes, please provide a contact who will be willing to discuss these statistics with us.

Name:_____

Phone:_____

b. If your answer is no, what is the criteria you use to make an investment in tools?

2. Please identify your most critical software engineering problems with a short rationale for their criticality.

APPENDIX E

WHITE PAPER

CONCEPTUAL FRAMEWORK FOR EXAMINING THE ROLE OF STANDARDS IN THE MCCR ACQUISITION PROCESS

Prepared by

Marko Slusarczuk
Sarah Nash
Tom Frazier
Peter Ashton
Stanley Dubroff

20 December 1984

APPENDIX E

LIST OF FIGURES

WHITE PAPER

CONCEPTUAL FRAMEWORK FOR EXAMINING THE ROLE OF STANDARDS IN THE MCCR ACQUISITION PROCESS

EXECUTIVE SUMMARY

Mission Critical Computer Resources (MCCR) are an increasingly important element in the development and deployment of major weapons systems. Effective management of the acquisition of MCCR is vital to maintaining a successful national defense, especially given the complexity and dynamic nature of computer technology and the diverse groups involved in the acquisition process. Standards can play an important role in that process by reducing uncertainty both for government and industry. DoD decisionmakers must have a method or framework in which to evaluate the relative costs and benefits associated with a particular standard. This White Paper provides such a framework. Prior work has looked at impacts on only one dimension, failing to capture all aspects of the problem. One unique aspect of our approach is its attempt to integrate each area which has been previously studied separately.

The conceptual framework describes the factors affecting the use of standards in the MCCR acquisition process. It also defines the relationships among those factors which indicate the impacts of standards. The objective in developing the framework is to provide high-level decisionmakers with an interactive analytical tool using real-time information for understanding how

standards influence and interrelate with the MCCR acquisition process. The framework is an initial presentation of the key factors and the interrelationship of those factors that describe the impact of standards. Two specific methodologies for evaluating the impacts of standards, the consumer welfare method and the risk/return method, are presented. By incorporating these methodologies, the framework consists of both a "road map" that assists decisionmakers in analyzing the potential effects of a standard and also contains the information necessary to rationalize a given policy decision regarding standards.

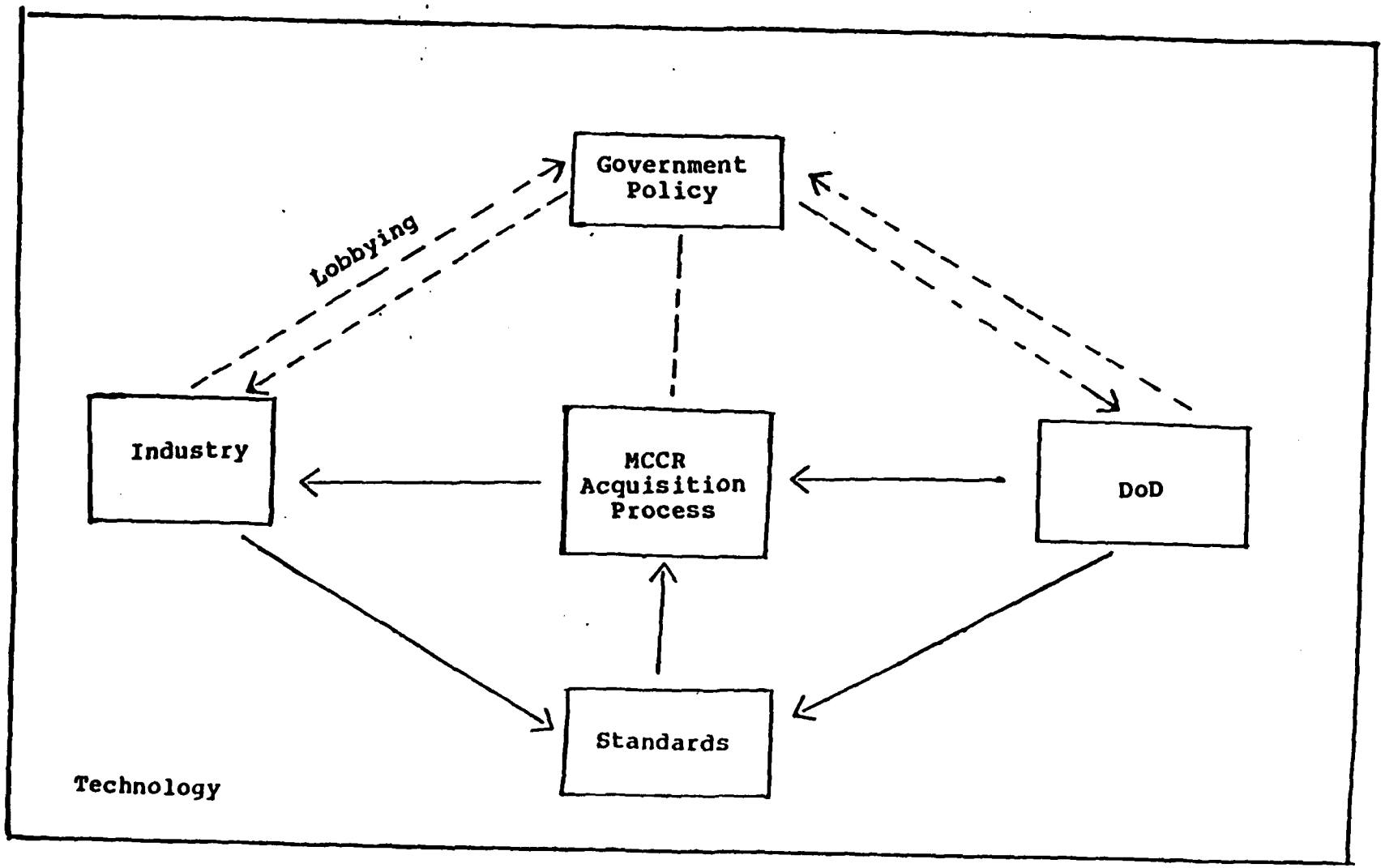
The framework will be modified, extended, and refined based on the results of a series of historical case studies which will serve as analogues to current standardization policy issues. A valuable by-product of this refinement effort will be the establishment of a database of relevant literature concerning standards and the MCCR acquisition process. Ultimately, the framework will be an interactive decisionmaking tool which will assist the decisionmaker in analyzing the appropriate implementation of standards and provide a rationale for that prescription.

The conceptual framework consists of several dimensions: the MCCR acquisition process, standards and the standardization process, the DoD, the computer and semiconductor industries, and government policy. The dimensions are linked as shown in Exhibit 1, and the arrows indicate the direction of influence. The linkages indicated with a dotted line, although possibly significant in the overall process, are not included in the analysis. The most significant linkage for the purpose of this discussion is the MCCR-standards linkage.

The framework also includes the environment of <u>technology</u> which influences each of the dimensions and the linkages. This

Exhibit 1

CONCEPTUAL FRAMEWORK AND LINKAGES



is one critical aspect of the framework because computer resources are driven by technology. Within the life cycle of the MCCR acquisition process, DoD must plan for and implement technology insertion. DoD must manage and consider important concepts such as matching the technology life cycle with the weapons systems life cycle and recognizing learning curve benefits from a technology. These considerations interact with the use of standards and can influence their impact in the MCCR acquisition process.

Within each component of the framework, there are many separate elements. For example, the MCCR acquisition process is viewed in terms of the phases of the life cycle as defined in A-109: mission analysis, exploration of alternative systems, competitive demonstrations, full-scale development and evaluation, production/performance appraisal, deployment and operation, maintenance and support, and retirement and disposal.

Standards perform various <u>functions</u> such as conveying information, promoting compatibility/interoperability/transportability, optimizing variety, and improving quality; these functions describe the impact that a standard will have. Various types of standards also help define their impact; these types of standards range from the most flexible to the least flexible: interface, process, and product standards. Finally, various groups are affected by standards and may desire them at any particular point in time. These groups include buyers, sellers (suppliers), users, and maintainers.

DoD and industry are the two primary groups involved and affected by the use of standards in the acquisition of MCCR. DoD develops certain strategies which are implemented as part of the MCCR acquisition process in order to meet various goals. These

goals stem from perceived threats to national security, operational and logistical requirements, and specific problems or issues unique to MCCR. Standards may represent an element of DoD strategy vis-a-vis the acquisition of MCCR. DoD writes standards in situations where they will reduce uncertainty in the MCCR acquisition process, assisting in maximizing quality and flexibility while minimizing cost and delivery time of a system.

Industry also formulates and implements strategies based on opportunities and constraints in the environment and the goals of each company and its organization. Standards can reinforce or reorient industry strategy directly. Standards may also raise or lower the risks faced by companies resulting in lower or higher expected payoffs from investment projects including technology insertion programs. Industry acts as a proponent of standards when it believes (collectively) that the standard will have beneficial impacts.

Standards interact with the MCCR acquisition process to raise and/or lower risk. The direction of the <u>impacts</u> on risk produces the beneficial and harmful effects. Standards can influence each of the phases of the acquisition process and various functions of standards are important at particular phases. For example, the quality function (performance standard) may be critical at the test and evaluation phase of the acquisition process to facilitate comparison. Successful standards tend to be more flexible at early phases of the acquisition process (and the technology life cycle) and less flexible at later stages. Standards can assist the technology insertion process, but careful analysis of the standard is necessary to prevent freezing the existing technology and locking out new technologies.

The analysis of the risks and returns of using a standard in the acquisition of MCCR depends upon assessing the impacts on all dimensions. The framework identifies the critical components and interrelationships necessary for performing such a comprehensive analysis.

WHITE PAPER

CONCEPTUAL FRAMEWORK FOR EXAMINING THE ROLE OF STANDARDS IN THE MCCR ACQUISITION PROCESS

Mission Critical Computer Resources (MCCR)* are an increasingly important element in the development and deployment of major weapons systems. Effective management of the acquisition of MCCR is vital to maintaining a successful national defense, especially given the complexity of computer resources and technology and the diverse parties involved in the process. Currently, MCCR represents almost 3 percent of DoD's total budget.** Standards can play an important role in the process of acquiring MCCR by reducing uncertainty in the MCCR acquisition process. To formulate effective, efficient policy, DoD decisionmakers must have both a clear understanding of the general effects of standards in the MCCR acquisition process and the ability to analyze the potential impacts of a particular standard.

This paper represents the first step in developing a comprehensive analytical tool for DoD decisionmakers for evaluating the factors and forces affecting the use of standards in the acquisition of MCCR. The purpose of the tool is to provide decisionmakers with an interactive decision process using real-time

---

* An appendix includes a glossary of terms included in this paper.

** DoD, Embedded Computer Resources Standards Standardization Program Plan, Draft (Washington, D.C.: DMSSO, September 15, 1984).

information to determine how standards may be used in the MCCR acquisition process. This White Paper presents an initial framework identifying and describing the factors influencing the use of standards in the acquisition of MCCR. It develops and explains key relationships among these factors which form the basis for identifying likely impacts of standards. An overall analysis of a standard can be made by examining each of these key relationships and correlating the interplay of these factors.

Two methodologies integrate the overall analysis; these are risk/return and consumer surplus analyses. Both methods seek to measure the impacts of a standard on each of the components and provide a way to determine the overall impact. These methods are complementary and can be used together to reach a final estimate. By incorporating these methodologies,* the framework consists of both a "road map" that assists decisionmakers in analyzing the potential effects of a standard and also contains the information necessary to rationalize a given policy decision regarding standards.

Prior analyses have concentrated on impacts at one or another of the components.** One unique feature of our approach

---

* These methodologies will be refined in greater detail as more data become available.

** Prior studies examining the impacts of standards generally have focused on the impacts to one group, such as a single department in the government, overlooking other impacts on other groups or on other factors such as technology insertion. See, for example, Roger Schave et al., "Potential Effects of Standardization on Avionics Software Life-Cycle Cost," IEEE Computer (1979); Harold Stone, "Life Cycle Cost Analysis of ISA Standardization for Military Computer Systems," IEEE Computer (April 1979); Logistics Management Institute, Costs and Benefits and Federal Automated Data Processing Standards: Guidelines for Analysis and Preliminary Estimating Techniques, Washington, D.C., August 1978; and U.S. Department of Commerce, National Bureau of Standards, A Cost-Benefit Analysis of Proposed Federal Input/Output Channel Level Computer Interface Standards (Washington, D.C.: NBS, June 1978).

is that we are integrating the analysis by examining all the components and the interrelationships of these factors within a single framework. In this way all factors impinging on the process can be analyzed.

For example, assume that DoD decisionmakers are faced with determining the likely effects of implementing a standard military computer family that would permit only certain types of computers for use by DoD. The conceptual framework can be used to identify the critical factors necessary to evaluate the significant potential impacts of the standard. For example, at what stage in the technology life cycle are the particular computer systems that are being standardized? Will industry resist the standard because the technology is still in a nascent, growing phase or because the standard limits competition? How does the standard affect various DoD goals such as interoperability, technology insertion, or cost minimization? At what phase of the acquisition process will the standard be critical and have its intended effect? What are the costs and benefits, in terms of risk reduction to DoD, from implementing the standard? These questions and answers to them are vital if DoD decisionmakers are to understand adequately the effects of a standard and if DoD is to provide a suitable rationale for implementing specific standards. The objective of this framework is to assist DoD in conducting that analysis.

