

AD-A185 985

A PLAN FOR COLLECTING ADA SOFTWARE DEVELOPMENT COST
SCHEDULE AND ENVIRONM. (U) TECOLOTE RESEARCH INC SANTA
BARBARA CA N J BRENNER ET AL. 02 APR 87 CR-0134/1

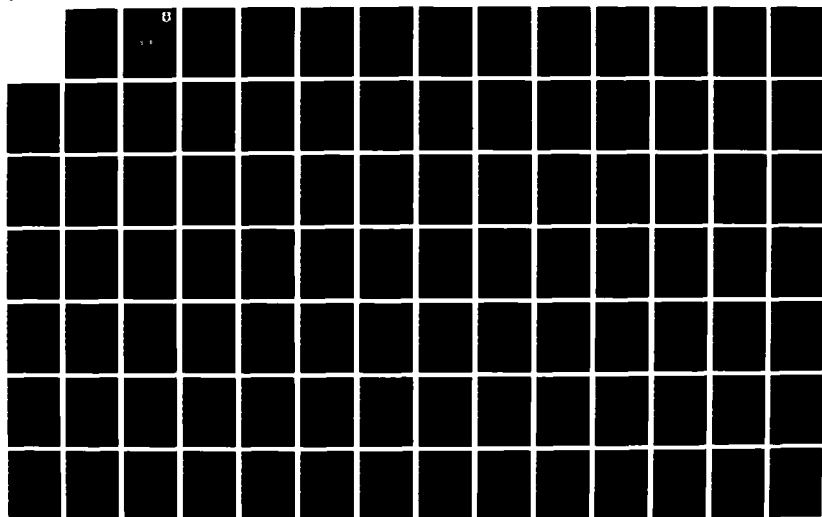
1/2

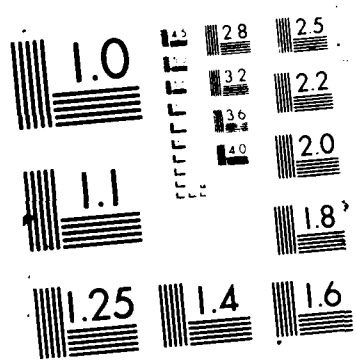
UNCLASSIFIED

ESD-TR-07-167 F19620-04-D-0019

F/G 12/5

NL





A Plan for Collecting Ada Software Development
Cost, Schedule, and Environment Data

DTIC FILE COPY

NEAL J. BRENNER
DANIEL D. GALORATH
DAVID G. LAWRENCE
JUDY C. RAMPTON

Tecolote Research, Inc.
5266 Hollister Avenue, No. 301
Santa Barbara, California 93111

2 April 1987

DTIC
ELECTE
OCT 21 1987
S D
CD

APPROVED FOR PUBLIC RELEASE: DISTRIBUTION UNLIMITED

Prepared For

ELECTRONIC SYSTEMS DIVISION
AIR FORCE SYSTEMS COMMAND
DEPUTY COMPTROLLER
HANSCOM AIR FORCE BASE, MASSACHUSETTS 01731



AD-A185 905

87 10 7 013

LEGAL NOTICE

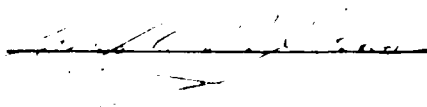
When U.S. Government drawings, specifications or other data are used for any purpose other than a definitely related government procurement operation, the government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

OTHER NOTICES

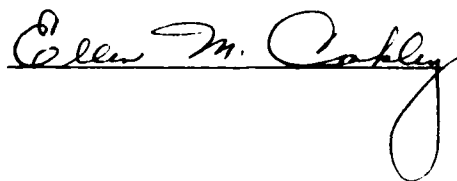
Do not return this copy. Retain or destroy.

" THIS TECHNICAL REPORT HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION."

JOSEPH P. DEAN, Capt, USAF
Senior Software Cost-Research Analyst
Directorate of Cost
Comptroller

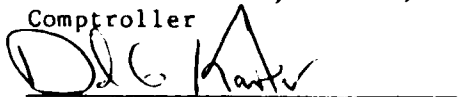


ELLEN M. COAKLEY
Technical Director of Cost
Comptroller



FOR THE COMMANDER

DAVID G. KANTER, Colonel, USAF
Comptroller



Unclassified

SECURITY CLASSIFICATION OF THIS PAGE

ADA185 905

REPORT DOCUMENTATION PAGE

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		5. MONITORING ORGANIZATION REPORT NUMBER(S) ESD-TR-87-167	
4 PERFORMING ORGANIZATION REPORT NUMBER(S) CR-0134/1		7a NAME OF MONITORING ORGANIZATION ELECTRONIC SYSTEMS DIVISION (ACCR)	
6a NAME OF PERFORMING ORGANIZATION Tecolote Research, Inc.	6b OFFICE SYMBOL (if applicable)	7b ADDRESS (City, State, and ZIP Code) Hanscom AFB Massachusetts, 01731-5000	
8a NAME OF FUNDING / SPONSORING ORGANIZATION Deputy Comptroller	8b OFFICE SYMBOL (if applicable) ESD/ACCR	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F19628-84-D-0019	
8c ADDRESS (City, State, and ZIP Code) Hanscom AFB Massachusetts, 01731-5000		10 SOURCE OF FUNDING NUMBERS	
		PROGRAM ELEMENT NO.	PROJECT NO.
		TASK NO.	WORK UNIT ACCESSION NO.
11 TITLE (Include Security Classification) A Plan for Collecting Ada Software Development Cost, Schedule, and Environment Data			
12 PERSONAL AUTHOR(S) Neal J. Brenner, Daniel D. Galorath, David G. Lawrence, Judy C. Rampton			
13a TYPE OF REPORT Technical	13b TIME COVERED FROM TO	14 DATE OF REPORT (Year, Month, Day) 1987 April 2	15 PAGE COUNT 162
16 SUPPLEMENTARY NOTATION			
17 COSATI CODES		18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)	
FIELD	GROUP	SUB-GROUP	
		Ada Software Data Collection	
19 ABSTRACT (Continue on reverse if necessary and identify by block number)			
This document identifies the data elements that need to be collected and methodologies to be used when developing an Ada Software Cost Database. It presents various data collection formats that can be used for data collection and cross references the data elements with ESD TR-87-166.			
20 DISTRIBUTION / AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT ☐ DTIC USERS		21 ABSTRACT SECURITY CLASSIFICATION Unclassified	
22a NAME OF RESPONSIBLE INDIVIDUAL Joseph Dean, Captain, USAF		22b TELEPHONE (Include Area Code) (617) 377-2679	22c OFFICE SYMBOL ESD/ACCR

SUMMARY

The DoD has supported the Ada programming language and Ada Program Support Environment (APSE) to enhance the state of the art of software engineering, produce more reliable software, and reduce overall software life cycle costs. However, at the time of this report, the complete cost ramifications of Ada and the APSE are uncertain.

This report identifies data that can be collected and used to measure the impact Ada has on the cost of developing military-standard software. Several potential cost driver impacts and issues have been identified. They are:

- The time spent in design will probably have a greater impact on Ada projects than non-Ada projects. Good designs may reduce errors and increase reliability. Incomplete or ambiguous designs may cause project failure.
- The allocation of costs to phases in the life-cycle of an Ada project may differ from projects using other languages. (Requirements and design costs phases will probably be higher.)
- Modern software engineering practices may be more critical in Ada developments. Medium- to large-scale projects developed without good software engineering practices may cost significantly more.
- The Ada language and tools may take longer to learn than prior development environments. The formal training level may be a more significant cost driver than traditional Fortran environments.
- Reusability requirements may impact both the use of preexisting software and the development of new software intended to be reused.

- Just the use of certain Ada language features may impact project costs disregarding other development or engineering changes.
- Ada Programming Support Environment (APSE) productivity tools may actually contribute more to cost differences than any other factor.
- Language and tool maturity may impact Ada projects for the next five years or more.
- Data collected on experimental Ada projects may be unduly influenced by the personnel and the fact that the project is being closely monitored (the Hawthorne effect).

This task was completed using the following steps. First, a literature search was performed as described in section 3. Next, an interview questionnaire was created to be used as a basis for interviews with software modelers, Ada software engineering experts, and Ada project managers. Their responses to the interview questions, detailed in sections 4 and 5, expanded the issues and impacts determined from the literature search.

Then, the Ada cost driver issues and impacts were used to modify existing data collection packages to be used to draw out the Ada cost impacts.

Candidate Ada projects were located via the interviews and contacts with Ada coordination groups. These projects range from small to large in scale and include new projects, reimplementations developed primarily to better understand the Ada impacts and experimental, non-mission-critical projects.

The attached data collection plan defines specific steps to be taken during on-site data collection during the next phase of this effort. Included are: types of project personnel to be interviewed, documents to be reviewed, data validation methods, etc.

Additionally, we recommend a tools study be performed to identify available APSE tools and quantify their individual characteristics and performance. The final deliverable of the tools study would be a comparative evaluation in the form of a consumer's guide. This guide would provide the information for the government and contractors to make informed choices and selections of the most appropriate tools.



Accession For	
NTIS CRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Date	
Availability Codes	
Avail and/or	
Unannounced	
A-1	

ACKNOWLEDGMENTS

The majority of the work on this report was performed by Daniel D. Galorath, David G. Lawrence, and Judy C. Rampton, all of Computer Economics, Inc.

We appreciate the time and consideration provided us by the respondents during the somewhat lengthy interview process. We thank Dr. Randall Jensen for his work on the data collection forms and Carolyn Gannon for her review of the entire report.

CONTENTS

SECTION	PAGE
1	INTRODUCTION.....
1.1	SCOPE.....
1.2	PROCEDURE.....
1.3	DEFINITIONS.....
2	LITERATURE SEARCH.....
2.1	LITERATURE SEARCH PURPOSE.....
2.2	LITERATURE SEARCH SOURCES.....
3	INTERVIEWS.....
3.1	INTERVIEW PURPOSE.....
3.2	INTERVIEW PROCEDURES.....
3.3	GENERAL INTERVIEW RESULTS.....
3.4	INDIVIDUAL INTERVIEW SUMMARIES.....
3.4.1	EXPERT A.....
3.4.2	EXPERT B.....
3.4.3	EXPERT C.....
3.4.4	EXPERT D.....
3.4.5	EXPERT E.....
3.4.6	EXPERTS F, G, AND H.....
3.4.7	EXPERT I.....
3.4.8	EXPERT J.....
3.4.9	EXPERT K.....
3.4.10	EXPERT L.....
3.4.11	EXPERTS M AND N.....
3.4.12	EXPERT O.....
4	ADA COST DRIVER IMPACTS AND ISSUES.....
4.1	PURPOSE.....
4.2	DESIGN AND DEVELOPMENT.....
4.3	COST IMPACTS.....
4.4	SOFTWARE ENGINEERING METHODS.....
4.5	IMPACT OF EXPERIENCE, CAPABILITY, AND TRAINING.....
4.6	REUSABILITY ISSUES.....
4.7	IMPACT OF LANGUAGE FEATURES.....
4.8	IMPACT OF AN ADA PROGRAMMING SUPPORT ENVIRONMENT (APSE).....
4.9	MATURITY ISSUES.....
4.10	DATA COLLECTION ISSUES.....
4.11	ITEMS RECOMMENDED FOR DATA COLLECTION.....
5	RECOMMENDATIONS FOR ADDITIONAL RESEARCH.....