The conceptual framework will be refined and revised with further study and analysis, including the development of a database and empirical study. A database will be developed that will serve as an adjunct to the analytical tool to be used by high level DoD decisionmakers to (1) formulate opinions, (2) rationalize and support decisions, and (3) answer "what if" questions concerning MCCR standards. Drawing from the literature, existing databases on standards, and from case studies analogous to

potential future standards issues, the database will store factual information about standards issues including general descriptive data, risks, benefits, costs, and footnotes to the literature. The case studies will use historical analogies to current issues facing DoD with respect to the use of standards in the MCCR acquisition process.* The results of the historical studies will be used to evaluate and refine the conceptual framework as well as provide DoD decisionmakers with information that will be useful in solving current standards policy issues. In this way, decisionmakers will understand how to apply the conceptual framework and will gain insights to solve issues of current and future concern.

Before describing the framework, several assumptions must be made. For instance, although the framework describes various components in a static, single-time dimension, it is recognized that there is dynamism among the components in which impacts are spread over time. It is assumed that standards have certain functions that are important within the acquisition process and as such are demanded by certain groups, e.g., DoD and industry. Also, we assume that both DoD and industry formulate and implement strategies vis-a-vis the MCCR acquisition process. We make certain assumptions regarding strategy and the various objectives which lead to the formulation of strategy. Finally, we recognize that other factors, such as actions by foreign governments and industries, may have some influence, but for clarity and ease of understanding, we have chosen not to describe those elements.

---

* For example, such case studies might include examining the implementation of FASP as an analogy to ALS; the development of the AN/YUK 7 as an analogy to the MCF; and the use of MIL-STD-1679A as an analogy to MIL-STD-SDS (software development).

COMPONENTS OF THE CONCEPTUAL FRAMEWORK

The conceptual framework consists of several components, including the MCCR acquisition process, standards and the standardization process, DoD, industry, and government policy. These components are linked in the manner shown in Exhibit 1. The dotted lines from DoD and industry to government policy indicate that although these linkages exist, and may be very important, they are outside of the scope of this study, and therefore will not be considered here. The linkages comprise the key relationships among the dimensions and represent focal points for understanding the impacts of standards, including particular costs and benefits.

The environment of technology affects each of the components and the linkages, and each component can exert an influence on technology. Several concepts within the technology "process" are important, such as the life cycle and the learning curve. These concepts will be described before the other components of the conceptual framework to provide the proper background and focus.

The Environment of Technology

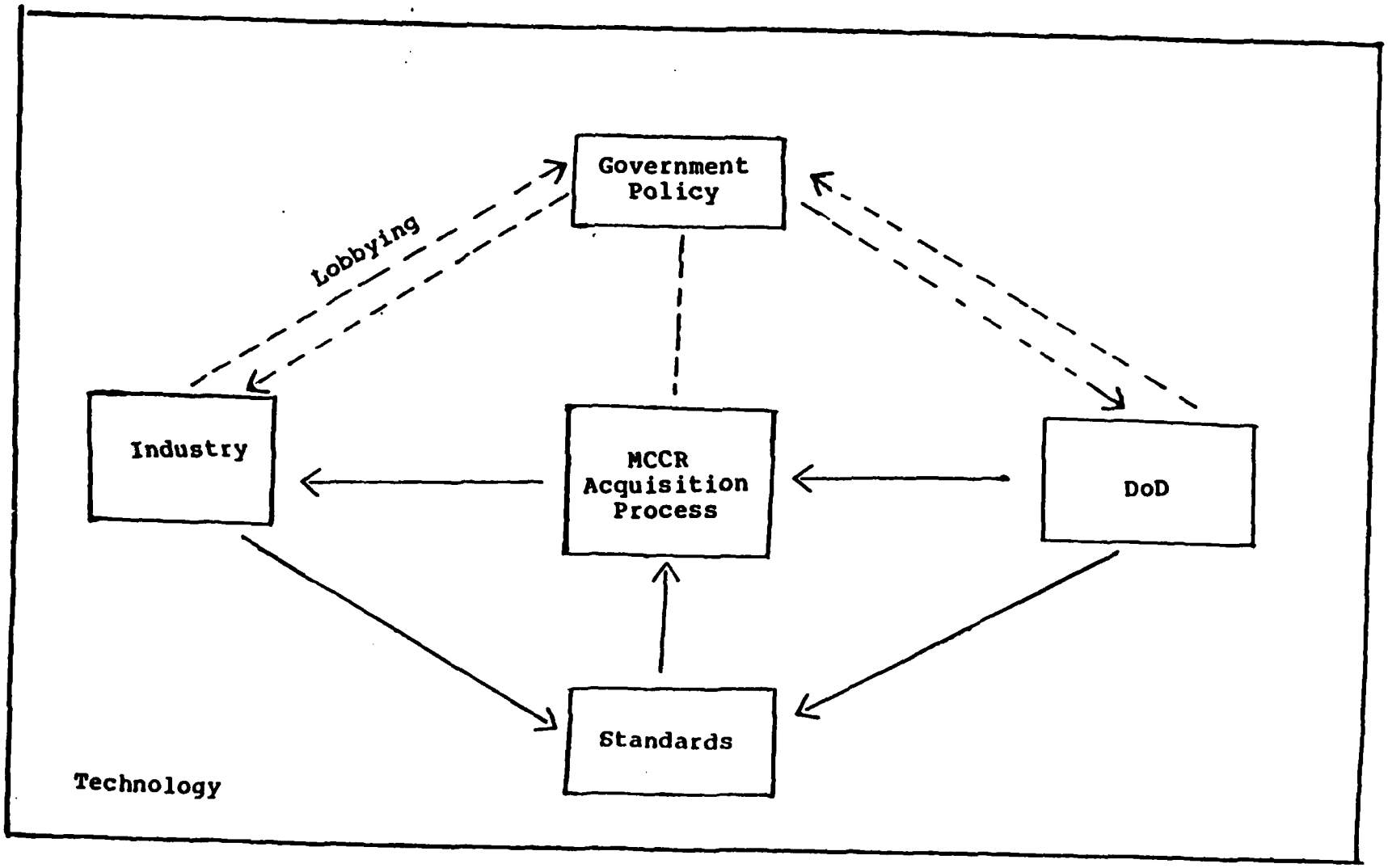
Three general concepts influence the technology process and the technology environment: (1) technology "push" versus "pull"; (2) the technology life cycle; and (3) the learning (experience) curve.

Technology "Push" Versus "Pull"

In many industries, a technology evolves and is adopted because users (consumers) demand certain types of products; this

Exhibit 1

CONCEPTUAL FRAMEWORK AND LINKAGES



has been characterized as technology "demand-pull." In this situation, the market pulls the technology and innovation results from identifying user needs and responding to those needs.* Under other circumstances, ongoing research activities or advancements in the state of science may result in an innovation, but no market exists for it. The developer, therefore, must "push" the technology by either creating a new market or demonstrating how the new technology can replace an existing one. DoD and NASA, for example, have actively pushed certain technologies before it was certain a commercial market existed.** Frequently, these efforts do result in "spin-offs" that can be marketed commercially.

Technology push, i.e., compelling or creating a market to accept a new product is achieved by reducing risk.+ Barriers to

---

* See, for example, Eric Von Hippel, "The Dominant Role of Users in the Scientific Instrument Innovation Process," Research Policy (1976); David Ford and Chris Ryan, "Taking Technology to Market," Harvard Business Review (March/April 1981); and Geoffrey Kiel, "Technology and Marketing: The Magic Mix?" Business Horizons (May-June 1984).

** Some of these studies of government technology push include Norman Asher and Leland Strom, The Role of the DoD in the Development of Integrated Circuits, P-1271 (Alexandria, Va.: Institute for Defense Analyses, May 1977); Herbert Kleiman, "A Case Study of Innovation," Business Horizons (Winter 1966); James Utterback and Albert Murray, The Influence of Defense Procurement and Sponsorship of Research and Development on the Development of the Civilian Electronics Industry (Cambridge, Mass.: Massachusetts Institute of Technology, Center for Policy Alternatives, June 1977); William Abernathy and Balaji Chakravathy, "Government Intervention and Innovation in Industry," Sloan Management Review (Spring 1979); G. P. Dinnean and F. C. Frick, "Electronics and National Defense: A Case Study," Science, March 18, 1977; and Robert Wilson, Peter Ashton, and Thomas Egan, Innovation, Competition, and Government Policy in the Semiconductor Industry (Lexington, Mass.: D. C. Heath, Lexington Books, 1980).

\+ In this manner, one can think of technology push as being somewhat analogous to technology insertion.

innovation and adoption of innovation stem from two types of risk: market risk and technical risk. Market risk is the risk inherent in attempting to sell a product and the uncertainty of that product generating a profit. Technical risk involves the risk of successfully developing a product or cost-saving process. With technology push, the primary barrier is market risk, i.e., making either producers (suppliers) and/or users understand that a particular technology is worth implementing or adopting.

In order to push a technology, a group, such as DoD, must reduce market risk. It does this by encouraging the use of the new technology through procurement, through dissemination of information about the technology, through "gatekeepers," or through funding R&D projects directed in this area.* It creates incentives to make it profitable to adopt and use this technology. The rate at which an innovation is adopted is influenced by several factors.

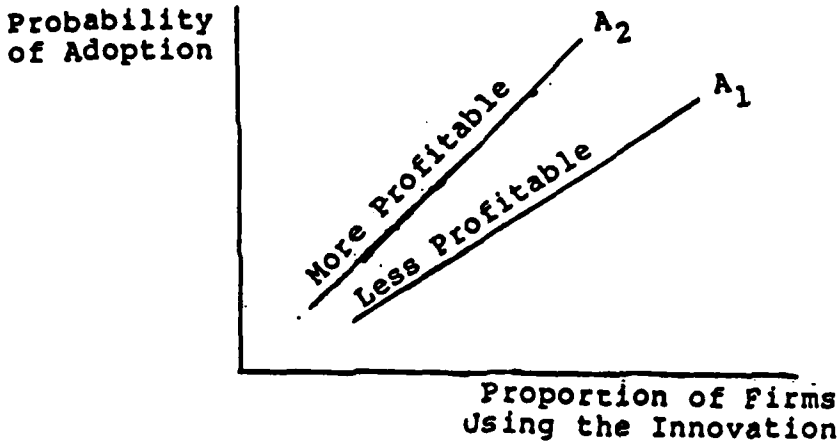
The profitability of the innovation, the number of firms using the innovation, and the amount invested in the innovation affect its rate of adoption. In Panel A of Exhibit 2, for example, A_1 and A_2, which designate two different innovations, show a direct relationship between the probability of adoption and the proportion of firms already using the innovation (degree

---

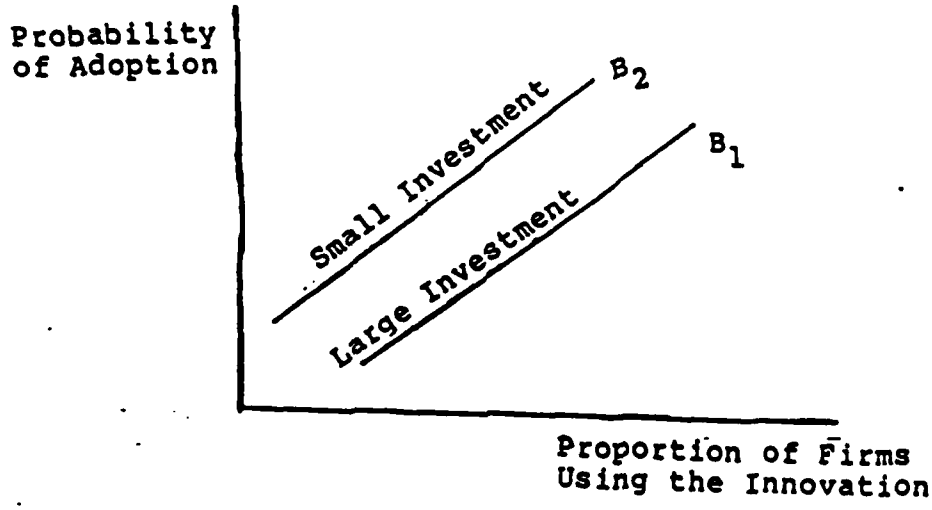
* The concept of "gatekeepers" is a particularly useful one in encouraging adoption of a new technology. Gatekeepers are members of an organization who, by virtue of their personality, expertise or other factors, control and disseminate information about innovations and are viewed as "experts" in judging the usefulness of a new technology. Gatekeepers know the appropriate people to whom the information should go to encourage adoption. Thus gatekeepers can play a vital role in the adoption of a new technology or ensure that it is never adopted. For more on this concept, see T. J. Allen, Managing the Flow of Technology (Cambridge, Mass.: The MIT Press, 1977); and Rosabeth Moss Kanter, Innovations for Productivity in the American Market: The Change Masters (New York: Simon & Schuster, 1983).