CONTENTS (Continued)

SECTION	PAGE
6 DATA COLLECTION FORM STRATEGY AND CROSS-REFERENCE TO ESD FORMS.....	33
6.1 OVERALL CHANGES WITHIN FORMS.....	33
6.1.1 MIL-STD 2167 TERMINOLOGY WITHIN FORMS.....	33
6.1.2 RANGES OF VALUES WITHIN FORMS.....	33
6.1.3 ADDITIONAL COST DATA TO SUPPORT CURRENT COST MODELS.....	33
6.2 ESD FORM ABBREVIATIONS.....	34
6.3 SOFTWARE DEVELOPMENT PROJECT FORM CROSS-REFERENCE...	34
6.4 SYSTEM LEVEL OR CSCI LEVEL DOCUMENTATION FORM CROSS-REFERENCE.....	37
6.5 DEVELOPMENT COMPUTER SYSTEM AND TOOLS FORM CROSS-REFERENCE.....	38
6.6 TARGET COMPUTER SYSTEM FORM CROSS-REFERENCE.....	41
6.7 COMPUTER SOFTWARE CONFIGURATION ITEM FORM CROSS-REFERENCE.....	43
6.8 RESOURCE EXPENDITURE DATA CROSS-REFERENCE.....	48
6.9 COMPUTER SOFTWARE SIZE CROSS-REFERENCE.....	49
BIBLIOGRAPHY.....	50
APPENDIX	
A QUESTIONNAIRE.....	A-1
B RESPONDENTS.....	B-1
C DATA COLLECTION FORMS AND INSTRUCTIONS.....	C-1
D DATA COLLECTION PLAN.....	D-1

INTRODUCTION

1.1 SCOPE

The scope of the research described in this report is to determine the issues and impacts of using Ada for military-standard software development and how to measure those issues and impacts.

1.2 PROCEDURE

The first step was an extensive review of Ada literature for cost driver impacts and issues.

Next, an interview questionnaire was built and personal interviews were conducted with software modelers, Ada software engineering experts, and Ada project managers. Their responses to the interview questions expanded the issues and impacts determined from the literature search.

Then, the Ada cost driver issues and impacts determined during the literature search and personal interviews were used to create new data collection forms. These forms are intended for use during the next phase, actual data collection.

In anticipation of the next phase, a list of Ada projects was compiled, showing candidates for Ada collection based on information provided by interview respondents or contacts with Ada coordination groups. A data collection plan was developed which defines specific steps to be taken during on-site data collection during the next phase of this project.

1.3 DEFINITIONS

Ada

A high-level programming language designed in accordance with the requirements of the Department of Defense. Certain features were borrowed from such modern programming languages as Pascal. It has been designated by the Department of Defense as its official computer language.

APSE

During Ada's development, it became apparent that special-purpose software tools and resources were needed to provide support for Ada applications programs. These tools and resources, called an "Ada Programming Support Environment" (APSE), consist of an integrated set of tools, data base facilities, and control interfaces.

Cost Driver

A factor having an impact on the cost of accomplishing or predicting the cost of a task.

IRAD

Abbreviation for Internal Research And Development. Used to describe a project funded internally by a contractor for research purposes. IRAD projects are not usually subject to military standards.

PDL

Abbreviation for Program Design Language. A formalization of the use of pseudocode.

LITERATURE SEARCH

2.1 LITERATURE SEARCH PURPOSE

The purpose of the literature search was twofold: first, to obtain current assessments and findings relative to Ada software cost estimating approaches, models, and techniques; second, to obtain recommendations relative to Ada cost drivers which should be identified in the Ada Software Data Collection Formats.

2.2 LITERATURE SEARCH SOURCES

To accomplish the literature search, the following sources were searched for pertinent information:

- Ada Clearinghouse
- Data Analysis Center for Software
- Defense Technical Information Center
- Dialog Information Service
- Internal Libraries
- International Society of Parametric Analysts Library
- National Technical Information Service
- Scientific and Technical Aerospace Reports
- Space System Cost Analysis Group Library
- UCLA Engineering Library

Relevant works are listed in the bibliography.

INTERVIEWS

3.1 INTERVIEW PURPOSE

With any relatively new and rapidly advancing technology, published material lags the actual experience of those "in the trenches." The purpose of the interviews was to expand and enhance the data gathered by the literature search to reflect this experience.

3.2 INTERVIEW PROCEDURES

A questionnaire was prepared for use as a basis for the interviews. A copy is attached to this report as Appendix A. Interviewers were allowed to depart from the prepared questions when deemed necessary based on the respondents' experience and prior answers. Interviews lasted between one and two hours and were conducted both in person and by telephone.

3.3 GENERAL INTERVIEW RESULTS

The interviews proved a valuable source of information. Even though responses ranged from extremely positive to very negative, the issues of concern to the respondents were surprisingly uniform. Responses were also consistent among those with similar backgrounds. Respondents seemed to fall into one of three schools of thought regarding Ada:

- The Ada language and tools are a major advancement of software engineering.
- Ada is just another programming language.
- Ada will never have much impact.

Software cost and schedule modelers tended to be very middle-of-the-road in their responses. As a group, they indicated interest in the traditional software metrics and desired to see data similar to that available in the past. Present software models were judged adequate for Ada estimation with the provision that some of the parameters be modified slightly based on Ada experience.

Respondents with hands-on experience ran from wildly enthusiastic to extremely negative. One respondent said Ada's success would be furthered if DoD would quit granting waivers, while another thought DoD should give up on Ada entirely. The most negative responses often came from those viewing Ada as just another language.

Positive responses were more often received from those with more sophisticated tool sets, such as the Rational environment. Some respondents commented on the need for a different overall method of developing software than typical Fortran developments, including additional concentration on data and increased design formality when dealing with Ada development. These respondents insisted that once this Ada experience base is achieved, Ada can be use effectively.

The response to the applicability of Ada to various program types could be predicted based on the respondent's experience with Ada. Those with positive experiences stated it could be used successfully with almost any application. Those respondents having negative experiences saw Ada as useful for only a limited amount of applications, typically non-real-time applications.

The feature of Ada most distressing to the respondents group was the Ada tasking provisions. The most negative felt they would never work, while others were of the opinion that the tasking features just needed maturity.

Overall, strong responses in either direction seemed to follow the respondent's initial conceptions. If the respondent anticipated problems in a particular area, he usually found them. Respondents who approached a problem from the viewpoint of making it work usually received better than expected results.

3.4 INDIVIDUAL INTERVIEW SUMMARIES

This section summarizes the interviews. It attempts to capture the overall tone of the individual interviews in a few sentences. The names of

the experts have been removed and the job descriptions made vague to hide identities. The attributed interviews are in Appendix E of this document, delivered under separate cover. Appendix E is not for distribution.

3.4.1 EXPERT A

Expert A, a Chief Engineer for a major defense contractor, is also the author of a well-known software development cost and schedule estimating model.

The immediate result of using Ada on most large software projects will be to increase costs due to immature software tools, poor support environments, and costs of training. The major productivity gains will probably come from the environment rather than the language. In the long run, Ada-related gains will be typically 30 percent for development and 50 percent for life cycle support.

The software development life cycle will probably change, with more effort spent in design and less required later in the cycle. Failing to provide adequate time for preliminary or detailed design could have greater negative impact on an Ada project than a project using another language.

Requiring software to be reusable may increase development costs 5 to 30 percent. However, reusing code written in Ada should be cheaper than reusing Fortran code.

A new standard is needed for counting lines of code in Ada.

The "wave" of desire for reusability was occurring at the same time as the Ada "wave."

Writing reusable code will be easier in Ada.

Who will be responsible for the maintenance and support of reusable code?

Attempting to compile the PDL prior to critical design review (CDR) is generally not productive, concentrating on language syntax and not design.

Graphical design tools will be helpful.

Include schedule constraints data in the data collection, particularly on the up-front design phase.

3.4.2 EXPERT B

Expert B is an independent consultant and an educator specializing in Ada-related courses.

Anyone with an aptitude for programming can learn to program in Ada. Ada goes hand in hand with good design. Its structure helps enforce good design principles. Since properly developed Ada is so entwined with design, design and implementation will merge. Thus, design will become high-level implementation and implementation will become low-level design. Ada is not tied to any one design method, but to the common principles of any good design method (good design methods clearly define the problem faced, data inputs and outputs, and the solution).

Current implementations of the Ada tasking model seem to be inefficient for some mission-critical application areas. Reliance on current operating systems interfaces has decreased portability to some extent.

We don't know enough about tools to standardize today.

Ada's strong typing and generics features will impact costs.

To speed Ada's acceptance, the Government must stop granting waivers.

While there are no good metrics to measure language training, the enthusiasm of the staff for the Ada language will have a positive effect.

The core of the Ada language (Pascal subset plus exception handling) is easily learned by almost any software developer.

Design and implementation will merge, design becoming high-level implementation and implementation becoming low-level design.

"We don't know enough about tools to standardize."

3.4.3 EXPERT C

Expert C is an Air Force expert on Ada.

Ada projects will cost more over the next two years due to inexperience and tool immaturity. On small projects, Ada won't be much different than what we have now. Large projects will benefit more when Ada matures.

Ada can be applied to a wide application domain. At first it will be more successful with non-real-time applications. It will take time for embedded applications to be adequately supported by optimized compilers and tools.

It will be a while before many language features are used extensively.

Strong typing will reduce some coding errors; however, design errors will not be reduced by the language constructs themselves. These design errors will be detected earlier in the development phases.

Managers will suggest the use of language subsets to avoid inefficiencies in compilers or operating systems.

Ada is a reader's language.

Compared to other languages we are using, writing reusable code will be easier in Ada.

Designers need familiarity with Ada.

Ada PDLs will speed coding. Current tools are not of the quality expected in production environments.

"We suffer with any new technology."

3.4.4 EXPERT D

Expert D is a Systems Engineer with a major computer and software company.

Ada is more readable and maintainable right now, when software personnel are dedicated to these goals. Ada should yield less errors per line of code due to compiler features that find errors early in the development cycle.

Ada-like tool sets can help any language. The point is that Ada tools are a planned set. Thus they will likely be more cohesive.

Management will be helped by management tools that run within the Ada environment. These tools will be able to extract management data from Ada's data base.

Ada can make "design on the fly" easier for small projects since stubs are convenient.

Programmers should constrain any machine dependencies to a few packages. Thus, most Ada code will be portable.

Type statements will reduce costs considerably by eliminating range errors.

3.4.5 EXPERT E

Expert E is a Senior Software Cost Research Analyst for the Air Force and formerly worked as a software engineer.

Ada sizing is difficult since the definition for lines of code is unclear.

Ada training requires about three months with typical people.

Requirements and preliminary design specifications should be language-independent. Ada as a PDL is appropriate after preliminary design review (PDR).

We should see different effort loading and schedules when Ada is used. Violating Brooks' law (Brooks' law states adding more people can impede project performance) may be possible during the coding phase, due to Ada's structure. However, such an overloaded coding staffing plan doesn't make much sense, because testing must be performed with the same staff levels as non-Ada projects.

Additional data needed includes: similarities and differences in tools and compilers between contractors and subcontractors, funding availability, and program stretch-out issues. Additionally, more information regarding rehosting and reuse factors is needed.

3.4.6 EXPERTS F, G, AND H

These respondents all work at an Air Force base and are deeply involved in Ada development. Expert H is the Director, Expert F the Lead Engineer, and Expert G a Staff Engineer. They were interviewed as a group.

If code is not reusable, Ada may be more costly due to the difficulty of writing in the Ada language. So far, Ada has not been able to meet the embedded system requirements due to poor implementation of tasking functions and large object code sizes.

Embedded system reusability should be built into the hardware (e.g., firmware) rather than as reusable software. If you specify reusability, the result will probably be inefficient when embedded system requirements must be upheld.

Current Ada development systems can take hours to recompile when one minor code change is made. Thus, a large machine may not be able to support more than three programmers. Therefore, a large project might require a large capital investment in software development computer systems.

Ada is very readable. It is easier to read than to write. This should simplify maintenance.

3.4.7 EXPERT I

Expert I is a Chief Scientist at a major defense contractor and the author of a well-known software cost and schedule model.

Short term, the Ada learning curve will have a major impact. Long term, if source code size is constant, Ada may reduce costs somewhat. Long term, size and cost will remain similar to other languages. Ada is a "good language," but not a panacea.

In an internal experimental project, source code size doubled and productivity increased. However, the increased productivity was not enough to offset code size increases. These results may be skewed because work was performed by an Ada Research Group in a monitored environment. On any initial experimental projects, Ada can become the goal and not the tool. Programmers can become enamored with the language, detracting from getting the job done.

Use of Ada should provide fewer interface and system problems, but Ada methods will have the greatest impact. Using Ada before PDR is not practical. Designers need pictures (i.e., data flow diagrams, etc.) to convey system concepts. Lack of adequate system design prior to employing an implementation tool like Ada can be detrimental to both software development and maintenance. Ada's support environment will take four to five years to mature.

The use of modern methods will provide most of the gains attributed to Ada.

Who will be responsible for maintenance and support? This question refers to determining both who will bear the cost and who will supply the skilled people. Given the turnover rate of enlisted personnel, they will not be able to develop the experience needed to master Ada.

Existing software cost estimation models can estimate the cost of an Ada project as well as any other project.

We should collect data on the number and type of tools used on a project. Also, terminal response time might become a driver when heavy APSE use is required.

3.4.8 EXPERT J

Expert J works for a well-known firm that markets a software cost and schedule estimation model.

Short term, Ada will be more costly, perhaps as much as 30 percent more than later projects. The APSE cannot be fully forecast, since it is one to two years away. (Fortran required ten to fifteen years before tools had impact, but with Ada it will take much less time.) Using Ada as the PDL will lower costs. For this expert's model calibration and estimation, a constant expansion ratio should be used for the conversion of object instructions to source lines of code.

Data that should be collected includes: source and machine level instructions, application, schedule (by phase if possible), reliability, platform, cost, personnel quality, preexisting design, any complicating factors, and concurrent hardware development.

3.4.9 EXPERT K

Expert K is an engineer and computer scientist.

Using Ada should reduce costs up to 20 percent for projects greater than 1000 lines of code. Ada requires clear, concise, and unambiguous specification.

"You can't fake Ada." You must have a design layout from top down or you can't develop Ada well. If Ada is not used as a PDL, costs will be higher. Ada should be applied after requirements are baselined. A bad specification will cost more if the project is in Ada than in other languages. When Ada is used as a PDL, the design actually becomes the code during implementation rather than requiring another complete transformation.

Failure to develop a good design may cause a project to fail completely.

Ada design errors are more difficult to correct during later phases than for other languages.

Not many people know how to test an Ada job. The relationships between stimulus and response are not completely understood.

Ada trends: Cost of documentation will drop, productivity will jump 3 to 1, the error rate will drop (design errors, also), reusability will be a significant driver.

Portability can be difficult if I/O is involved, since machine dependencies may come into play.

Staffing should be about 80 percent of a Fortran project. It is better to "back off" staff and stay "lean and mean."

Management can be a key driver of Ada projects. There is more potential for management errors. It is especially important to control the design architecture and visibility among modules.

3.4.10 EXPERT L

Expert L is the Avionics and Weapons Systems Integration Tactical Software Engineering Manager for a major defense contractor.

Initially, costs will be higher, due in part to the immaturity of tools. Any implementation of a new language has problems. Costs should be better in time.

Ada will be used much more for large-scale, ground-based systems where timing is not critical. Tools are not currently mature enough for airborne embedded applications; therefore, current projects tend towards large-scale implementations.

Machine code to source expansion ratios are extremely difficult to estimate at this time due to the wide variety among compilers and differences in run-time libraries.

Ada is just another language, with similar types of tools being adapted to other languages. Ada is really a manifestation of computer science coming of age.

Tasking provisions are inadequate. There is no way to tie tasks to an event priority. Pragma are not enough. Also, there is no way to require maximum delays.

Portability problems have occurred with validated compilers. Portability is in the hands of compiler implementors. Compiler writers are not understanding contractors' needs regarding throughput and problem resolution response time. However, Ada should be more portable than other languages due to the compiler validation process.

Initial results with Ada will not be skewed by use of the best people, since many companies don't consider Ada significant enough yet to allocate best people.

The better the Ada coders, the fewer the lines of source code that will be produced. Projects need more people who have a higher overall (design) view of an Ada project and fewer straight coders.

It's unsure at present if development time will be lessened by using Ada unless the project can take advantage of reuse. There may be more design and less coding, with no impact on testing or integration.

On reuse: Initially it will cost more to write software that is generic--the level of documentation is higher and functionality is more general. The overall impact will be beneficial for big contractors who build their own internal reusable software.

Poor quality specifications are expensive regardless of implementation language, and this is probably the same with Ada. However, Ada will point specification problems out earlier.

Using Ada as a PDL will keep costs lower. Ada should be applied to a project after PDR. Ada does not require any design aids, but they can be helpful.

The level of testing is important. We will NOT experience reduced test time with Ada.

Several features of the Ada language will have cost impacts: tasking, overloaded operators, test costs, and lack of traceability for debugging. Debugging will be more difficult because of the unexplored relationship between source, assembly, and machine code.

The cost of buying hardware and tools for development, and the fear of contract penalty costs, will impact overall Ada costs.

Projects in other languages should not be converted to Ada. The industry needs more language interfaces, especially to Jovial. This will allow reuse of "proven" code, particularly on embedded systems.

Additional useful data includes documentation page counts and levels thereof; also, the specification level requested in the RFP, first which military standard, then what tailoring.

3.4.11 EXPERTS M AND N

Experts M and N market a well-known software cost and schedule model. Expert M developed the model.

Because Ada is a big, complex language, it will have a steep learning curve; thus, short-term cost increases. Estimations of Ada projects are similar to Pascal and PL/I. During the 1990s, reusability will become a significant cost reduction factor. Long term, additional reliability may be achieved due to reuse of fully tested reused software.

Standard Ada tool sets will be used by the more disciplined software organizations. They do well with any new technology. Ada must grow to the full APSE. This may be a stumbling block if it doesn't.

Ada may lend itself to some schedule compression due to parallelism during implementation of package designs.

More high-quality data is needed. Additionally, iterative cost, time, environment, people, and system data should be collected. These should include actual staffing and resources by month, by labor category. Additionally, the monthly defect pattern (rate and cumulative), by severity, is needed.

3.4.12 EXPERT O

Expert O is a Senior Staff Engineer and the Principal Investigator for the Ada C3 IRAD for a major defense contractor.

The potential of Ada is very positive. Overall, costs can be significantly reduced with proper use of Ada. However, there is extra cost in the early development. Testing of code has moved to testing of design. The design phases are critical, with coding trivial. Integration is no longer the driver. Ada as a PDL is a natural evolution. Ada makes prototyping easier. Ada should be applied to a project from the beginning.

Ultimately, Ada will be usable for all types of projects. Currently, 1750-based avionics projects are difficult due to performance and memory constraints (compiler immaturity). Compilers should become available to solve these problems.

The biggest problems are tasking and reusability.

Ada promotes more reliable software. Ada projects may be able to violate Brooks' law (increase staffing after PDR by assigning packages to different groups).

Size in lines of code will decrease with Ada as compared to Fortran. It is clear that an Ada line does a lot more than a Fortran line.

PDR and CDR schedules should require 20 percent more time than Fortran or Jovial projects. However the code, unit testing, and integration are reduced dramatically. The language itself provides more productivity by uncovering errors earlier in the life cycle. However, the most improvement comes from use of the APSE and modern software engineering techniques.

Software life cycle and development costs can be reduced now with the use of Ada and a good compiler. The impact on development productivity (lines/man-month) is hard to predict, because both size and effort will be less. For example, if for implementing a specific capability in Ada vs Fortran the effort is decreased by x percent and the lines are decreased by $2x$ percent, the cost falls but so does the productivity. Thus, the idea of increased productivity refers to "absolute work" productivity.

Success with early projects may be somewhat skewed due to application of best people, but future projects will benefit from the experience of "superstars" for good ground work.

The number of data types and number of data objects would be interesting as a sizing metric.

Ada COST DRIVER IMPACTS AND ISSUES

4.1 PURPOSE

The purpose of this section is to discuss the major Ada cost driver issues and impacts identified. These impacts and issues are the key influences in the modification of the data collection forms.

4.2 DESIGN AND DEVELOPMENT

Almost all respondents identified the design phase as being especially important to a project using Ada. The only difference is the degree to which they feel design is important.

Several experts feel that the design phase is critical to the development. Some experts feel that failure to develop a good design may cause a project to fail completely. One expert reminded us that a bad design can cause serious problems in non-Ada implementations.

Ada design errors are more difficult to correct during later phases than for other languages. In particular, the practices of "tuning" or "patching" during integration are more difficult and time-consuming. This is due to the interdependencies enforced by the compiler and the promotion of top-down design by Ada [Chang-84]. There also may be more potential payoff in improving the efficiency of analysis and validation methods than in speeding up the coding activity [Boehm-82].

Changes in the software development life cycle were also projected. In general, design will become more important and take longer. One expert suggested that PDR and CDR schedules should increase by 20 percent over Fortran or Jovial projects, with dramatic reductions in the code, unit test, and integration schedules. One expert felt that failure to allow extra time for the design of an Ada project would have an inversely proportional impact on project costs.