Exhibit 2

PANEL A: PROBABILITY OF ADOPTION
AS A FUNCTION OF SATURATION AND PROFITS



PANEL B: PROBABILITY OF ADOPTION
AS A FUNCTION OF SATURATION AND SIZE OF INVESTMENT



of saturation).* Panel A also shows that the probability of adoption is higher at every level of saturation for A_2 than for A_1, because A_2 is a more profitable innovation. Panel B of Exhibit 2 shows a higher probability of adoption of an innovation requiring a smaller investment (B_2) than one requiring a larger investment (B_1), other things being equal.

On a priori grounds, these are relationships that one would expect. The risks of introducing a new technology generally diminish as market saturation grows, therefore, one would expect increased adoption as experience and information increase. Likewise, the more profitable the investment in a new technology, the greater the compensation for anticipated risks of undertaking that investment. Finally, it seems reasonable to expect firms to be more reluctant to commit large amounts (particularly when they have difficulty raising large capital amounts) than small amounts to the development of new technologies.**

Technology Life Cycle+

A considerable research effort has been dedicated to examining the development and growth of a technology. This research

---

* Edwin Mansfield, The Economics of Technological Change (New York: Norton & Co., 1968).

** This process of "push" frequently involves a revolutionary idea or concept as opposed to an evolutionary process; once the technology is used in one or a few applications, "demand-pull" overtakes technology push, leading to the use of the technology in many other areas. One example is the integrated circuit; it was first developed and used by DoD; once industry learned its capabilities, the applications for integrated circuits widened rapidly.

\+ This theory was first developed by Raymond Vernon, "International Investment and International Trade in the Product Cycle," Quarterly Journal of Economics (May 1966); and L. T. Wells, The Product Life Cycle and International Trade (Cambridge, Mass.: Harvard University Press, 1972).

has identified distinct phases in the life cycle of a product and has related them to the phases of the acquisition process as well as the development of standards.

Panel A of Exhibit 3 displays the various stages of the technology life cycle. Industry growth typically follows an S-shaped curve; the flat introductory phase reflects the difficulty of overcoming buyer inertia and technology push. During the growth phase, a market is created and many buyers demand the product or technology as it demonstrates its usefulness. The technology eventually becomes widely accepted and growth levels off and finally declines as new substitute technologies appear.

The concept of the life cycle has been extended to evaluate the pattern of product and process innovation and when particular standards are likely to be developed.* Panel B of Exhibit 3 indicates the patterns of product and process innovation. This exhibit shows that, over time product innovation slows, particularly after a "dominant design" has been adopted; process innovation increases as product innovation declines, and greater emphasis is placed on cost reduction and efficient, large-scale production.

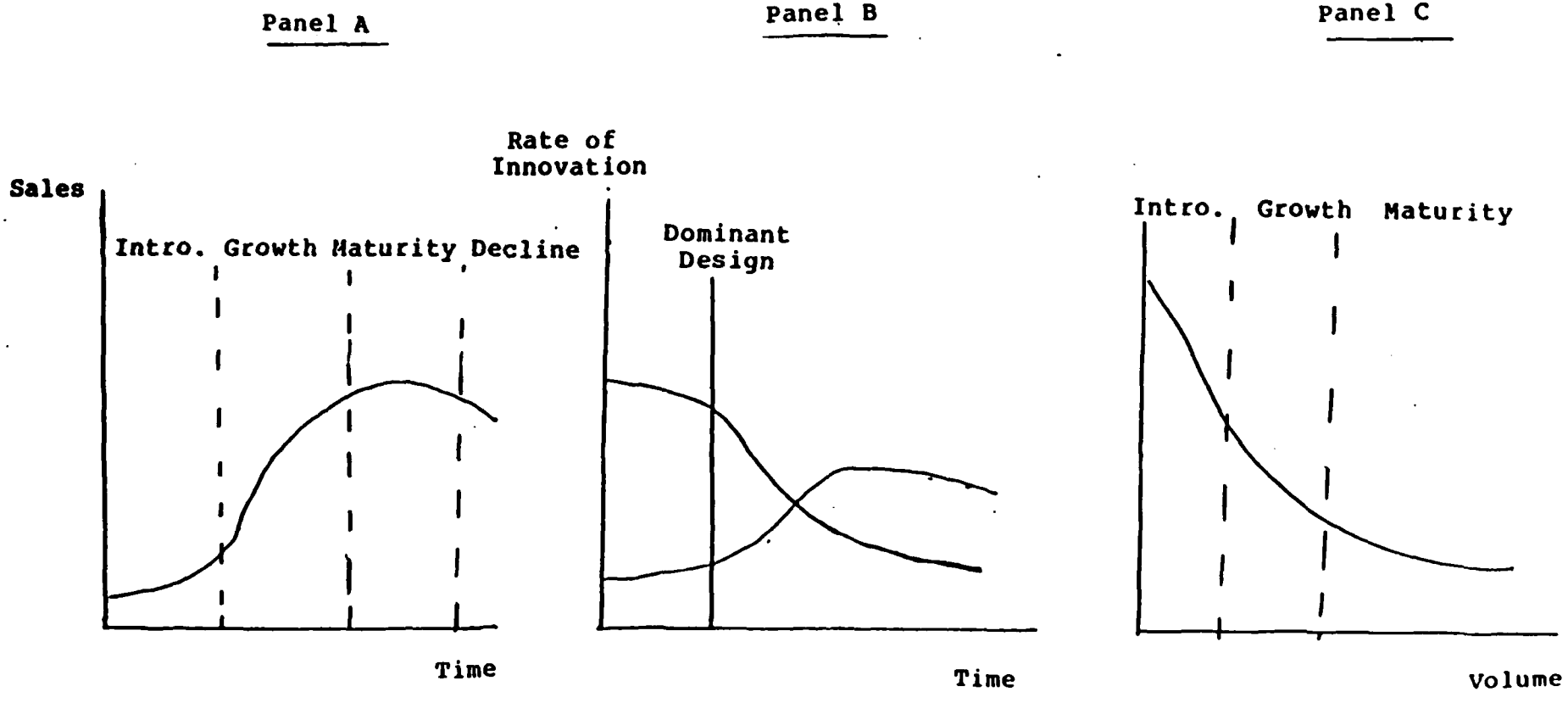
One hypothesis regarding the life cycle is that as the life cycle or technology matures, standardization becomes more important. In early stages, there is radical product innovation, wide variation in product design and methods of production. Most standards are unimportant and not desired at this stage. As the

---

* W. J. Abernathy, The Productivity Dilemma (Baltimore, Md.: Johns Hopkins University Press, 1978); J. M. Utterback and W. J. Abernathy, "A Dynamic Model of Process and Product Innovation," OMEGA (1975); and David L. Bodde, "Technical Standardization, Competition, and Innovation," Draft Working Paper, 1975.

Exhibit 3

TECHNOLOGY PROCESS CONCEPTS



product enters the early mature stage, standardization begins, first with a dominant design and then with cost competition. In the late mature phase, markets are well-defined, cost competition is critical, and standardization becomes very important as a means of reducing cost.

The Learning (Experience) Curve

The learning curve builds on life-cycle theory. The learning curve predicts that as a company (or industry) produces more (especially in labor-intensive industries), its unit costs will decrease because it learns how to produce more efficiently with production experience.* These cost declines are usually more significant in the early and growth phases of the product life cycle. Panel C of Exhibit 3 plots the learning curve against the phases of the technology life cycle, indicating this trend.

The implications of learning curve theory are quite simple. If costs decline with experience in an industry and if that experience is kept proprietary, the established technology leader will have an inherent cost advantage over latecomers and can pursue aggressive pricing strategies in anticipation of future cost declines.** In areas with little or no competition, the learning curve may be considerably "flatter," because there is less pressure to reduce costs.

---

* A particularly good description of the concept of the learning curve is contained in the Defense Contract Audit Manual, Appendix F, "Improvement Curve Analysis Techniques," May 1979.

** Some of these advantages have been termed "first mover" advantages and may involve not only a cost advantage but an advantage derived from brand recognition, advertising, or buyer loyalty. See Oliver Williamson, Markets and Hierarchies (New York: The Free Press, 1978).

Reference to the learning curve in the acquisition of MCCR may be important to determine who the low-cost suppliers are and how they can apply their experience to DoD's needs. Certain companies develop their strategies based on the learning curve concept; for example, in anticipation of future cost reductions, they may lower prices before current costs would dictate such a reduction. This can affect corporate strategy vis-a-vis the MCCR acquisition process and influence the point in time when standards may be desired. Those companies who are first to market a technology may desire product and process standards as a means of reducing costs and reducing the risk of investing in that technology. DoD must also be aware of where in the technology life cycle a product is, because this will influence its ability to push a technology and gain industry acceptance of standards. For instance, it may be very difficult for DoD to enforce a product standard if no dominant design has yet emerged.

Texas Instruments (TI), a significant supplier of integrated circuits to DoD, built much of its early growth on the learning curve concept. TI was an early producer of integrated circuits which became a dominant design. Part of TI's strategy focused on selling devices to DoD which provided an assured market and reduced the risks of rapid capacity expansion necessary to take advantage of the learning curve. Given the sharp declines in cost with increased output in this industry (integrated circuit costs have declined on average by 28 percent with each doubling of quantity produced),* this strategy was crucial to TI's success as a leading producer of integrated circuits. The timing was critical for DoD. DoD was pushing this technology and because of the position of the technology in the life cycle, it was easy to get industry adoption, as well as realize substantial cost savings with some standardization on a dominant design.

---

* Wilson, Ashton, and Egan, op. cit.

The MCCR Acquisition Process

MCCR is defined as computer systems "applications involving: (1) intelligence systems; (2) cryptography for national defense; (3) command and control of military forces; (4) weapons or weapons systems; (5) direct support to military or intelligence operations."*

The development of any major system is determined largely by the operational requirements. The requirements for the development and/or acquisition of computer resources result from the operational requirements. The MCCR acquisition process is part of the overall process by which DoD acquires major weapons systems and it is governed by a number of directives.** OMB Circular A-109 describes the distinct phases of the life cycle of the acquisition process for major systems.+ Because of the importance of technology evolution in the acquisition process, it is useful to understand how DoD acquires computer resources within the life-cycle context. The phases as defined by A-109 include:++

---

* DoD, Report to Congress on Computer Technology (Washington, D.C.: OUSDRE, August 1983), p. ii.

** For example, see DoD Directive 5000.1, "Major System Acquisitions," March 29, 1982; DoD Directive 5000.2, "Major System Acquisition Procedures," March 8, 1983; and DoD Directive 5000.29, "Management of Computer Resources in Major Defense Systems," April 26, 1976.

\+ Office of Management and Budget (OMB), Major Systems Acquisitions, OMB Circular A-109 (Washington, D.C.: OMB, 1976).

++ Greater detail on these phases and the management of the acquisition process is contained in Defense System Management College, "Management of Software Acquisition," Fort Belvoir, Virginia, January 1984.

- Mission analysis -- analysis of existing systems, capabilities, priorities, and opportunities and identification of a particular need, threat, or deficiency in existing systems.

- Exploration of alternative systems -- analysis of alternative ways to meet the perceived need including initial solicitation, proposals, and evaluations.

- Competitive demonstrations -- analysis of alternative system design concepts to verify the chosen concepts and provide a basis for selecting a system for full-scale development.

- Full-scale development, test, and evaluation -- selection of contractor(s) including monitoring and evaluation of progress to full-scale development to assure effective performance under expected conditions.

- Production/performance appraisal -- full-scale production of the desired system and continued monitoring of contractor performance.

- Deployment and operation -- actual field deployment and use of the system.

- Maintenance and support -- logistics and maintenance support of the system during deployment.

- Retirement and disposal -- replacement of system with new system, including disposal and/or use with non-active forces.

The life cycle of the weapon system may be as long as 25 to 35 years. Yet the typical life cycle of a computer technology generation is about 5 to 10 years. This disparity creates a unique problem. A major weapons system deployed in 1985 with a life expectancy of 25 years nevertheless could be outmoded or less effective than expected within 10 years due to development of new, more efficient computer technology.

DoD is concerned with ensuring timely technology insertion and utilizing state-of-the-art technology in all computer systems embedded in major weapons systems to reduce the risk of using less effective or less efficient technology. Technology insertion must be planned early on in the life cycle and continually monitored and updated. For example, by the demonstration phase, a system's projected life-cycle costs, including pre-planned product improvements (P^3I), must be within the amounts reflected in the latest FYDP (Five Year Defense Plan). Yet even the process of technology insertion (P^3I) can cause lengthy delays and cost overruns. It is this unique problem of managing concurrent yet very different life cycles within the acquisition process that is of continual concern to DoD program managers.

Standards and the Standardization Process

A standard's function(s), type, proponents, and the groups it affects influence the effect it will have on the MCCR acquisition process. For the purpose of this study, standards are defined as documents which establish engineering and technical limitations and applications for items, materials, processes, methods, designs, and engineering practices.* The formal standardization process as defined by DoD is divided into three steps: (1) standards are developed and agreed upon; (2) those standards are then communicated to users; and (3) standards are applied in a cost-effective manner.** The bulk of the standards (and specifications) evaluated in this study are mandatory, MIL

---

* Rowen Glie, Speaking of Standards (Boston, Mass.: Cahners Publishing Company, 1972).