In addition, most respondents feel using Ada as a PDL will aid the transition from design to implementation, eliminating one translation. Two experts see Ada being used today as just another language, due to tool immaturity. They do feel using Ada as a PDL may reduce costs by locating errors earlier in the development cycle. They added that any standard pseudo-code would be helpful in this regard.

A third feels designers need familiarity with Ada. Also, he stated Ada PDLs will speed coding. A fourth goes even further, stating the design and implementation will merge, design becoming high-level implementation and implementation becoming low-level design. A fifth agrees, saying the design becomes the code.

One expert is opposed to the use of Ada as a PDL prior to PDR. He feels a more visual representation of the design (i.e., data flow diagrams, etc.) is necessary at that point. A second feels that attempting to compile the PDL prior to CDR is generally not productive, concentrating on language syntax and not design. He also mentioned graphical design tools will be helpful.

Ada may allow schedule compression due to greater parallelism during the implementation of package designs. Two experts pointed out that it may be possible to violate Brooks' law after PDR or during coding, but one pointed out that this may not be a wise staffing plan, since test staffing levels will be the same as non-Ada projects.

One expert stated, "Nobody knows how to test an Ada job." He feels the relationships between stimulus and response are not completely understood. Another agreed that the level of testing is important. He was adamant that test time not be reduced. The amount of time spent in test correcting problems may be one measure of the quality of the design [Newell-85].

4.3 COST IMPACTS

Most respondents expect Ada's goal of reduced costs to be met once the Ada development environment matures. Initially, almost all respondents said Ada projects will cost more.

Two reasons often given for increased early costs are staff inexperience and inadequate tools (an issue also raised at the AdaJUG meetings). There will be costs for education and inefficiency until the staff becomes fully productive. The length of time required for recompilation when a minor change is made can require hours. Thus, a large capital investment may be required to support an Ada development. Also, design and testing tools are few.

Several respondents believe Ada will never offer any advantage over prior languages. Several stated Ada may be more costly if code is not reused, due to the difficulty of writing in the Ada language, while another expects Ada to mostly benefit large projects. He said Ada use on small projects will not differ much from current experience with other languages.

According to those respondents seeing reduced costs for Ada projects, the savings will occur after CDR. Therefore, costs allocated to design effort are expected to take a larger portion of the overall cost. One said his experience (a small project without military-standard documentation requirements) showed coding became trivial and integration was no longer a driver. Another said effort will be much more front-loaded.

Reduced costs later in the development cycle may be attributed to the fact that the cost to fix errors increases with the stage in the life cycle at which it is fixed [Hamer-85 and Wallis-85]. One experience seems to indicate that errors are located earlier. Another also mentioned he believes Ada will reduce effort at the "back end."

Ada's potential impact on maintenance costs is a split decision, with some of the respondents seeing increased cost and others less cost. Two called Ada "a reader's language." However, concern over the ability of

maintenance personnel to handle the complexity of the language was most often cited as a reason for increased cost. Also, maintenance staff turn-over was identified as a concern, giving them less time to develop an understanding of the code.

Those foreseeing reduced maintenance costs feel that Ada allows a more direct relationship between design and implementation, making the code easier to understand, especially when Ada is used as the PDL. Increased readability due to Ada's English-like constructs is also advanced as a reason maintenance costs should be lower.

A couple of modelers think that current models can be used to estimate the cost of Ada developments. Estimations of Ada projects are similar to Ada and PL/I. (At the AdaJUG meetings, a number of Ada users do not know how to estimate the cost of Ada projects.)

4.4 SOFTWARE ENGINEERING METHODS

The role of modern software engineering techniques may be greater with Ada projects. Ada features alone do not ensure good software development practices. In fact, Ada's very power generates more difficulty in making design decisions. Methods that aid engineers in the allocation of function and data structures are beneficial [Roy-85].

One modeler believes the use of modern methods will provide most of the gains attributed to Ada. An expert views modern design methods with Ada as a coincidental coming of age for computer science.

Some believe traditional milestones may no longer be appropriate for Ada or accurately reflect the actual state of the project. For example, CDR may be harder to pin down. More design details might be left to be finalized during implementation.

The criteria for meeting milestones might be different with Ada projects. Suggested milestones include noting the point in the software life cycle when all the packages are named, when all procedures are named,

when the type statements are identified, and how many objects are defined at those times. If milestones are changing, managers may need to respond to those changes. (This has been mentioned at AdaJUG meetings.)

Some respondents suggested that managers accustomed to experiencing the most difficulty during integration phase might push staff toward integration prematurely. This will force the design phase, felt to be critical to the success of an Ada project, to be abnormally shortened. This could create exactly the integration problems that the manager had been attempting to avoid.

4.5 IMPACT OF EXPERIENCE, CAPABILITY, AND TRAINING

Staff experience, capability, and training may have more cost impact on projects using Ada. Almost without exception, respondents stated modern programming techniques and concepts will be more significant than prior experience with other, more procedural languages. When offered a choice between a recent college graduate with training in structured techniques and Pascal or experienced Fortran personnel, most opted for the recent graduate.

Also, since Ada is a relatively new language and APSEs are still evolving, there is not a body of trained personnel. For the foreseeable future, many projects will require staff training. One Ada teacher stated there are no good metrics to measure language training, the enthusiasm of the staff for the Ada language will have a positive effect.

In addition, most respondents feel Ada requires more experience than other languages before personnel are proficient. It is a split decision as to whether Ada syntax is more difficult to understand than other languages. However, it is generally agreed that it will take longer to appreciate the trade-offs in determining which feature of the language is best to implement a particular algorithm.

Most respondents stated that Ada will take longer to learn than other languages, although personnel should be able to produce Ada projects

before mastering all the language features. Not only is the language felt to be richer, but the tool set is more complex. This may increase the time to proficiency even further, impacting first, second, and even future projects until a "fusion" point is reached. However, one feels that anybody can learn a "base" Ada subset that may be used to implement almost any software function.

4.6 REUSABILITY ISSUES

One of the great hopes for Ada is that future software will take advantage of previous, reusable components, lowering development costs. One expert noted the "wave" of desire for reusability was occurring at the same time as the Ada "wave." Unfortunately, according to the respondents, just using Ada for development does not assure reusability. However, two experts feel that writing reusable code will be easier in Ada compared to other languages we are using.

All respondents who discussed reusability feel it is more expensive to develop software for reuse. Increased documentation and reliability requirements were mentioned as primary cost drivers. It was suggested that only large software developers that maintain internal software libraries would benefit. While one expert agrees that costs will increase initially, he feels additional costs will be nominal after personnel achieve an understanding of reuse issues.

The political ramifications of reusability were mentioned by several experts. The question most often posed: Who will be responsible for support? Due to these political issues, most feel the only reuse will be contractors reusing their own code.

Part of the reusability issue is rehosting. Operating systems have made the rehosting of Ada code difficult. Also, some Ada features perform tasks that are usually thought to be operating system tasks. Encountering unimplemented features in validated Ada compilers slows rehosting and thus reusability.

4.7 IMPACT OF LANGUAGE FEATURES

Several language features were mentioned as potentially impacting productivity. For example: packages (collections of related programs and data) are seen as potential productivity enhancements; overloaded operators (the feature of giving a new meaning to an operator, useful for defining arithmetic for types that are not built into Ada) have mixed forecasts of their impacts; strong typing (the restriction against mixing data type across assignments in expressions) is seen as a productivity improvement; and generics (a method of overcoming Ada's sometimes overly restrictive typing) [Saib-85] are seen as potentially dangerous. The enforcement of inter-module dependencies by the compiler is viewed as extremely helpful.

The feature most often mentioned is Ada's strong typing. According to some, this should decrease programmer-induced errors and allow them to be discovered earlier in the development cycle. Other respondents mentioned the reverse side of strong typing. Type conversions are more difficult, and the overuse of derived types can make Ada source code more obscure and complex than necessary. It was suggested by one respondent that, while Ada will lead to early detection of programmer errors, it will not eliminate design and logical errors.

One expert foresees fewer interfacing and system problems. This view has also been expressed at AdaJUG meetings.

Ada is supposed to be an easy language to read, easier to read than write. This reading ease should simplify maintenance.

As discussed in Maturity Issues (Section 5.9), tasking received significant concern from the respondents. At AdaJUG meetings it has been suggested that tasking can create difficult debugging and integration problems. Exception handling routines can also require more expertise during later phases.

4.8 IMPACT OF AN Ada PROGRAMMING SUPPORT ENVIRONMENT (APSE)

According to most respondents, the power of Ada will be derived not only from the richness of the language, but the use of an APSE [Babich-83]. The respondents predicting the greatest success for Ada often view Ada and the APSE as inseparable. (APSE is used in the context of the environment used with the language. This is not intended to imply that there will be one standard APSE.) Those having negative experiences with the language were usually using Ada as simply another programming language.

Though most agreed an APSE is still in the development stage, some respondents thought a good APSE might contribute more to the success of a project than the Ada language itself. One mentioned an APSE directly supporting modern development methods further enhances programmer productivity.

Several respondents think a standardized tool package is a noble idea, but not realistic. "We don't know enough about tools to standardize." One said it will be two years before tools have an impact on costs. (He added that Fortran required 10 to 15 years before tools had impact, but with Ada it will take much less time.)

Another suggested collecting data on the number and type of tools used on a project. He also suggested terminal response time might become a driver when heavy APSE use is required.

One expert expressed concern that the available APSEs do not provide an acceptable listing showing the relationship of source to assembly to machine code for debugging purposes. This view has been supported by other software developers at AdaJUG meetings. Lack of symbolic debuggers will keep near-term costs higher.

4.9 MATURITY ISSUES

As two experts stated, "Any implementation of a new language has problems," and "we suffer with any new technology." In that respect, if any one particular feature of Ada came under fire during the study, it was

tasking. Almost all respondents feel the current tasking implementations are inefficient and therefore unusable. However, when pressed, many respondents feel the design of the tasking provisions in the language is adequate.

However, the opposite view has been expressed at AdaJUG meetings. Ada lacks a suitable method of providing regularly scheduled tasks within maximum time constraints.

Lack of efficient tasking has led some to condemn the entire language and predict it will not succeed. Others anticipate that as compiler writers become less concerned with passing validation and turn their attention to optimization, this problem will go away [Van der Linden-85].

The optimization issue was explored further. Optimization is a near-term problem. Ada does not currently meet embedded (airborne) system needs because of large object code sizes and slow run times, while the processors have limited memory and severe timing restrictions. Ada will be used more for large-scale ground-based systems until these problems are solved. Eventually, compiler writers will solve these problems.

The actual relationships between source, assembly, and machine code are not yet appreciated, which one expert called a "lack of traceability." Implementations can also vary widely between different compilers and operating systems.