** DoD, Overview of the Defense Standardization and Specification Program (Washington, D.C.: OUSDRE, 1983).

or FED standards, although some are developed within the voluntary consensus process or in a de facto manner.* Another set of standards are international (ISO) standards. International standards represent a higher level of standardization in terms of requiring broader agreement of many groups. Nevertheless, they can be important to DoD especially in relation to supporting the NATO forces.

Functions of Standards

The analysis of standards proceeds from understanding the various functions they perform, including information; compatibility/transportability/interoperability; variety reduction (optimizing flexibility); and quality.** Functions determine the impacts of a standard.

The information function defines terms and establishes measurement and test methods; these standards generate and disseminate information to buyers, sellers, and other users of the standard. An example of a standard that contains this function is a test method for evaluating the functional properties of integrated circuits.

---

* De facto standards evolve in a market from one or a small group of companies who, because of market dominance, establish standards without necessarily using the voluntary consensus process.

** This delineation of the functions of standards was based on a broad survey of the literature on standards and was first formally defined in Putnam, Hayes & Bartlett, Inc., The Impacts of Private Voluntary Standards on Industrial Innovation, prepared for National Bureau of Standards, Washington, D.C., November 1982.

The compatibility function ensures that two related products will fit with one another. In specialized uses such as software, special categories of compatibility such as interoperability and transportability are important. Interoperability refers to "the ability of two systems to exchange data and understand the relationships between these data objects."* Transportability is the ability of software to be installed on different environments and perform with the same functionality.** Software standards such as Ada provide compatibility.

The variety reduction function minimizes proliferation and attempts to achieve the optimum variety of a particular product. This function permits producers to cut costs by making longer production runs and to reduce consumer (buyer) search costs. One example would be a single standard instruction set architecture (ISA) as proposed under 5000.5X, which would have reduced the different varieties of ISAs bought by DoD.

The quality/reliability function establishes minimum levels at which a product must perform to be acceptable. This function involves a determination of better or worse. Such standards are extremely common within DoD, since most system acquisitions must meet well-specified performance criteria.

Standards may have multiple functions and in fact the functions of standards often operate in tandem; for example, test methods provide information needed to evaluate quality. The information function may also be essential to determine how to establish compatibility or reduce variety. Consider a standard

---

* DoD, Report to Congress on Computer Technology, p. 14.

** Council of Defense and Space Industry Associations (CODSIA), DoD Management of Mission-Critical Computer Resources, Washington, D.C., March 1984, Volume II, p. 113.

that establishes environmental specifications under which computer hardware must operate. The standard may stipulate the temperature range, humidity, altitude, and other conditions under which the computer must function. This standard conveys information from buyers to suppliers about certain performance characteristics of the computer and also establishes a level of quality that the computer must meet to be acceptable to the buyer.

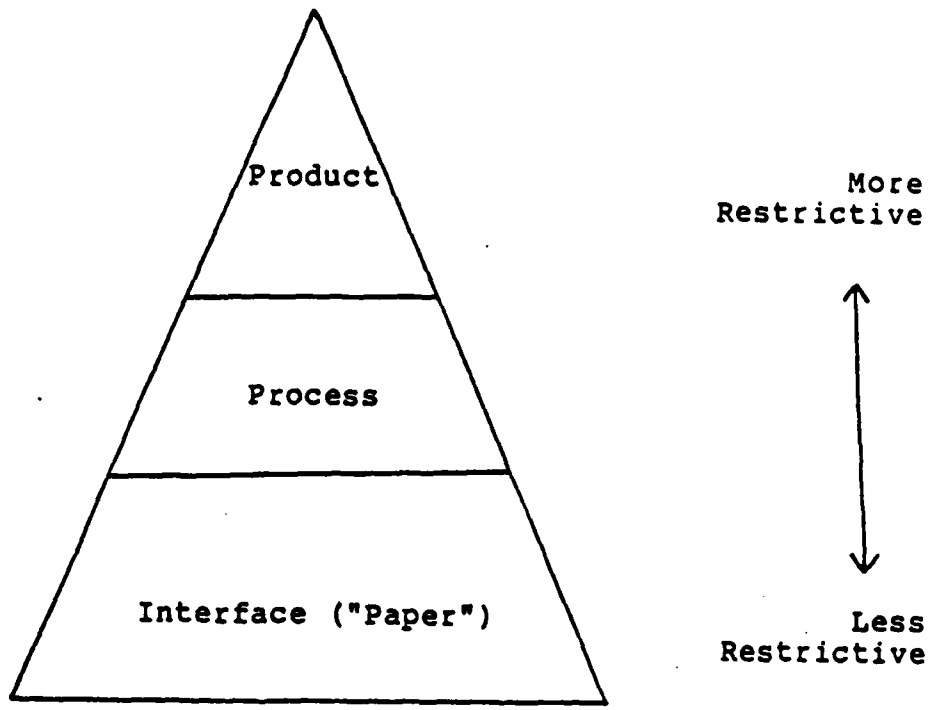
Types of Standards

In addition to being categorized by function, a standard can be categorized by its type. Does the standard cover a particular product; or does it involve a process; or does it simply establish an interface? This categorization scheme is essentially a hierarchy as shown in Exhibit 4; interface or "paper" standards are less restrictive and merely define certain characteristics. For example, an interface standard may define the communications protocol to be used in connecting two "boxes." Process standards, while somewhat more restrictive, define the manner in which a process is be done, but do not determine the actual output. Process standards include such standards as those governing software design, development, and documentation. Product standards do define a particular output or outcome and therefore tend to be the most restrictive. MIL STD 1750A is a product standard because it defines a particular commodity, i.e., a 16-bit instruction set architecture (ISA).

These categories indicate the likely effects of a particular standard (or the function of a standard) both in terms of whether certain groups may resist the standard and in terms of how well the standard will be integrated into the MCCR acquisition process and the technology life cycle. For instance, a product standard that is promulgated too early in the technology life cycle may be

Exhibit 4

TYPES OF STANDARDS



resisted by suppliers (industry) because it does not allow enough flexibility in adapting to anticipated changes in the technology. On the other hand, a simple interface standard may be insufficient for users or maintainers of a product who require accurate and detailed information about a particular product and how it functions. Such information may not be critical at early stages of the acquisition process, but may become essential at later stages.

Proponents of Standards

Depending on which of the various groups that is the actual proponent of the standard, the standard itself will be perceived differently and will have different effects. There are essentially four sets of standards developed by different combinations of groups: government mandatory standards, government-industry (joint) standards, industry voluntary standards, and industry de facto standards.

Government mandatory standards are standards that must be adhered to by everyone doing business with the government. Most DoD and MIL standards, federal regulations, and the like are considered mandatory standards. Usually they are published by the government and compliance is considered mandatory. Enforcement is overseen by the appropriate government agency.

Joint government-industry standards may be developed through the voluntary consensus process, but adoption and use by the government makes the standard mandatory. The primary difference between this standard and a government mandatory standard is that industry has had input into the process and is less likely to disagree with the standard.

Industry voluntary standards are developed through the consensus process;* there is general industry agreement but no one is compelled to obey the standard. The impacts of these standards may be different because no one is obliged to follow the standard; a standard may be implemented, yet have no impact because no one complies with it.

Industry de facto standards are standards that evolve from one or a small group of corporations which often dominate a segment of an industry. These firms have enough market power simply to establish the standard, and market forces dictate that everyone else must follow that standard. For example, IBM has been responsible for setting many de facto standards in the computer industry. These standards may have far-reaching effects or none at all depending on whether competitors are willing to follow the standard and whether buyers believe it is a beneficial standard.

Groups Affected by Standards

For the standards described above, there are different groups who at a given time may desire a particular standard. Standards as a whole are desired because they benefit the economy or some particular process in our economy.** The "demanders" of

---

* For a detailed discussion of the voluntary consensus process of developing and implementing standards, see American Society for Testing and Materials, Standardization Process (Philadelphia, Pa.: ASTM, 1980) and ASTM, The Standards System of the U.S. (Philadelphia, Pa.: ASTM, 1975).

** For a detailed discussion of the factors influencing the demand for standards, see D. Bottaro, "Analysis of Factors Affecting the Demand for and Supply of Voluntary Consensus Standards," Massachusetts Institute of Technology Energy Lab Working Paper, 82-003WP, August 1981.

standards fall into one (or more) of several groups: buyers, suppliers (sellers), users, and maintainers. Buyers are those who buy the product affected by the standard, in this case, DoD or one of the Services. Suppliers are those who sell the product in question, usually industry. Users are those who actually use the product, usually DoD or the Services in this case. Maintainers are those who must maintain and support the product and ensure its effective and continued operation after deployment.

Each of these groups may desire particular standards (or functions of standards) to assist them in performing their jobs. Buyers desire standards that reduce costs, ensure adequate product performance, or permit accurate comparisons. Suppliers want standards that clearly define the buyers' (and/or the users') needs and ensure that the product will be sold. Users demand standards that facilitate the operation of the product, for example, by assuring compatibility or performance. Maintainers want standards that enhance logistical capabilities, such as reducing variety or ensuring a certain level of reliability. Also, because maintainers typically did not develop the system, they are interested in standards that will assist them in maintaining something they did not develop, i.e., software documentation.

Each group may also desire different types (functions) of standards at different points in time. Buyers and sellers are interested in standards earlier in the acquisition life cycle than users or maintainers. However, to ensure effective performance during deployment, users and maintainers may need to be involved early in the acquisition process to assure that their needs are met.

Department of Defense

In acquiring MCCR, DoD formulates and implements certain strategies based on its goals and requirements. These requirements compel DoD to acquire the best computer resources utilizing the latest technology. However, this process must be viewed in the unique context of the life cycle of a weapons system, which is usually in excess of 30 years. Given the rapid changes in computer technology during this life cycle, DoD has a unique problem of ensuring the use of up-to-date computer resources in systems that have a very long life cycle. The enormous financial resources which DoD commits to MCCR further compound this problem. DoD spent over $5 billion for MCCR in 1981; this amount is projected to increase to $38 billion by 1990.* As stated in a recent draft report,

> ... [I]t is important that computer resources in Defense systems be managed as elements or subsystems of major importance during the various life cycle phases, with particular emphasis on computer software and its integration with the surrounding hardware.**

Exhibit 5 shows the factors influencing DoD's development and implementation of strategy. DoD constantly reviews the threats against which it must operate. The overriding objective is to minimize the threat in the most efficient, productive manner. DoD formulates specific strategies to meet its goals; these goals in turn are based on perceived needs and requirements. DoD in its Report to Congress on Computer Technology identified various requirements for MCCR. These requirements, listed in Exhibit 6, are divided into two categories: those derived from operational needs and those derived from logistical

---

* Electronic Industries Association, DoD Digital Data Processing Study -- A Ten Year Forecast, October 7, 1980.

** DoD, Embedded Computer Resources Standards Standardization Document Program Plan, p. 3.

Exhibit 5

FACTORS INFLUENCING THE DEVELOPMENT OF DOD STRATEGY
VIS-A-VIS MCCR ACQUISITION PROCESS

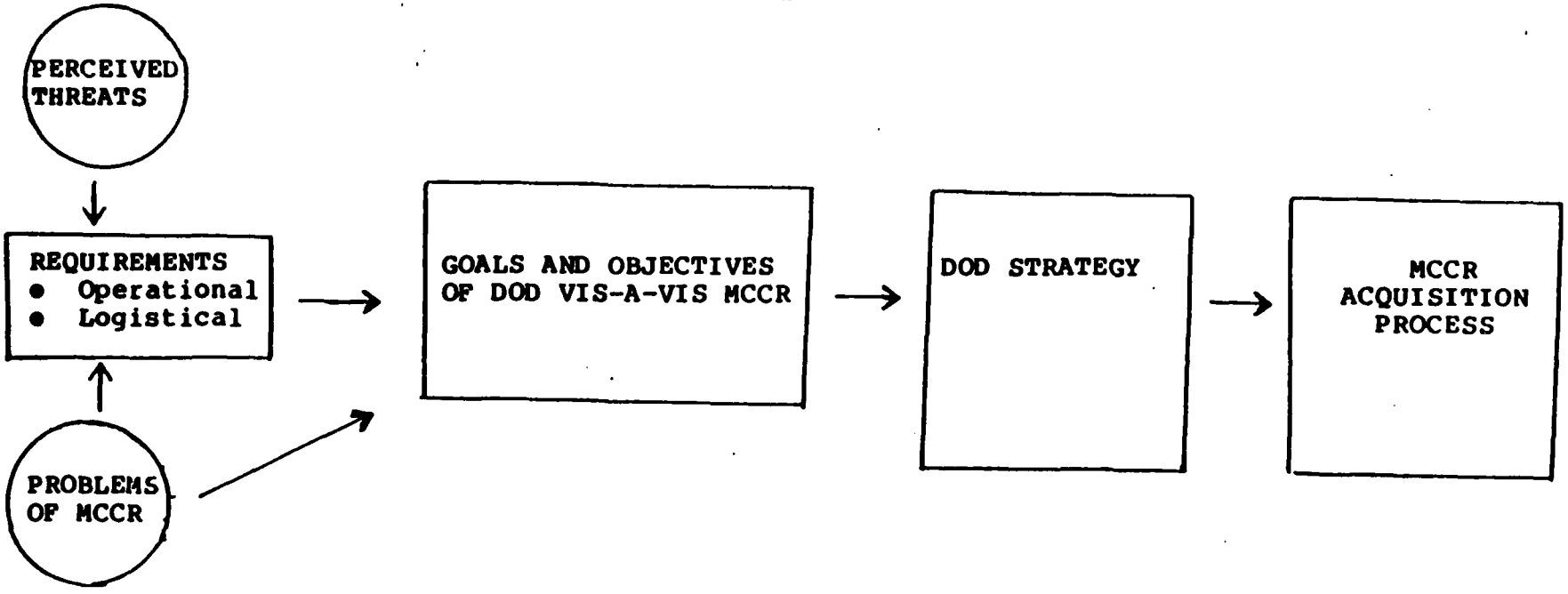


Exhibit 6

DOD REQUIREMENTS FOR MCCR*

Operational Requirements

- Enclosures
- Survivability and vulnerability
- Interoperability
- Interchangeability
- Reliability
- Maintainability
- Hardening
- Vibration, acceleration and shock
- Thermal
- Power, size, and weight

Maintenance and Support Requirements

- Specific operational requirements
- Maintainability
- Manpower and training
- Sparing and repair

---

* Adapted from DoD, Report to Congress on Computer Technology (Washington, D.C.: DoD, August 1983).