Some respondents predicted managers will suggest the use of language subsets to avoid inefficiencies in compilers or operating systems. Language features that have been poorly implemented by compilers, operating systems, or the hardware may not be used. Also, more subtle features of the language (such as generics) may be avoided. Some advanced Ada features may be lost if there is widespread use of subsets to mimic other more familiar languages [Hummel-84].

Confidence in existing software libraries with proven track records may also slow the move to Ada. There is a reluctance to translate long-standing libraries to Ada.

It has been suggested that interfaces to other languages be further developed to allow the use of existing libraries in other languages to benefit current Ada projects. Projects developed in other languages should not be converted to Ada. Instead, new features should be implemented in Ada.

The immaturity of Ada tools was noted by most respondents. According to one, current tools are not of the quality expected in production environments. Quality of the development environment is considered significant [Roy-85]. Experts think it will be three to five years before the APSE matures.

Respondents with a positive attitude towards Ada generally feel that Ada will mature faster if the government quits granting waivers. At the AdaJUG, some participants believe the problems with Ada will be solved if everyone is forced to use it. SPOs, on the other hand, only want to use Ada if there is an implementation for their target machine; they do not have the budget or schedule to solve language problems.

4.10 DATA COLLECTION ISSUES

Data from early projects may not accurately reflect the true impact of Ada. Two major reasons are advanced. First, for the simple reason that when a project is closely monitored, performance improves (the Hawthorne Effect).

Second, most respondents agree first projects may involve better and more motivated staff than typical projects. For example, some Ada research groups will perform initial development. One modeler specifically recommends against collecting data from projects involving such research groups since Ada often becomes the goal rather than the tool and clouds the results.

Respondents disagree on the amount first projects will differ from subsequent Ada projects. One says they can cost as much as 30 percent more. Another feels that initial Ada projects are not significant enough within many contractors to get the best people, as suggested above. He attributes the difference between first and subsequent projects to the learning curve.

4.11 ITEMS RECOMMENDED FOR DATA COLLECTION

Schedule is considered to be a cost driver, especially when it is constrained. These constraints should be collected, particularly on the design phase, which many experts believe to be critical. Funding availability, which impacts schedule, can also affect cost.

The similarities and differences in tools and compilers between contractors and subcontractors should be collected. The number and type of tools are of interest. Also, terminal response time might become a driver when heavy APSE use is required.

Instead of a single collection, iterative collection of cost, time, environment, people, and system data was suggested. These should include actual staffing and resources by month and by labor category. Additionally, the defect pattern (rate and cumulative), monthly, by severity, was recommended.

There was the suggestion that a new metric is needed for measuring size. The definition of a line of code is unclear. Also, because compilers vary in efficiency, the expansion ratio is difficult to project. One suggested metric is number of data types and number of data objects. In addition to metrics for measuring code size, metrics for measuring reused software are needed.

Additional useful data includes documentation page counts and levels thereof. Also, the specification level requested in the RFP: first, which military standard; then, what tailoring.

RECOMMENDATIONS FOR ADDITIONAL RESEARCH

5. In addition to the actual data collection, we recommend several other research projects. These projects can be divided into two general areas. First are studies that will provide data for estimating software costs. Second are studies that will help SPOs estimate costs other than direct labor, so they can better budget the total cost of doing an Ada development.

In the first category are costs for requirements and maintenance. Requirements generation cost, when there is concurrent hardware development, is a system level cost that contains both software and hardware costs. Currently there is no attempt made to isolate the software portion of these costs. This is also true even when there is no hardware development.

Lower maintenance costs is one of the hoped-for results of Ada developments. Also, maintenance costs for software in mature higher-order languages are hard to project. A study to determine what quality (e.g., open trouble reports and engineering change proposals and severity) and complexity metrics at software turnover should be measured, and what operational environments influence maintenance costs should be initiated, so that data to be collected is identified and a procedure for data collection is in place and ready. A pilot study to generate and test hypotheses on mature language developments would lay groundwork for the Ada effort.

Current software cost estimating models fail to take a systems approach to cost analysis. They rely on size as the primary driver, and during a project's system definition phase, size estimates are hard to make and usually are not very good. This leads to underestimated costs and schedules.

For early estimates, a model that relies on characteristics of the system being developed is needed. System parameters are more stable than

code size estimates and would therefore produce better early estimates. Cost and schedule estimates could be made from code size once the project has advanced to the point where good size estimates can be made.

The second type of study involves indirect costs such as education and capitalization. Because there is a lack of Ada experience, staff must be educated. SPOs will need an idea of the costs of education and the time required to make programmers and designers proficient in Ada and modern design methods.

Because development of Ada code requires more central processor time and memory than current higher order languages, SPOs should expect much higher capitalization costs. There should be a document that quantifies these costs by development environment type and also contrasts the capabilities and benefits of the different environments.

We recommend a tools study be performed. This study will explore the history of software tools and development methods and their impacts on productivity, and identify the trends in productivity gains. This information will be combined with exploration of expected advances, especially with respect to Ada Programming Support Environments. The goal will be to project, on a time line, the improvements that can be expected due to the improved tools and methods. This will provide cost estimators a guideline when trying to account for tools in estimates of future Ada (and non-Ada) projects.

Finally, SPOs need a resource that describes compiler performance. There are compiler studies being done by industry, and ESD would benefit by collecting these studies and their updates so SPOs can use them for reference.

DATA COLLECTION FORM STRATEGY AND CROSS-REFERENCE TO ESD FORMS

This section provides a cross-reference from the Ada data collection package (Appendix C) to the ESD software cost data base collection package (referred to as the ESD forms). Additionally, the rationale for differences between the Ada forms and the ESD forms are addressed where appropriate.

6.1 OVERALL CHANGES WITHIN FORMS

In some cases, minor heading wording changes are made for similar data. These are not individually identified.

6.1.1 MIL-STD 2167 TERMINOLOGY WITHIN FORMS

MIL-STD 2167 terminology is used in the Ada data collection package. This causes some semantic differences between the two forms that are not individually identified.

6.1.2 RANGES OF VALUES WITHIN FORMS

Since the software environment is so important to Ada and some environment data is difficult to collect precisely, several items are collected as ranges, using the least, expected, and maximum values rather than single values. This technique quantifies the uncertainty. In cases where actual values are known, the form has a place for this actual value instead of the range.

Additionally, this form arrangement can collect both the original estimates and the actual data to see the differences.

6.1.3 ADDITIONAL COST DATA TO SUPPORT CURRENT COST MODELS

Several data are added to support the major software cost models as of 1986. These are not all Ada-specific drivers, but are identified as major software cost drivers. The ESD forms are missing several of these data items.

6.2 ESD FORM ABBREVIATIONS

In the cross-reference tables, the ESD forms are referred to by the following abbreviations:

SDP	Software Development Project Summary Data Form
DTC	Development and Target Computer Data Form
CPCI	Computer Program Configuration Item Summary Data
RED	Resource Expenditure Data Form
CPCIFSD	Computer Program Configuration Item Function and Sizing Detail Data Form

6.3 SOFTWARE DEVELOPMENT PROJECT FORM CROSS-REFERENCE

Ada Question	Ada Topic	ESD Form	ESD Question	ESD Topic
1.1	Project Name	SDP	1.	Project Name
2.	Development Contractor/ Organization	SDP	2.	Development Contractor/ Organization
3.1	Mission Description	SDP	3.1	Mission Description
3.2	Major Hardware Interfaces	SDP	3.2	Major Hardware Interfaces
3.3	Major System Functions	SDP	3.3	Major System Functions
3.4	Major Software Functions	SDP	3.4	Major Software Functions
3.5	Number of CSCIs	SDP	3.5	Number of CPCIs
3.6	Computer Software Configuration Item (CSCI) List	SDP	3.6	CPCI Names
3.7	System User	SDP	3.7	System User

Ada Question	Ada Topic	ESD Form	ESD Question	ESD Topic
3.8	Relative Magnitude of the Software Effort	[NONE]		
	Will provide information on whether the project is primarily software, hardware, or a mixture.			
3.9	Software Development Standards	[NONE]		
	Will help determine level of DoD standards applied to the project.			
4.	Project Milestones	SDP	6.	Development Schedule
	Military Standard 2167 terminology added.			
5.	Modern Development Method Use	SDP	4.1	Specification
5.	Modern Development Method Use	SDP	4.2	Design
5.	Modern Development Method Use	SDP	4.3	Development
5.	Modern Development Method Use	SDP	4.4	Coding
5.	Modern Development Method Use	SDP	4.5	Testing
5.	Modern Development Method Use	SDP	4.6	Validation/ Verification (Inspection)
5.	Modern Development Method Use	SDP	4.7	Formalisms

Development practices may be an important part of Ada's software engineering evolution. Thus, specific development methods are included to capture their impacts during development. Both Ada-related practices and other, less sophisticated practices are

Ada Question	Ada Topic	ESD Form	ESD Question	ESD Topic
-----------------	--------------	-------------	-----------------	--------------

added. This should provide data regarding the impact of software engineering methods on Ada. Additionally, the level of expertise with each practice is collected, since methods may improve productivity when developers are experienced, but impede project progress when they are first learned.

- | | | |
|----|------------------------------|--------|
| 6. | Software Quality
Required | [NONE] |
|----|------------------------------|--------|

The data will provide a qualitative assessment of the required software quality. This data may be contrasted with data regarding the software goals and costs. The following quality goals are included:

- Usability
- Reliability
- Efficiency
- Integrity
- Testability
- Portability
- Correctness
- Maintainability
- Reusability
- Interoperability

- | | | | | |
|----|----------------------------|--------|----|----------------------------|
| 7. | Software Change
History | SDP | 8. | Software Change
History |
| 8. | Comments | [NONE] | | |

6.4 SYSTEM LEVEL OR CSCI LEVEL DOCUMENTATION FORM CROSS-REFERENCE

Ada Question	Ada Topic	ESD Form	ESD Question	ESD Topic
1.	Project Name and Date			
2.	Development Contractor/ Organization			
3.	CSCI Name if CSCI Level			
4.	Document	SDP	7.	Documentation
4.	Document	CPCI	9.	Documentation
4.	Document	CPCI	7.	Quality of Specification

This form replaces the documentation questions in the ESD SDP and CPCI forms. It uses 2167 terminology and presents a more comprehensive list of documents. It also asks if the document is GFD or provided to the project, whether the contractor wrote the document, for the quality of the document, and the date the document was completed.

6.5 DEVELOPMENT COMPUTER SYSTEM AND TOOLS FORM CROSS-REFERENCE

Ada Question	Ada Topic	ESD Form	ESD Question	ESD Topic
1.1	Development Contractor/ Organization Name			
1.2	Development Contractor/ Organization Location			
2.	Project Name			
3.1	Development System Attributes	DTC	2.1	Information if different from target computer
3.1	Average People per Terminal	DTC	2.4	Average Engrs./ Programmers per Terminal
3.2	Access Modes	DTC	2.3	Access Modes
3.3.1	Turnaround Time No Recompile	DTC	2.2	Turnaround Time
3.3.2	Turnaround Time Recompile Required	DTC	2.2	Turnaround Time

Turnaround time is subdivided to collect separate data encompassing both the simple edits, and the time delays created when a minor change causes a major recompile (which could take hours). These differences may cause major differences within Ada developments.

3.3.3 Terminal Response [NONE]

Since Ada can be more dependent on support tools, terminal response can play a larger role in developer productivity, with slow terminal responses having a significant impact on the tool usefulness and developer morale.

3.3.4	Major Changes per Month	DTC	1.8	Virtual Machine Volatility
More detail.				
3.3.5	Minor Changes per Month	DTC	1.8	Virtual Machine Volatility

Ada Question	Ada Topic	ESD Form	ESD Question	ESD Topic
-----------------	--------------	-------------	-----------------	--------------

More Detail.

3.4	Hours Development System Operates	[NONE]		
3.5	% Development System Hours Available	DTC	2.9	Development Computer Resource Availability
3.6	Development System Security Classification	[NONE]		

Development costs may be influenced by the level of security imposed on the development computer.

4.	Tools	SDP	5.	Software Development Tools Used
----	-------	-----	----	---------------------------------------

Tools should impact Ada costs significantly. Ratings of specific automated or manual (i.e. methods) tools or lack thereof, are included to capture their impacts during development. Both Ada-related tools and more general tools are listed for two reasons. First, tools listed on the ESD form are added to ensure traceability. Second, some fairly primitive tools may be required to bootstrap Ada developments.

Additionally, the level of expertise and frequency with which each tool is used is collected since tools may improve productivity when developers are experienced, but impede (or only slightly improve) project progress when they are first learned.

Specific life cycle phases where each tool is applied to the project is added to collect data regarding the appropriateness and impact of tools.

The CSCI affected by the tool use is asked for so the effect of the tool on the CSCI cost can be measured.

5.1	Number of Development Sites	DTC	2.5	Number of Development Sites
5.2	Development Computer and Site Locations	DTC	2.6	Development Site Locations

Ada Question	Ada Topic	ESD Form	ESD Question	ESD Topic
6.	CSCIs Developed on Development Computer	[NONE]		
	A project can have several different development computers with different tool sets.			
7.	Comments	[NONE]		

6.6 TARGET COMPUTER SYSTEM FORM CROSS-REFERENCE

Ada Question	Ada Topic	ESD Form	ESD Question	ESD Topic
1.1	Development Contractor/ Organization Name			
1.2	Development Contractor/ Organization Location			
2.	Project Name			
3.1	Target Computer Manufacturer and Model	DTC	1.1	Manufacturer and Model Number
3.1.1	Main Memory Size (Words)	DTC	1.2	Main Memory Size in Words and Word Size in Bytes
3.1.2	Word Size	DTC	1.2	Main Memory Size in Words and Word Size in Bytes
3.1.3	Number of Processors in the Target	[NONE]		
Multi-processor systems can cause additional levels of effort throughout the development cycle in either Ada or non-Ada developments.				
3.1.4	Maximum Main Memory Size	DTC	1.3	Maximum Main Memory Size
3.1.5	Virtual Memory Machine	[NONE]		
Virtual memory target removes memory space constraints.				
3.1.6	CPU Processing Speed (MIPS)	DTC	1.4	CPU Processing Speed
3.1.7	Reserve Memory Requirement	DTC	1.5	Reserve Memory Requirement

Ada Question	Ada Topic	ESD Form	ESD Question	ESD Topic
3.1.8	Reserve Timing Requirement	DTC	1.6	Reserve Timing Requirement
3.1.9	Programming Languages	[NONE]		
This data will provide information when Ada is required to interface with other programming languages.				
3.1.10	Difference Between Development and Target Computer	DTC	2.1	Difference Between Development and Target Computer
3.1.11	Accessibility to Target	[NONE]		
The target machine(s) may not be fully accessible. This data will point out productivity differences for any type of project. Additionally several persons have stated concerns regarding Ada target debugging efficiency and difficulty.				
3.1.12	Target Simulator Size	[NONE]		
This data identifies potential memory constraints not directly related to the total target machine size.				
3.2.1	Major Changes per Month	DTC	1.8	Virtual Machine Volatility
3.2.1	Minor Changes per Month	DTC	1.8	Virtual Machine Volatility
3.3	Concurrent Development with Software	DTC	1.7	Concurrent Development with Software
4.	CSCIs Executed on Target	DTC	1.12	CPCIs Hosted This Computer
5.	Comments			

6.7 COMPUTER SOFTWARE CONFIGURATION ITEM FORM CROSS-REFERENCE

Ada Question	Ada Topic	ESD Form	ESD Question	ESD Topic
	[DROPPED]	CPCI	8.8	Languages Used
This has been moved to the Development and Target Computer Summary Data Form to cover any cases where other languages are interfaced with Ada.				
1.1	CSCI Name	CPCI	1.	CPCI Name
1.2	Development Contractor/ Organization			
1.3	Project Name			
2.	CSCI Functional Description	CPCI	2.	Functional Description
2.1	Operating Environment	[NONE]		
3.1	Milestones	CPCI	3.1	Milestone Data
3.2	Schedule Acceleration/ Stretchout Assessment	CPCI	3.2	Schedule Acceleration/ Stretchout Assessment
4.1.1	Analyst Quality	CPCI	4.2	Average Personnel Quality Percentile
4.1.2	Programmer Quality	CPCI	4.2	Average Personnel Quality Percentile
4.1.3	Team Programming Language Experience	CPCI	4.1.C	Average Experience Languages Used
4.1.4	Development Methods Experience	CPCI	4.1.B	Average Experience Techniques Used

Ada Question	Ada Topic	ESD Form	ESD Question	ESD Topic
4.1.5	Average Quality and Experience Development Virtual System Experience	CPCI	4.1.D	Average Experience Virtual Machine
4.1.6	Average Quality and Experience Applications Area Experience	CPCI	4.1.A	Average Experience Applications Area
4.1.7	Support Software/ Tools Experience	CPCI	4.1.E	Average Experience Support Software Tools

The overall rating of the tools experience will further quantify the Ada relationship between language and tools. Additionally, it will provide a cross-check to the detailed tool ratings and ensure tool usage is not misstated.

4.2	Average Formal Training	[NONE]
-----	----------------------------	--------

Some persons have stated that self-taught people will not be able to learn Ada as readily as those who are formally trained. This data quantifies the developers' formal training.

4.3	Peak Designer Staff	CPCI	4.4	Peak Manloading
4.4	Peak Programmer Staff	CPCI	4.4	Peak Manloading
4.5	Peak Tester Staff	CPCI	4.4	Peak Manloading
4.6	Maximum Staffing Rate	[NONE]		
4.7	Overall Personnel Availability	CPCI	4.3	Manpower Availability
5.	Reliability Requirement	CPCI	5.	Reliability Requirement
6.1	Inherent Difficulty of Application	CPCI	6.	Complexity

Ada Question	Ada Topic	ESD Form	ESD Question	ESD Topic
6.2	Inherent Complexity of Data Structures	[NONE]		
This data may provide insights into the cost impacts of Ada's data handling capability.				
6.3	System Integration and Test	[NONE]		
7.1.1.1	Total Size Excluding Documentation	CPCI	8.1	DSLOC Excluding Documentation
7.1.1.2	Documentation Lines	CPCI	8.2	Documentation Lines
7.1.2	Operational Response Requirements	CPCI	8.5	Operational Response Requirements Distribution
7.1.3.1	Source Statement Mix Executable	CPCI	8.6	Source Statement Type Mix
7.1.3.2	CSCI Source Code Mix	CPCI	8.4	Size Breakdown by Operation as a Percent of Item 8.1
7.1.4.1	Memory Constraint Percent	DTC	1.10	CPU Memory Constraint Evaluation
7.1.4.2	CPU Time Constraint Percent	DTC	1.11	CPU Time Constraint Evaluation
7.1.4.3	Real Time Operation Percent	CPCI	8.5	Operational Response Requirements
7.1.4.4	Multi-Processor Percent	[NONE]		

This data code effort required to develop multi-processor
functions due to multi-processing requirements. It will be
useful for both Ada and non-Ada projects.

Ada Question	Ada Topic	ESD Form	ESD Question	ESD Topic
7.1.4.5	Multi-target Percent	[NONE]		
This data identifies CPCIs that must run on multiple targets and alerts the analyst that there may be extra effort.				
7.1.5	CSCI Reused Code From Other Projects	CPCI	8.9	Reusable Code From Other Projects
More detail.				
7.2	Function Point Data	[NONE]		
7.3	Size/Complexity	[NONE]		
Number of Ada Objects, Program Units, Layers of Program Units, Blocks.				
8.1	Total Data Base Size (Words)	CPCI	8.3	Data Base Size in Bytes or Characters
8.2	Total Unique Data Items	CPCI	8.3	Data Base Size in Bytes or Characters
8.3	Total Number of Records	CPCI	8.3	Data Base Size in Bytes or Characters
8.4	Unique Data Types	CPCI	8.3	Data Base Size in Bytes or Characters
9.	Special Display Requirements	CPCI	8.7	Special Display Requirements
10.	Software Failure History	CPCI	10.	Software Failure History

Failure history is expanded to collect not only the number of errors, but the phase when these errors are introduced. Several persons stated we will find errors earlier in the life cycle on Ada developments. This data should help show Ada's impact on reliability and software anomalies. Additionally, this data should help assess the developer capabilities and tools impact on reported errors.

Ada Question	Ada Topic	ESD Form	ESD Question	ESD Topic
11.	Software Change History	SDP	8.	Software Change History

This now measures the impact of requirement changes on the CSCI.

12.1 Development Environment

12.1.1 Resource Dedication [NONE]

This data will quantify costing impacts of sharing development computers and other resources.

12.1.2 Resource/Support Location [NONE]

This data will provide insights into the productivity impacts of the availability of expert advice and various project resources to the developers. This may be especially important while Ada is still new to many developers.

12.1.3 Security Level [NONE]

Security can be a major cost driver on any software project. This data may be especially useful when comparing Ada experimental projects with actual mission critical software.

12.1.4 Contract Type [NONE]

12.2 Specific Development Goals [NONE]

The following potential development goals have been added to collect data regarding differences in cost due to perceived or real development goals:

- Maximum Maintainability
- Maximum Reuse of Pre-existing Software
- Maximum Reusability of CSCI-Level End Products on Future Developments
- Maximum Reusability of Top-Level CSC End Products on Future Developments
- Maximum Reusability of Lower-Level CSC End Products on Future Developments
- Maximum Output Clarity
- Maximum Use of Off-the-Shelf Software
- Language/Tool/Method Evaluation

Ada Question	Ada Topic	ESD Form	ESD Question	ESD Topic
13.	Special Ada Features	[NONE]		
These data will show if any Ada subsets are used within the CSCI. This is important since many people believe that some features are too complex or inefficient.				
14.	Comments			

6.8 RESOURCE EXPENDITURE DATA CROSS-REFERENCE

The Resource Expenditure Data Form is unchanged from the ESD package.