(or maintenance) needs. Because operational needs are the overriding concern to DoD, they determine, in part, the logistical needs.

In addition, DoD identifies specific problems or issues inherent in MCCR which must be addressed by its strategy. These problems include:*

- Lack of adequate competition;
- System schedule slip;
- System failures;
- Latent defects;
- Inability to reuse and transport software;
- High cost of maintenance support and logistics;
- Low operational availability;
- Difficulty in maintaining and upgrading software as needs change and as technology changes.

The unique requirements of DoD as well as the problem of timing between the technology life cycle and the system life cycle cause these problems and are an important input into the development of objectives determining DoD's strategy in acquiring MCCR.

DoD's objectives in acquiring MCCR can be deduced from an understanding of its requirements and problems. While not necessarily exhaustive, a list of objectives provided below

---

* Defense Science Board, Task Force on ECR Acquisition and Management (Washington, D.C.: OUSDRE, November 1982); DoD, Report to Congress on Computer Technology, pp. ii-iii; also see CODSIA, DoD Management of Mission-Critical Computer Resources, Volume II, pp. 21-22.

indicates the factors driving DoD's strategy in acquiring MCCR. These objectives include:

- Reducing the cost and time of acquiring MCCR;
- Improving operational readiness;
- Enhancing interchangeability, interoperability, and transportability of MCCR;
- Improving maintenance and logistics support of MCCR;
- Reducing the variety proliferation of MCCR; and
- Enhancing the ability to introduce preplanned product improvements (technology insertion) to MCCR.

DoD develops strategies to acquire MCCR that will meet these objectives. For every new start of a major system, an initial acquisition strategy is proposed and implemented. Part of this strategy will include consideration of technology insertion (P^3I). Further, DoD strategy must be developed in sufficient detail at the time of issuing solicitations for concept exploration to permit competitive exploration of alternative system design concepts.

DoD strategy changes for different systems. For one system, DoD may opt for the lowest cost system even though its performance may not be the best offered, but it is sufficient to meet DoD's needs. In other situations, system performance (quality/reliability) is of greatest concern and cost is a less important factor in DoD strategy. As one understands DoD strategy and how it affects the MCCR acquisition process, one can begin to evaluate how standards can be used in that process and can identify potential impacts.

Industry

Because industry supplies MCCR to DoD via the acquisition process, the behavior of companies in industries such as computers and semiconductors must be examined. Exhibit 7 displays the factors influencing industry behavior and the development of strategies by companies within an industry.* The behavior of these companies is conditioned by the environment in which they operate; for example, standards and the MCCR acquisition process are elements of the environment and affect the corporate behavior of these companies. The actions of competitors as well as government policy also affect industry behavior. In addition, other factors, including the organization of the companies and their goals, influence behavior.

The organization represents the manner in which a company structures its human, capital, and financial resources. Companies in industries also pursue goals; these goals may be short-run, such as maximizing market share, or long-run, such as profit maximization. Given a set of goals and a particular organizational structure, a company will formulate and implement a set of strategies by matching these capabilities with the opportunities that it perceives in the marketplace.

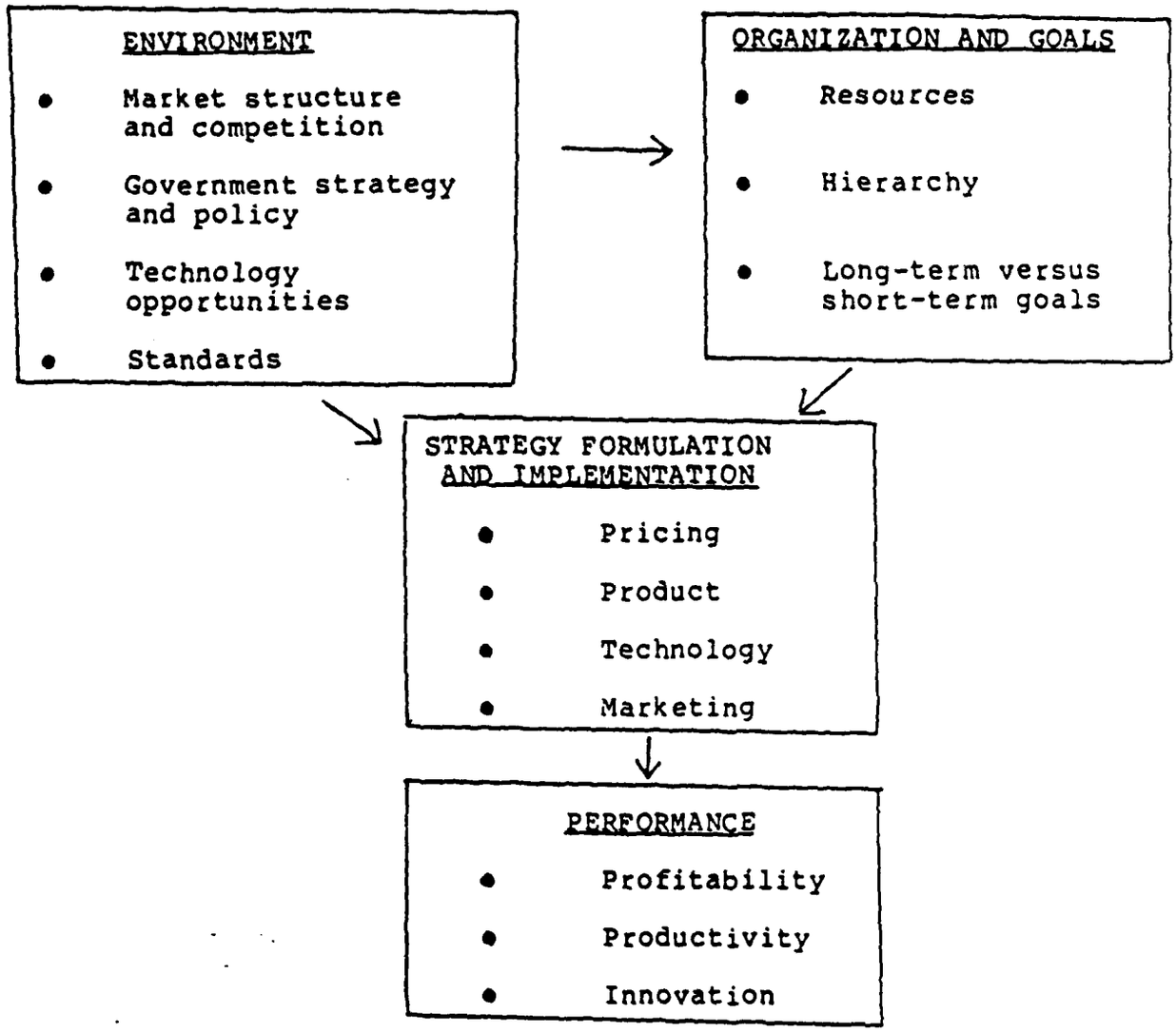
A company selects various strategies with respect to pricing, product choice, technology, and marketing. The company implements a set of strategies in hopes of achieving its goals and attaining a certain level of performance. For example, certain companies in the semiconductor industry have followed a set of strategies that emphasize introducing new technologies, producing high-quality/high-priced devices in relatively low volume, and concentrating on specialized markets or market

---

* This industry model has been developed in Putnam, Hayes & Bartlett, Inc., op. cit.

Exhibit 7

FACTORS INFLUENCING CORPORATE STRATEGY



niches. These firms perform reasonably well as long as competition is not severe and they are able to maintain a technological lead. Successful strategy implementation depends upon both environmental factors (market structure, degree of competition) and organizational factors (good in-house researchers, flexible management of the technology development process).*

LINKAGES IN THE FRAMEWORK

The components of the conceptual framework provide the building blocks for isolating the effects of standards in the MCCR acquisition process. The next step is to identify the key linkages among the components which explain the factors influencing the impacts of standards. These linkages are shown as solid lines in Exhibit 1.

DoD -- MCCR Acquisition Process

DoD strategy and behavior are linked closely to the MCCR acquisition process since DoD is the primary motivating force behind that process. In Exhibit 5, we noted that DoD implements a strategy which is directly related to the MCCR acquisition process. That strategy encompasses various factors designed to affect the MCCR acquisition process at various stages to obtain the desired results. To be effective, DoD's strategy may differ at various stages of the acquisition process. For example, even though P^3I may not be introduced until late in the life cycle, it must be planned for and budgeted in the early phases of the life

---

* See Peter K. Ashton and James A. Dalton, "Strategic Behavior and Performance in the Semiconductor Industry," Texas Business Review (Spring 1983).

cycle. Changes in technology, however, can affect DoD's plans with regard to technology insertion. DoD's ability to match an effective strategy with each stage of the acquisition process is a key to achieving its objectives.

The beneficial and/or harmful impacts of a particular strategy are isolated by determining the impact on three concepts:

- Budget planning;
- Cost predictability;
- Uncertainty (risk) reduction.

For example, DoD's MCCR acquisition strategy must facilitate the budget planning process for the acquisition life cycle. This strategy should also induce cost predictability both in terms of overrun control and in terms of properly making risk/return assessments. DoD's strategy should also reduce risk, where risk is defined as the uncertainty associated with the timely developing and deploying of a cost-effective, reliable MCCR.

The uncertainty or risk of the MCCR acquisition process is a function of several factors including cost, quality, time, and system flexibility. By changing each of these factors, risk (uncertainty) can be raised or lowered. Each of these factors is in turn influenced by a number of variables impinging on the MCCR acquisition process.

Cost is defined as the total acquisition cost for a particular MCCR. It is influenced by the existing technology base, the extent to which suppliers can move down the learning curve, particular DoD requirements for that MCCR, and the amount of competition in supplying that MCCR.

Quality reflects the probability that a MCCR will perform as anticipated. Several elements affect the quality of MCCR including the existing technology base, particular DoD requirements, and tradeoffs which the supplier(s) or DoD makes between cost and development time.

Time is defined as the length of time needed to develop and deploy a MCCR. It, too, is defined by the existing technology base and DoD requirements as well as the shape of the learning curve and the acquisition life cycle itself.

System flexibility reflects the capability of MCCR to be interoperable, transportable, and compatible with other MCCR. The existing technology base and DoD requirements affect flexibility as does the existence (or non-existence) of standards.

DoD takes each of these four factors into account in developing a strategy; one objective to minimize uncertainty. In order to minimize uncertainty, DoD must maximize system flexibility and quality while also minimizing cost and development time. Since each of these factors is interrelated and affected by similar elements, DoD must make tradeoffs among them to obtain an optimal condition with minimum uncertainty. For instance, to increase quality, DoD must make greater use of time or money (cost) or both.

The following example illustrates the tradeoffs between cost, time, quality, and system flexibility of MCCR that face DoD. Assume that DoD chooses to encourage competition and the number of competitors throughout the MCCR acquisition process. DoD may benefit because this reduces uncertainty by lowering cost and by indirectly improving quality through maintaining many alternative sources. With a greater number of competitors, movement down the learning curve may be slower, but greater emphasis may be placed on process (cost) innovation because of the diversity of approaches. On the other hand, uncertainty may increase

because delivery time is delayed due to an extended life cycle for solicitation and evaluation of different proposals. The strategy affects other factors such as budget planning which is made more complex and less certain. Each of these impacts are weighed to determine the overall impact of DoD strategy on the MCCR acquisition process and then compared with other impacts flowing from the other linkages.

DoD -- Standards

DoD and the Services formulate and implement various types of standards and specifications as part of their strategy for acquiring MCCR. Standards represent one focus of DoD strategy vis-a-vis the MCCR acquisition process. Its impact on the MCCR acquisition process is the primary focus of this study and is treated in detail as a separate linkage. In this section, we point to the reasons why DoD uses standards to acquire MCCR and discuss the likely results.

DoD has explicitly identified certain objectives with regard to its use of standards. These objectives are to "assure physical and operational interchangeability and interoperability while balancing specific mission requirements with technological growth and cost effectiveness."* DoD employs standards where they will reduce the risks inherent in the MCCR acquisition process. The previous section showed that standards can affect system flexibility which in turn influences risk. Standards can also indirectly influence other elements such as cost, time, and quality. The primary benefit to DoD of using standards properly is that standards improve DoD's ability to make tradeoffs among quality, cost, time, and flexibility, thereby making it easier to achieve

---

* DoD, Overview of the Defense Standardization and Specification Program, p. 4.

an optimal solution to reducing risk. Costs result when standards elevate risk by exacerbating one of these elements.

For example, standards that define the manner in which software will be developed and documented provide a general framework for software acquisition and allow common implementation by developers, users, and maintainers.* The benefits of the standard to DoD include information that is transferred among users and enhancement of interoperability; the standard may also reduce costs and development time by providing a uniform method of developing and documenting software. Insertion of new technology may be easier because software developers have to spend less time "reinventing the wheel." They can concentrate on new programming because the design and documentation needs are clearly specified.

DoD implements such standards where such benefits clearly exist; what is less clear is a standard that may generate certain benefits, but also increases costs in other ways. Suppose the documentation standard described above were written in such a way so as to prevent using certain new programming techniques which in turn delayed technology insertion. In this case although the standard may have provided useful information and reduced costs, it also prevented the use of advanced programming techniques and caused DoD to acquire software which was not state-of-the-art. Measuring these lost opportunities may be difficult, but DoD frequently faces these tradeoffs which can condition its behavior leading to suboptimal choices.

---

* Such as MIL-STD-1679A, "Software Development."

Industry -- Standards*

Standards affect industry in two ways: (1) by influencing strategic behavior, and (2) by altering their investment behavior. Strategic behavior is affected, for example, if a standard establishes a minimum quality level that necessitates pushing the "state of the art," it may force a company to pursue a product or technology strategy designed to achieve this quality level. This is one way in which DoD can insert new technologies. As another example, DoD may implement a standard that reduces the proliferation of different varieties of applications software. Companies that had supplied software not covered in the standard then face three strategic choices: (1) exit from the market of supplying to DoD; (2) develop new software that meets the standard; or (3) seek a waiver from DoD to permit it to use the non-standard software. These options may be more or less risky depending on commercial applications for the company's "nonconforming" software, the nature of competition in the DoD marketplace, and the company's overall strategic outlook vis-a-vis that particular product.

Standards influence industry behavior in another way. Standards can alter the expected profitability of an investment project by changing the risk/return payout of the project. The expected profitability of any investment project is a function of the expected returns and the relative riskiness of those returns. (See Equation 1.)

$$E(\pi) = \sum_{I}^{N} P_1 V_1 + P_2 V_2 \ldots + P_n V_n \qquad (1)$$

---

* This linkage is described in detail in Putnam, Hayes & Bartlett, Inc., op. cit.

where

$E(\Pi)$ = Expected profitability

P_i = The probability (risk) of the expected return for i^{th} product

V_i = Expected return for i^{th} product

The functions of standards can alter V_i, that is, the risk associated with earning a particular return. Standards may both raise and lower, market and technical risks.*

Test and measurement methods provide valuable information which permits the comparison of results of various innovations. If a researcher knows that one prototype will measure favorably against other test products, R&D effort can concentrate on the design that is superior, thereby reducing development costs (and technical risks) and increasing expected profitability. In these situations where risk is lowered and/or returns go up, and expected profits increase, standards will be desired; standards will not be demanded by industry when they raise risk (or lower returns) and lower expected profitability.

For example, DoD may implement a standard that provides information to <u>all</u> suppliers of software that DoD will purchase only software that operates on certain compilers. This reduces the risks of developing certain software since suppliers are aware of the precise specifications of what DoD wants and how the software must operate. Certain companies that do <u>not</u> have compatible or transportable software may face higher risks of

---

* As discussed above at page 8, market risk represents the risk associated with attempting to sell a product and the probability that the product will generate a profit. Technical risk involves the risks of successfully developing a product or cost-saving process.

doing business with DoD, especially if there are established competitors already making compatible software.

The impacts of standards to industry result from their influence on industry behavior and through impacts on risk. Standards can raise or lower risk, and not all companies may be affected in the same manner. The nature of the impact depends upon the competitive position and strategies of companies prior to the imposition of the standard.

Industry -- MCCR Acquisition Process*

The MCCR acquisition process has a direct impact on industry and industry strategy.** This process represents a manifestation of DoD strategy and its impact on industry. It includes the interaction of the MCCR acquisition process and standards, which is the crucial linkage discussed in the next section.

As we discussed above, the MCCR acquisition process may be viewed as an element of the environment which conditions and constrains industry strategy (see Exhibit 7). The MCCR acquisition process may influence a company's goals either by creating new goals or reinforcing existing goals. The structure and resource base of the company may also be affected by the acquisition process; for instance the company may need to acquire and/or train personnel to deal particularly with DoD and respond to DoD's unique requirements.

---

* A more detailed discussion of this linkage appears in Wilson, Ashton, Egan, op. cit.

** Industry can influence the MCCR acquisition process indirectly in three ways: (1) through lobbying efforts that influence the acquisition process directly; (2) through lobbying efforts that feed back on DoD strategy; and (3) through industry participation in implementing standards used in the MCCR acquisition process.

Since companies pursue different strategies, the MCCR acquisition process will affect various companies differently. Certain firms may aggressively pursue strategies aimed at winning DoD business while others are less aggressive and concentrate greater effort in commercial markets. Changes in the way DoD affects the MCCR acquisition process can alter corporate strategy. For example, greater emphasis on cost control or less emphasis on reliability may compel certain companies that previously had emphasized high cost/high quality products to reorient their strategy.

The analysis of industry reaction to one MCCR acquisition program will demonstrate how industry is affected. The VHSIC (very high speed integrated circuits) program was announced in late 1978 as a six-year $200 million program to push technology in the development of integrated circuits. Many saw this as a response to the Japanese and European government efforts in the VLSI (very large scale integration) area. Industry initially greeted the VHSIC program with mixed feelings. Some pointed to its likely effects on companies' organization and resources; this program was expected to be engineer-intensive and, given the scarcity of trained design and process engineers, a potential problem of committing enough talent to the program was identified. Others claimed that the program would divert resources away from promising commercial areas to military applications with little commercial relevance. Still others believed that the VHSIC program would not insert enough basic technology and not "push" the technology in the appropriate direction.

Many of these complaints have proven ill-founded and most companies have welcomed this government interest and support of integrated circuit technology. These companies believe VHSIC is providing needed capital resources and will help stimulate continued growth and innovation in this area. Some companies

have responded to the program, reorienting their strategies in hopes of "cashing in" on the capital for R&D provided by DoD. It is still too early to tell what the final impacts of the program will be; it is clear such a program can have significant effects on industry and its strategies and indicates the factors evaluated by industry in making impact analyses of such programs.

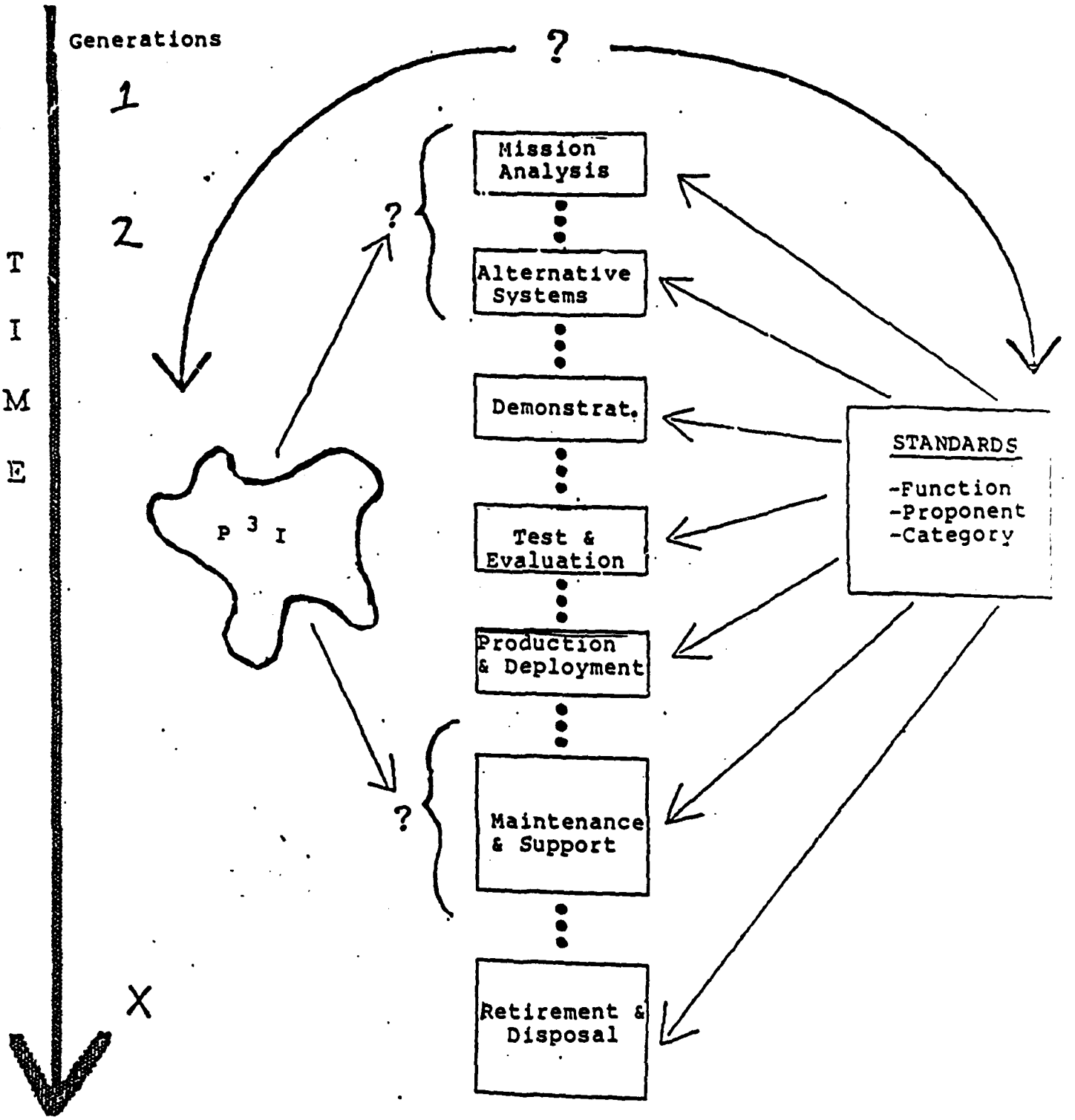
Standards -- MCCR Acquisition Process

The linkage between standards and the MCCR acquisition process is the crucial relationship in the framework because it demonstrates the impacts of standards. We have seen that DoD pursues certain goals in the MCCR acquisition process and that the use of standards is one element of DoD's strategy for achieving its objectives by reducing risk in the MCCR acquisition process. Further, we have seen how industry and standards interrelate and how the MCCR acquisition process can affect industry. But to fully understand whether DoD's use of standards will achieve various objectives and what the impacts are, the standards-MCCR acquisition process linkage must be analyzed.

Exhibit 8 depicts a schema showing that standards interrelate with all phases of the MCCR acquisition process. The functions of standards impact each of the phases of the MCCR acquisition process, although certain functions are more important at some phases than others. In the early phases of the MCCR acquisition process, basic information is needed to help define needs and assist contractors in understanding DoD's requirements. During the demonstration, and test and evaluation phases, the quality function is important because it gives DoD program managers a means to measure performance and make comparisons among competing systems and against overall system requirements. Later on at the maintenance and support phase, the variety

Exhibit 8

STANDARDS - MCCR ACQUISITION
PROCESS LINKAGE



optimization and compatibility functions are important for adequate spare parts availability and to facilitate repair by means of compatible modular construction.

In addition, the flexibility of standards (i.e., interface, process, or product) will change with the different phases of the MCCR acquisition process and the technology life cycle. In the early phases of the acquisition cycle and the life cycle, standards will be written so as to maximize flexibility and encourage alternative designs. At later stages, standards may be more process- or product-oriented, emphasizing physical dimensions and designs.

The proponents of standards will be involved in different phases of the acquisition process. Clearly buyers and suppliers will be most actively involved in the phases up through production and deployment; at this point users and maintainers assume significant roles and will demand standards that facilitate use and maintenance of the system. The buyers and suppliers, however, do not drop out of the picture completely; if a system fails to operate properly in the field, feedback to the buyer and supplier from the user will compel them to alleviate the problem.