6.9 COMPUTER SOFTWARE SIZE CROSS-REFERENCE

The ESD Computer Software Size Summary Data Form is not numbered, thus this cross-reference lists only the differences between the Ada form and the ESD form.

Ada Question	Ada Topic	ESD Form	ESD Question	ESD Topic
-----------------	--------------	-------------	-----------------	--------------

4.1 Size format CPCIFSD

The size format allows size specifications in several formats including:

- Source Lines of Code
- Carriage Returns
- Semicolons
- PDL Lines
- Ada Statements
- Other

This should help clarify the specific sizing data more completely.

4.2 CSC Size CPCIFSD

The following data are requested for each CSC:

- Total Size excluding Comments and Documentation
- Comments and Documentation
- Number of Machine words
- Size Reused
- Size to be Reusable
- Pre-existing Code Size
- Mods to Pre-existing
- Number of Function Points
- Number of Ada Objects
- Number of Packages
- Number of Tasks
- Number of Blocks
- Layers of Blocks
- Number of Ada Program Units
- Layers of Ada Program Units
- Number of Ada Statements
- Language

BIBLIOGRAPHY

Babich, Wayne A., "Productivity Issues in the Ada Language System," IEEE Computer Society, IEEE Computer Society Press, Silver Spring, Maryland, 1983.

The Ada Language System (ALS) developed by SofTech should increase productivity by minimizing error and eliminating unnecessary work. The environment is intended to decrease the effort required to track and organize the components of software under construction, and to minimize errors and regressions induced by mistakes in intra-team coordination.

Basili, Victor R., Katz, Elizabeth E., "Metrics of Interest in an Ada Development," IEEE Conference, IEEE Computer Society Press, Silver Spring, Maryland, August 1983.

Basili, Victor R., Katz, Elizabeth E., Panlilio-Yap, Nora Monina, Ramsey, Connie Loggia, and Chang, Shih, "Characterization of an Ada Software Development," IEEE Conference, IEEE Computer Society Press, Silver Spring, Maryland, September 1985.

Boehm, Barry W., Software Engineering Economics, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1981.

Boehm, B., "Software and Its Impact: A Quantitative Assessment," Writings of the Revolution, YOURDON inc., New York, New York, 1982.

There is more potential payoff in improving the efficiency of your analysis and validation efforts than in speeding up your coding.

More thorough analysis and design more than pays for itself in reduced testing costs.

Brooks, Fredrick P., Jr., The Mythical Man-Month, Addison Publishing Company, Inc., Philippines, 1975.

Buxton, J., "'Stoneman' Requirements for Ada Programming Support Environments," NTIS ADA-100404, February, 1980.

Buzzard, G. D. and Mudge, T. N., "Object-Based Computing and the Ada Programming Language," Computer, March 1985.

Chang, Shih-Chio and Yau, Stephen S., "Estimating Logical Stability in Software Maintenance," IEEE, IEEE Computer Society Press, Silver Spring, Maryland, 1984.

Addresses the subject of the logical ripple effect and a new approach for logical ripple effect analysis (determining the effect that finding one error in the source code will have).

Freedman, Roy S., Programming with APSE Software Tools, Petrocelli Books, Inc., Princeton, New Jersey, 1985.

Galorath, Daniel D., "Short and Long-term Ada Impacts," Proceedings of the International Society of Parametric Analysts, ISPA, May 1986.

German, Steven M., "Monitoring for Deadlock and Blocking in Ada Tasking," IEEE Transactions on Software Engineering, Volume SE-10, Number 6, IEEE Computer Society Press, Silver Spring, Maryland, November 1984.

Hamer, P. and Frewin, G. "Software metrics - a critical overview," The Software Development Process State of the Art Report, Pergamon Infotech Limited, Maidenhead, Berkshire, England, 1985.

Cost-to-fix errors increases more or less exponentially with the stage in the life-cycle at which it is fixed.

Helmbold D. and Luckham, D., "Debugging Ada Tasking Programs," IEEE Computer Society 1984 Conference on Ada Applications and Environments, IEEE Computer Society Press, Silver Spring, Maryland, October 1984.

Honeywell Inc., The Programming Language Ada Reference Manual, Springer-Verlag, Berlin, Heidelberg, New York, 1981.

Hummel, H., Nast, M., Uthke, E., "Training Concept for the Cost-Effective Development of Reliable Software Using the Programming Language Ada," Proceedings of the Third Joint Ada Europe/Ada TEC Conference, pp. 26-28, 1984.

"The possibility of realizing the full potential of Ada's modern features could be lost if trainees are allowed to gain the habit of using a subset of Ada in the style with which they are familiar in other languages."

The TYPE of training is VERY important. Depending on the type of software to be developed, different aspects of the language should be emphasized in the training course.

Jensen, Dr. Randall W., "Projected Productivity Impact of Near-term Ada Use in Software System Development," Proceedings of the International Society of Parametric Analysts, ISPA, May 1985.

Jones, Anita and Ardo, Anders, "Comparative Efficiency of Different Implementations of the Ada Rendezvous," Proceedings of the AdaTEC Conference on Ada, ACM, 1982.

Klumpp, Allan R., "Space Station Flight Software: Hal/S or Ada?," Computer, March 1985.

Litvintchouk, Steven D., Matsumoto, Allen S., "Design of Ada Systems Yielding Reusable Components: An Approach Using Structured Algebraic Specification," IEEE Transactions on Software Engineering, Volume SE-10, Number 5, IEEE Computer Society Press, Silver Spring, Maryland, September 1984.

Meeson, R. N., Jr., "Function-Level Programming in Ada," IEEE Computer Society 1984 Conference on Ada Applications and Environments, October 1984.

Narfelt, Kjell-Hakan and Schefstrom, Dick, "Towards a KAPSE Database," IEEE Conference on Ada Applications and Environments, IEEE Computer Society Press, Silver Spring, Maryland, 1984.

Newell, A., "Programmer Productivity," The Software Development Process State of the Art Report, Pergamon Infotech Limited, Maidenhead, Berkshire, England, 1985.

It is often said that a few hours of analysis can save hundreds of hours of programming (and reprogramming), and there is no doubt that productivity in testing will be one result of good analysis and design.

Programmers in Ada should be able to become largely independent of the "target" environment.

The additional time for analysis and design can completely absorb the savings in programming.

Organick, E. I., Carter, T. M., Maloney, M. P., Davis, A., Hayes, A. B., Klass, D., Lindstrom, G., Nelson, B. E., and Smith, K. F., "Transforming an Ada Program Unit to Silicon and Verifying Its Behavior in an Ada Environment: A First Experiment," IEEE Software, IEEE Computer Society Press, Silver Spring, Maryland, January 1984.

Peters, Lawrence J., Software Design: Methods & Techniques, YOURDON inc., New York, New York, 1981.

Privitera, J. P., "Ada Design Language for the Structured Design Methodology," Proceedings of the AdaTEC Conference on Ada, ACM, 1982.

Roy, D., "SEL Workshop 86 paper," Proceedings of the Tenth Annual Software Engineering Workshop, December 1985.

The quality of the development environment significantly impacts software development productivity.

Even with the features of Ada, it is possible to develop poor software. The features will have to be closely controlled by competent project managers because these features are powerful, hence dangerous. Moreover, those powerful features provide another dimension of design decision. We feel that a methodology that helps the software engineer allocate function and data structures to packages and tasks is necessary.

We found that Ada is sufficiently complex, that we kept learning throughout the pilot project, and even beyond. We also found that none of the standard training devices (seminars, books, computer-aided instruction) could alone address the broad range of issues that really are at the heart of the problem.

In the Ada era, a comprehensive education in the software engineering principles that form the basis of the Ada culture must replace ad-hoc training in the syntactic recipes of a language.

That is why we recommend a variety of continuous education measures in our report: Assuming adequate familiarity with modern software engineering practices, at least 4 person-weeks is the minimum training time. This time includes teaching a methodology adapted to Ada and 50 percent hands-on experiments under the supervision of an expert.

Ada should prove to be an excellent tool in the hands of competent and properly trained software developers. It will not be a panacea, compensating for inadequate methods or training, but it will be beneficial if properly applied.

There will be major difficulties at BOTH ends of the programmer competency scale. Many of the brightest programmers will tend to produce overly complex designs, using every possible feature of the language; the application itself becoming a side issue, many of the less competent programmers will never really understand the Ada technology.

Saib, Sabina, Ada: an introduction, Holt, Rinehart, Winston, New York, New York, 1985.

Tichy, Walter F., "Adabase: A Data Base for Ada Programs," Proceedings of the AdaTEC Conference on Ada, ACM, 1982.

Urban, Joseph E., Fisher, David A., "Ada Environments and Tools," IEEE Software, IEEE Computer Society Press, Silver Spring, Maryland, March 1985.

Van Der Linden, P., "Experience with Ada," Software World, Volume 15, Number 2, 1984.

For successful use of Ada, programmers MUST be educated.

Van der Linden, P., "Looking Forward With Ada," ACM Ada Letters, Volume V, Number 1, July, August 1985.

Ada is a management problem.

Early compilers may emphasize "passing" validation, more than trying to be useful or optimizing.

Wallis, P. J. L., "Economic factors in Software Production," The Software Development Process State of the Art Report, Pergamon Infotech Limited, Maidenhead, Berkshire, England, 1985.

Software could be better if its development did not depend on highly skilled manpower.

It is the need to rework development based on faulty design decisions which reduces productivity.

Software tools should make software development more cost-effective. Skilled programmers are a scarce resource and will continue to be so. Development techniques which provide a path away from today's labor-intensive methods will permit levels of production control and documentation adequate to the development of truly reusable software.

Ada should help prevent some of the interface errors.

Whitaker, Col. W. A., "Three Ada Examples," Digest of Papers, IEEE COMPCON San Francisco, IEEE Computer Society Press, Silver Spring, Maryland, Spring 1983.

This is an older article (1983). The author felt that there was "no insurmountable training problem at the programmer level." He felt translation of existing programs was an excellent way to bring programmers up to speed.

The more mathematical functions in the program, the easier it should be to code in Ada, because the mathematical portion of Ada is the closest to the other languages. His concluding question was, "Ada makes a lot of things possible, but can we make them happen?"

Wolf, Alexander L., Clarke, Lori A., and Wileden, Jack C., "An Ada Environment for Programming-in-the-large," IEEE Conference on Ada Applications and Environments, IEEE Computer Society Press, Silver Spring, Maryland, 1984.

Wolf, Alexander L., Clarke, Lori A., and Wileden, Jack C., "Ada-Based Support for Programming-in-the-Large", IEEE Software, IEEE Computer Society Press, Silver Spring, Maryland, March 1985.

APPENDIX A
QUESTIONNAIRE

APPENDIX A

QUESTIONNAIRE

The following is the base set of questions used during interviews of software modelers and Ada experts.