This linkage, as Exhibit 8 indicates, however, must be viewed in the context of the technology process and particularly the technology life cycle and technology push/insertion or P^3I. A critical issue facing DoD is <u>when</u> in the acquisition process P^3I is desired (because of the technology life cycle, DoD knows P^3I must occur) and <u>how</u> does DoD get industry to help provide P^3I? As noted earlier, DoD must plan for P^3I early in the life cycle. This may involve interaction with potential users as well as giving consideration to what the state of the art may be 10 or 20 years hence.

Standards can foster or inhibit the insertion of technology in the MCCR acquisition process. Early adherence to a quality standard (such as an ISA) that freezes on a particular technology or design clearly hampers technology insertion and can increase DoD's risks by deploying less effective systems. On the other hand, an information and compatibility standard such as for software documentation may facilitate technology insertion by making it easier to introduce new advanced software to an existing system whose requirements are clearly understood. In the latter case, risks are reduced because system flexibility has improved.

To assess the impact of standards in the MCCR acquisition process, one must examine whether the goals of DoD strategy in that process have benefited (or been achieved) as a result of the use of the standard. As discussed previously, the benefits of standards are derived from the manner in which they reduce risk. For example, as noted above, the quality function of a standard obviously helps assure that DoD obtains reliable MCCR. Compatibility standards can reduce costs and increase system flexibility. Standards that make it easier for DoD to trade off among the factors affecting uncertainty are perhaps the most beneficial. For instance, a standard that establishes common software terminology and definitions among the Services helps to reduce system development time by making it easier to determine mission needs and evaluate current systems. Such a standard may also enhance system flexibility by facilitating comparisons across the Services and thereby also reduce the cost of developing new software that can be utilized by all three branches of the military.

Aside from looking at the impacts on DoD, one must analyze the reaction by industry to a particular standard. DoD may believe that standards for software documentation will facilitate technology insertion as described above. However, if the standard provides a method for documentation with which industry is

unfamiliar or believes is not appropriate, industry may resist the standard and technology insertion may not proceed smoothly.

This last example demonstrates the necessity of evaluating all the linkages and impacts of a standard to fully determine its impact. Both DoD and industry analyze the likely effects of a standard and, depending on the perspective, the impact analysis may prove radically different. The last section of this paper will discuss techniques for conducting the impact analysis.

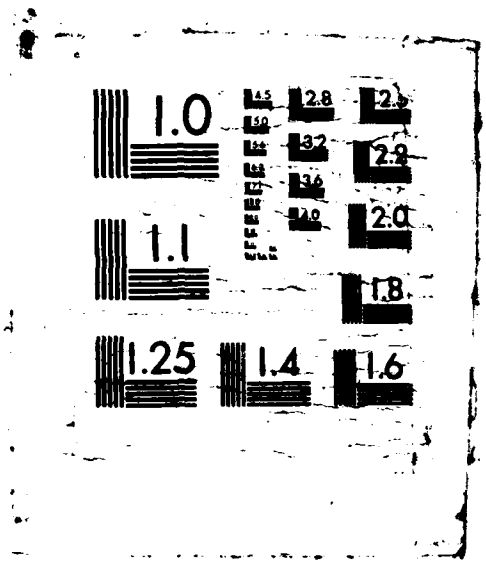
METHODOLOGIES FOR EVALUATING THE IMPACTS OF STANDARDS

Once one understands the components and their linkages in the framework, one can assess the impacts of standards or proposed standards and attempt to make an overall evaluation. As we noted earlier, prior analyses have examined the impact of standards on one or two dimensions; however, to gain a complete understanding of a standard one must evaluate the impacts at each component level and conduct an analysis integrating those impacts across the framework.

There are a number of ways to integrate the impacts in the framework. Common techniques include risk-benefit analysis and consumer welfare analysis. Each technique as developed herein merely provides a "road map" for decisionmakers and must be refined and developed further on the basis of further analysis and case study. Furthermore, these techniques may be used together to complement and supplement each other in arriving at an overall determination of impact.

AD-A175 352 COST EFFECTIVENESS TRADEOFFS IN COMPUTER STANDARDIZATION AND TECHNOLOGY I..(U) INSTITUTE FOR DEFENSE ANALYSES ALEXANDRIA VA A A HOOK ET AL. JUN 86 4/4

UNCLASSIFIED IDA-P-1931 IDA/HQ-86-31052 MDA903-84-C-0031 F/G 9/2 NL


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0

Consumer Welfare Analysis

Consumer and producer welfare analysis examines the benefits (and costs) of changing the demand and supply for a particular good, in this case, MCCR. The concept of consumer surplus provides a tool for measuring changes in social welfare. Consumer surplus is the difference between the maximum price that consumers are willing to pay for a product (such as an MCCR) and the actual price of that product. Producer surplus represents the profits earned by a producer in selling a product.

Consumer surplus can be measured as the area under the demand curve and above the horizontal line denoting price, as shown by the shaded area in Exhibit 9. If the price of the product falls, the price line moves down and the area showing consumer surplus increases. Producer surplus is commonly measured as the difference between revenue earned by the producer and the cost of making and selling the product.

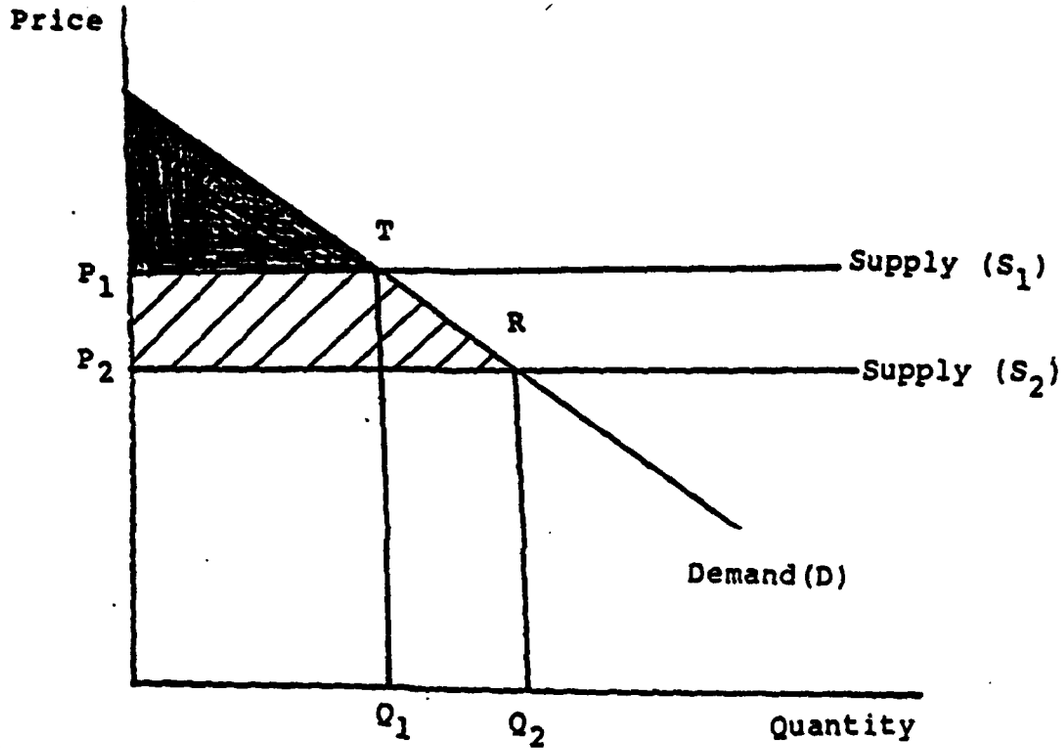
With the introduction of a standard, the supply curve (S_1 in Exhibit 9) is assumed to shift down to S_2 as a result of lower production costs.* The price of the standardized product falls from P_1 to P_2 and the quantity demanded increases from Q_1 to Q_2. The measure of benefits from introducing the standard is the increase in consumer surplus resulting from the decrease in price paid by the consumer, i.e., DoD, as shown by the cross-hatched area, P_2RTP_1. Depending on this shift in the producer's cost curves, producer surplus may either increase, decrease, or remain the same. Comparison of price-cost margins before and after the imposition of the standard provide the measure of change in producer surplus.

---

* It is assumed that the standard lowers production costs, for example, by optimizing variety and permitting the realization of scale economies.

Exhibit 9

CONSUMER SURPLUS ANALYSIS



In order to fully apply this technique, one must first be able to estimate the demand and supply for MCCR for a particular system and then estimate the life-cycle costs (and cost savings) of introducing a standard. Furthermore, as the framework demonstrates, these estimates must incorporate not only direct costs and savings, but indirect effects as well on not only buyers and suppliers, but users and maintainers as well. Clearly, sufficient data must exist before this method is feasible; however, it does provide a useful and relatively simple approach to the problem.

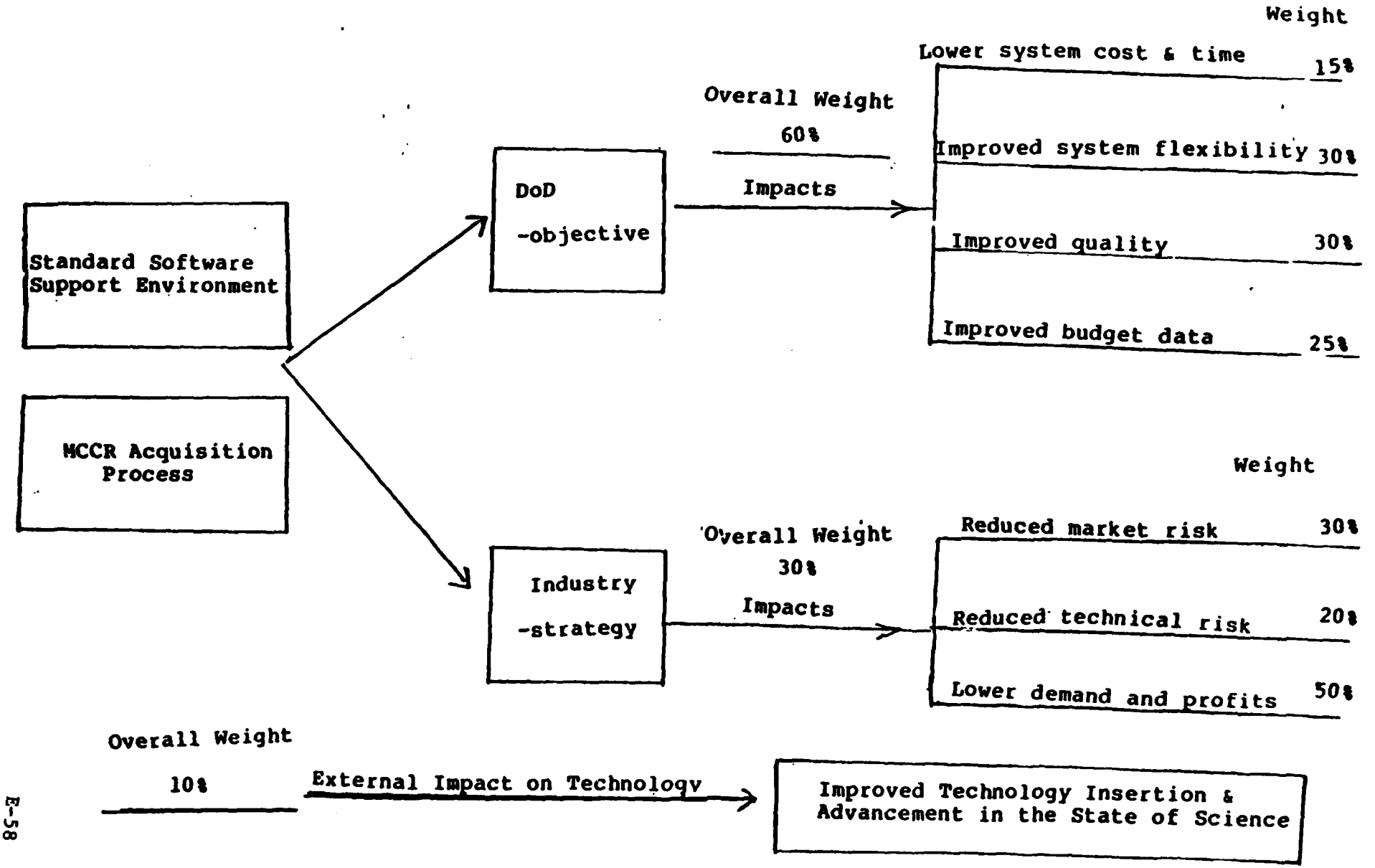
Risk-Benefit Analysis

Traditional cost-benefit analysis involves measuring the impacts of a standard in terms of the costs and benefits it generates. Measurement of costs and benefits in quantitative terms is not necessarily simple or straightforward. The analysis discussed here involves "risk-benefit" analysis which is less quantitative but provides a way to assess all the impacts (direct and indirect) of a standard. One evaluates impacts in terms of whether risk is increased or decreased and what the benefits are that result from changes in risk. A series of decision rules are implemented to determine the nature of the impact on risk and weights are established to assist in comparing impacts among the components in the framework. A hypothetical example of risk-benefit analysis follows and is depicted in Exhibit 10.

Assume DoD issues a set of standards designed to implement a standard software support environment that could operate with a variety of computer systems. DoD's objectives in this effort would be to reduce applications software proliferation, provide central configuration control, and improve software support from both development and life-cycle standpoints. In examining the

Exhibit 10

IMPACT ANALYSIS INCLUDING WEIGHTING OF RISK-RETURNS
FROM A SOFTWARE ENVIRONMENT STANDARD



likely impacts of the standards from DoD's point of view, the benefits derived from reduced risk would include: (1) lower cost and less time required to establish software support facilities; (2) greater total (hardware/software) system flexibility and easier upgrading; (3) improved system quality; and (4) better information about life-cycle cost.

The cost savings are measured by comparing the cost of independent development for software support per system plus the life-cycle support costs per system with the costs of implementing a standard software support environment per system. This quantitative estimate reflects only one set of benefits to DoD and must be weighted relative to the other benefits to DoD. Decision rules can be developed, for example, that state that improved quality and system flexibility are twice as important as dollar cost savings and thus these benefits would be weighted twice as heavily as the cost savings. (As shown in Exhibit 10, flexibility and quality are weighted twice as much as cost/time.)

The analysis must also evaluate the impact from industry's perspective. Industry may react favorably to the standard as it reduces both market and technical risk of developing software support tools and facilitates upgrading to new systems and enhancing existing systems. Again these benefits must be weighted according to a set of decision rules. Hypothetical weights for both DoD and industry are shown in Exhibit 10 as well as an overall weighting scheme. Industry may experience certain costs as well. Clearly, a standard software environment reduces software proliferation which may potentially reduce software sales and profits. Also, proprietary support software would no longer be necessary which might deter some companies from competing. The technology insertion process, however, would be improved by the standards, particularly as it would facilitate side-by-side hardware and software development. Summing up all

the impacts and weighting by the scheme shown in Exhibit 10 yields an overall impact estimate.

Again, a more rigorous approach to weighing and comparing the benefits and costs of the standard must be developed; nevertheless, this analysis provides a way to integrate the impacts of a standard and understand how the use of a particular standard affects the MCCR acquisition process and ultimately impacts both industry and DoD.

CONCLUSION

This study will develop a decisionmaking model to assist high level DoD officials in determining whether a proposed standard or proposed application of a standard affecting the MCCR acquisition process will have the anticipated or desired effect. The approach taken provides a methodology for examining the problem from the perspectives of the MCCR acquisition process, DoD objectives and goals, industry objectives and goals, and the standardization process. By developing a model which examines the proposed action from all these perspectives and the linkages among them, important secondary and tertiary effects which may actually exceed the primary or anticipated effect in scope and significance may be identified.

The model will guide the DoD official through a series of critical questions and issues that need to be examined from the perspective of the specific situation. In addition to focussing his attention on these critical areas, the model will provide him with an analysis of the probable outcome based on surveys of the relevant literature and actual case studies. This analysis will provide appropriate citations and references which can be used directly by the official in any supporting documentation he needs to produce.

Appendix A

GLOSSARY*

Ada™. An ANSI standard programming language trademarked by and under the control of the U.S. Department of Defense, with a number of modern characteristics.

Availability. Operational Availability (OA) is defined as the probability that, when used with actual operating environment, a system as equipment will operate satisfactorily at any time.

Higher Order Language (HOL). A software language for programming, which is more problem oriented than the traditional machine-level assembly languages.

Instruction Set Architecture (ISA). The attributes of a computer processor as seen by a machine (assembly) language programmer, i.e., the conceptual structure and functional behavior of a computer (at the machine-language level) as distinct from the organization of data flows and control, logic design, and physical implementation.

Interoperability. The ability of two environments (computer systems, processors, or weapon systems) to exchange data objects (data fields, records, files, or messages) and their relationships in forms usable by tools and user programs without conversion. Interoperability is measured in the degree to which this change can be accomplished without conversion.

Mission Critical Computer Resources (MCCR). Computer systems applications involving: (1) intelligence systems; (2) cryptography for national defense; (3) command and control of military

---

* The definitions contained in this glossary are from Council of Defense and Space Industry Associations (CODSIA), DoD Management of Mission-Critical Computer Resources, Volume II, prepared for the Undersecretary of Defense, Research and Engineering, March 1984.

APPENDIX F

REFERENCES

Ada Information Clearinghouse, "Ada IC", July 1985.

Agrawal, S. et.al., "On Performance Oriented Design of JSSEE(JSSEE Performance Considerations)", Report EE-ARCH-012, BGS Systems, Inc., Waltham, MA 02245.

Alfeld, L., and Graham, A., *Introduction to Urban Dynamics*, Cambridge, MA: MIT Press, 1976.

Arndt, D. "The Application of Ada Generics to Large-Scale Projects: A Case Study", Bell Technical Operations Corporation Technical Report, Tucson, AZ, no date.

Association of Computing Machinery, "Matrix of Ada Language Implementations", *ACM*, updated May 1985 by COMPASS.

Barnes, J. and Fisher, G., Jr., "Ada in Use", *Proceedings of the Ada International Conference*, Paris, 14-16 May 1985.

Boehm, B., *Software Engineering Economics*, Prentice Hall, 1981.

Boehm, B., and Standish, T., "Software Technology in the 1990's: Using an Evolutionary Paradigm", *IEEE*, 1983.

Brown, M., "Computer Validation Called Beneficial to Ada Use", *Government Computer News*, October 1985.

Brown, M. "UK Re-examines Ada Implementation Policies," *Government Computer News*, October, 1985.

Brykczynski, W., "Abstracts.1" Abstracts of Ada tool packages. Draft. Institute for Defense Analyses, Alexandria, VA, June 1985.

Carlyle, R., "Panacea or Placebo?", *Datamation*, August 1985.

Clapp, J., et al., "A Cost/Benefit Analysis of Higher Order Language Standardization", The MITRE Corporation Technical Paper M78-206, McLean, VA, September 1977.

Cormier, A., and Alberts, D., WIS Joint Mission Application Software Sizing Study: Volumes III and IV," Institute for Defense Analyses Technical Paper P-1868, October 1985.

Courtwright, T., "Ada Tools Update", briefing slides prepared for the Ada Joint Program Management Office/ADT, no date.

Defense Science &Electronics, "Ada: an in-depth look", March, 1984.

Defense Science and Electronics, "Interview with William R. Hattabaugh", March 1985.

Defense Science and Electronics, "The Second Annual Ada Directory", March 1985.

Dijkstra, E., "Structured Programming," in *Software Engineering Techniques*, J.N. Burton and B. Randall, Eds., NATO Science Committee, 1969.

Druffel, L., et al., "The STARS Program: Overview and Rationale", *IEEE*, November 1983.

Fahey, J., *USAF Aircraft 1947-56*, Ships and Aircraft, Falls Church, VA 1956.

Fisher, D., "Automatic Data Processing Costs in the Defense Department," Institute for Defense Analyses Technical Paper P-1046, Alexandria, Va, October, 1974.

Flaspohler, J., et al., "The Software Test and Evaluation Project: Tools Baseline," Georgia Institute of Technology, Atlanta, GA, September, 1985.

Foreman, J., "APSE Interactive Monitor, Final Report on Interface Analysis and Software Engineering Techniques", Volumes I-III, Technical Report, Texas Instruments Equipment Group, McKinney, TX, July 1985.

Foreman, J. "Building Software Tools in Ada: Design, Reuse, Productivity, Portability", Briefing Slides, Texas Instruments Co., McKinney, TX., July 1985.

Forrester, J., *Principles of Systems*, Cambridge, MA: MIT Press, 1969.

Fox, J., "Benefits Model for High Order Language", Technical Report TR78-2-72, Defense Advanced Research Projects Agency, Arlington, VA, 1978.

GEC Software, "Overview of the GEC Software IPSE Product Strategy", The General Electric P/C of England, London, August 1985.

George, J., "DoD Computing Activities and Programs Ten Year Market Forecast Issues, 1985-1995", Technical report for Electronics Industries Association, 1985.

Goodman, M., *Study Notes in System Dynamics*, Cambridge, MA: MIT Press, 1974.

Graham, A. "Parameter Estimation in System Dynamics Modeling," *Management Science*,1980.

Griffin, W., "Software Engineering in GTE", *IEEE*, November 1984.

Holmes, E., "P-System Poops Out", *Datamation*, August 1985.

Jensen, R, "Projected Productivity Impact of Near Term Ada Use in Software System Development," Hughes Aircraft Co., Fullerton, CA. No date.

Johnson Space Center, "Level C Space Station Project Office Proposes Use of Ada for Applications Software in Flight Subsystems", Houston TX, no date.

KITIA, "Views on a CAIS from Industry and Academia", Draft, August 1985.

Klumpp, A. "Space Station Flight Software: Hall/S or Ada?", *IEEE*, March 1985.

Kramer, J., "Interview with Col. Whitaker", notes of verbal communication, Institute for Defense Analyses, Alexandria, VA, 1985.

Kruchten, P., et al., "Software Prototyping Using the SETL Programming Language", *IEEE*, October 1984.

Marmor-Squires, A., et al., "The Support Systems Task Area", *IEEE*, November 1983.

Martin, J. "The Management of Mission Critical Computer Resources", Parts I-III *Defense Science and Electronics*, February, April and May, 1985.

McDonald, C., et al., "Seeking Ada's Full Potential", *Defense Science and Electronics*, April 1985.

Myers, Edith, "Picking Up the Pieces", *Datamation*, August, 1985.

Najberg, A., and Healy, R., "The Impact of Ada on Software Development Costs," The Analytic Sciences Corporation, Report No. TR-4612-5-2, Reading, MA, October 1984.

Office of the Under Secretary of Defense, Research and Engineering, "Department of Defense Computer Technology (Study Annex): A Report to Congress," Washington, D.C., January 1984.

Oglesby, C., and Urban, J., "The Human Resources Task Area", *IEEE*, November 1983.

Osterweil, L. "Software Environment Research: Directions for the Next Five Years", April 1981.

PA. Computers and Telecommunications, "Benefits of Software Engineering Methods and Tools, " London, England, June 1985.

Richardson, G. and Pugh, A., *Introduction to System Dynamics Modeling with DYNAMO*, Cambridge, MA: MIT Press, 1981.

Richmond, B. "A User's Guide to STELLA", High Performance Systems, Inc., Hanover, NH, 1985.

Riddle, W., and Wileden, J., "Environment Extensibility Impact on the STARS SEE Architecture: SEE-ARCH-007-001.0, Technical Paper P-1828, Institute for Defenses Analyses, April, 1985.

Ripken, K., "US DoD Motivation-Engineered Language Design," TECSI, EFDPA, London, September 1982. Quoted in Rogers, M, "IT Companies' Acceptance of and Attitudes toward Ada," in Barnes, J. and Fisher, G., Eds., *Ada in Use: Proceedings of the Ada International Conference*, Paris 14-16 May 1985, The Ada Companion Series, Cambridge University Press.

Russell, D. "First Ada Compilers Show Diversity", *Defense Electronics*, March 1984.

SofTech, "ALS Description", Technical Paper, Softech, Inc., Middletown, RI.

SofTech, "Architectural Descripition of the Ada Language System (ALS), JSSEE Report No. JSSEE-ARCH-001, Middletown, RI, December 1984.

Stanley, R, "Whither Ada?", *Defense Science and Electronics*, March 1985.

Stenning, V. et al., "The Ada Environment: A Perspective", *IEEE*, June 1981.

Stephen, D, et al., "DoD Digital Data Processing Study: A Ten-Year Forecast", Technical Report for the Electronics Industries Association, 1981.

Stone, H., "Life-Cycle Cost Analysis of Instruction-Set Archetecture Standardization for Military Computer Systems", *IEEE*, APRIL 1979.

Suydam, W., "Ada Programs Emerge as Compilers Vault Validation Hurdles", *Computer Design*, June 1985.

Systems and Software, "Packages Spawn Ada Growth", 1985.

Tlustos, C., "Users Need Ada Training," *Government Computer News*, October 1985.

U.S. Department of Defense, Instruction 7041.3 "Economic Analysis and Program Evaluation for Resource Management," October 18, 1972.

U.S. Department of Defense, "Military Standard: Software Support Environment" DoD-STD-1467, January 1985.

U.S. Department of Defense, "Requirements for Ada Programming Support Environments -- 'Stoneman'," February 1980.

Verity, J., "Empowering Programmers", *Datamation*, August 1985.

Wasserman, A., "Automated Development Environments", *IEEE*, April 1981.

Williams, J. "Compiler and Tool Set for Ada Design and Implementation", *Defense Electronics*, January 1983.

Distribution List for P-1931

Sponsor

Virginia L. Castor (8 copies)
Ada Joint Program Office
The Pentagon, Room 3D139
(1211 Fern/C-107)
Washington, D.C. 20301

Other

Defense Technical Information Center (2 copies)
Cameron Station
Alexandria, VA 22314

END

2-87

DTIC