General Questions

1. In general, how do you feel Ada will impact the cost of a software development project? Please consider small and large projects and short- and long-term impacts.
2. Are you familiar with any Ada projects in progress or which have already been completed? If yes, in what way was the use of Ada a positive or negative experience?
3. Does the type of project (mission critical vs. commercial, etc.) affect the impact of the use of Ada or the APSE? If so, to what degree?
4. Can the size (lines of source code) of an Ada project be estimated as well as the size of previous projects in other languages and environments (i.e., Fortran, Jovial, etc.)?
5. Can standard expansion ratios for machine to source instructions be used to predict source size?
6. What will be the different impacts of Ada as a language versus Ada as a development environment (APSE)?
7. What trends do you expect over the next 5 years for Ada projects? In specific, productivity, maintenance costs, errors, or any area you feel is significant.
8. The following have been advanced as design considerations for Ada. Would you comment on the ability of Ada or the APSE to address these concerns:

- Life-cycle support cost
- Interface control
- Analysis support
- Version control
- Management support
- Multi-tasking provisions
- Method independence (top down vs. bottom up, etc.)
- Maintenance
- Reliability
- Readability
- Transport across projects and computers

9. It has been said that, at first, Ada projects will be staffed by the best people and be more closely monitored, which may skew initial results. Do you agree with this statement and, if so, what differences do you expect between "real" Ada projects and "trial" Ada projects?
10. Will Ada ease re-hosting cost? Why?
11. How do you see Ada being applied? A full implementation vs. subsets, at the KAPSE, MAPSE, or APSE levels.
12. How will Ada affect configuration management?
13. What data do you feel should be collected to help determine the cost of an Ada project?
14. Do you think that all or any one of the Software development cost and schedule estimation models currently in use can accurately estimate the cost of an Ada project? If so why? If not, why not?
15. Will effort and time be impacted equally by the use of Ada or the APSE? If not, will there be a relationship between the impact on time and the impact on effort?
16. What will be the impact of requiring software to be reusable?

Structured Design (Object-oriented)

1. Compared to the use of other languages and environments, what would the cost of Ada be if structured designs were NOT used at the beginning of the project?
2. How important to Ada and APSE use is past experience with Structured Methods?
3. Would a poor quality specification cost more in Ada than in another language?
4. What will be the cost and schedule impact on the requirements specification, preliminary design, detailed design, code and unit testing, software integration and systems integration when Ada or the APSE is used?
5. Is the waterfall model of the software life-cycle applicable to Ada and the APSE?

Staffing

1. What will be the effect of adding more people to an Ada project? Is there a point where additional staffing is ineffective?

2. Will Ada allow different staffing profiles (will the ability to develop in parallel, if it exists, allow more effort with less schedule)?
3. In determining the cost of an Ada project, how can or should experience and training in Ada be measured?

PDL Questions

1. Can Ada be used as a preliminary design specification tool?
2. How will using or NOT using Ada as the PDL for the Project impact the cost?
3. When should Ada be applied to the Project?
4. Does Ada, as a design language, require any design aids? If so, what aids? (Data flow diagrams, etc.)
5. Will the process of refinement change Ada designs? Would it simply be a matter of filling in more and more details, or are structural changes likely to occur?
6. Will Ada as a PDL be usable for the public portions of packages, or will refinement force changes in the PDL?

Programmer Questions

1. How much more would a "first" Ada project cost as opposed to subsequent Ada projects?
2. How will Ada and the APSE affect programmer productivity? Please consider the short- and long-term effects and the overall cost of the project.
3. Will programmer portability be affected by the use of Ada or the APSE?
4. Will Ada affect the management of a project? If so, how will this impact the cost?

Debugging Questions

1. What, if any, new and different program errors may the use of Ada and the APSE cause? Please state the nature of these errors and where they will be experienced.
2. How will this affect cost and productivity?
3. Will the use of Ada and the APSE change where errors are located in the life cycle?

Environment Questions

1. Even though Ada environments are intended to be portable, will the implementation used affect Project cost (i.e., one compiler vs. another, etc.) (Ease of use - Turnaround)?
2. Is the Ada desire for a standardized tool package realistic and, if so, how will it impact costs?

Language Questions

1. What special features of Ada do you feel will have cost impacts? (i.e., generic functions, overloaded operators, packages, and any others which you may feel would impact the cost)
2. When converting an existing project to Ada, what impact will various source languages have and how may it be measured? (i.e., Jovial J-3, Jovial J-73, Fortran, CMS-2, COBOL, TACPOL, SPL/1)

Summary Questions

1. What other factors may impact the cost of an Ada project that we haven't covered?
2. What specific pieces of data do you need that you don't have now?
3. What specific pieces of data are superfluous to your analysis?
4. The following are factors which have been used in various cost models (Boehm, 1981). Please indicate:
 - a. Which do you feel will have the greatest impact on the cost of an Ada project?
 - b. Which do you feel will have the least impact on the cost of an Ada project?

Size attributes

Source instructions
Object instructions
Number of routines
Number of data items
Number of output formats
Documentation
Number of personnel

Program attributes

- Type
- Complexity
- Language
- Reuse
- Required reliability

Computer attributes

- Time constraint
- Storage constraint
- Hardware configuration
- Concurrent hardware development

Personnel attributes

- Personnel capability
- Personnel continuity
- Hardware experience
- Applications experience
- Language experience

Project attributes

- Tools and techniques
- Customer interface
- Requirements definition
- Requirements volatility
- Schedule
- Security
- Computer Access
- Travel/rehosting

Candidate Projects For Data Collection

1. What projects can you recommend for actual data collection during the next contract phase?

APPENDIX B
RESPONDENTS

The names of the respondents have been removed to ensure that the interviews distributed as part of this report cannot be attributed. The respondent list has been provided to ESD/ACCR under separate cover.

APPENDIX C
DATA COLLECTION FORMS AND INSTRUCTIONS

ESD ADA^{*} SOFTWARE DEVELOPMENT DATA COLLECTION FORMS

- PROJECT SUMMARY DATA
- SYSTEM LEVEL DOCUMENTATION DATA
- DEVELOPMENT COMPUTER SYSTEM AND TOOLS DATA
- TARGET COMPUTER SYSTEM DATA
- COMPUTER SOFTWARE CONFIGURATION ITEM SUMMARY DATA
- CSCI LEVEL DOCUMENTATION DATA
- COMPUTER SOFTWARE SIZE DATA
- RESOURCE EXPENDITURE DATA

^{*} ADA IS A REGISTERED TRADEMARK OF THE UNITED STATES GOVERNMENT (ADA JOINT PROGRAM OFFICE)

SOFTWARE DEVELOPMENT PROJECT SUMMARY DATA FORM

1. Project Name: _____ Date: _____

2. Development Contractor / Organization: _____

3. Project Description

3.1. Mission description: _____

3.2. Major Hardware Interfaces: _____

3.3. Major System Functions: _____

3.4. Major Software Functions: _____

3.5. Number of CSCIs: _____

3.6. Computer Software Configuration Item (CSCI) Names:

_____	_____	_____	_____
_____	_____	_____	_____
_____	_____	_____	_____
_____	_____	_____	_____

3.7. System User:

Development Contractor [] Other Commercial Company []
Department of Defense [] Other Government Agency []

3.8. Relative Magnitude of the Software Effort

Percent of the entire hardware/software system development cost
allocated to software _____

3.9. List the Software Development Standards that apply to this
development. (They are often listed in the contract or CPDP.)

_____	_____	_____	_____
_____	_____	_____	_____
_____	_____	_____	_____

SOFTWARE DEVELOPMENT PROJECT SUMMARY DATA FORM

4. Project Milestones:

Milestone	Contract Date	Estimate Date	Actual Date
4.1. Contract Award	_____	_____	_____
4.2. System Requirements Review (SRR)	_____	_____	_____
4.3. System Design Review (SDR)	_____	_____	_____
4.4. System Preliminary Design Review (PDR)	_____	_____	_____
4.5. System Critical Design Review (CDR)	_____	_____	_____
4.6. Start CSCI Integration into System	_____	_____	_____
4.7. Complete CSCI Integration	_____	_____	_____
4.8. Start Dev. Test & Evaluation	_____	_____	_____
4.9. Complete Dev. Test & Evaluation	_____	_____	_____
4.10. Start Init Operational Test & Evaluation ...	_____	_____	_____
4.11. Complete Init Operational Test & Evaluation	_____	_____	_____
4.12. Functional Configuration Audit	_____	_____	_____
4.13. Physical Configuration Audit	_____	_____	_____
4.14. Formal Qualification Review	_____	_____	_____
4.15. System Delivery	_____	_____	_____

5. Modern Development Method Use

(0 = No Use; 1 = Beginning, Experimental Use; 3 = Reasonably Experienced; 5 = Expert)

Method	0	1	2	3	4	5
	No	VLo	Lo	Nom	Hi	Vhi
Structured Requirements Analysis	[]	[]	[]	[]	[]	[]
Specification Type						
Functional	[]	[]	[]	[]	[]	[]
Procedural	[]	[]	[]	[]	[]	[]
English	[]	[]	[]	[]	[]	[]
Other	[]	[]	[]	[]	[]	[]
Design						
Top Down	[]	[]	[]	[]	[]	[]
Bottom Up	[]	[]	[]	[]	[]	[]
Object Oriented	[]	[]	[]	[]	[]	[]
Iterative Enhancement	[]	[]	[]	[]	[]	[]
Hardest First	[]	[]	[]	[]	[]	[]
Other	[]	[]	[]	[]	[]	[]
None _____						
Program Design Language (PDL)						
Ada	[]	[]	[]	[]	[]	[]
Conventional	[]	[]	[]	[]	[]	[]
Development						
Top Down	[]	[]	[]	[]	[]	[]
Bottom Up	[]	[]	[]	[]	[]	[]
Iterative Enhancement	[]	[]	[]	[]	[]	[]
Hardest First	[]	[]	[]	[]	[]	[]
Other	[]	[]	[]	[]	[]	[]
None						

SOFTWARE DEVELOPMENT PROJECT SUMMARY DATA FORM

0 1 2 3 4 5
No VLo Lo Nom Hi Vhi

Coding

Structured Code [] [] [] [] [] []
Other [] [] [] [] [] []
None _____

Program Librarian [] [] [] [] [] []

Team Development Strategies

Two-person [] [] [] [] [] []
Multi-person Democratic [] [] [] [] [] []
Chief Programmer [] [] [] [] [] []
Other [] [] [] [] [] []

Testing

Specification Driven [] [] [] [] [] []
Structure Driven [] [] [] [] [] []
Other [] [] [] [] [] []
None _____

Validation/Verification (Inspection)

Walk-Throughs [] [] [] [] [] []
Proof of Correctness [] [] [] [] [] []
Other [] [] [] [] [] []
None _____

Project Estimating and Control [] [] [] [] [] []

Rapid Prototyping [] [] [] [] [] []

[] [] [] [] [] []

6. Software Quality Required

0 1 2 3 4 5
No VLo Lo Nom Hi Vhi

6.1. Usability [] [] [] [] [] []
6.2. Reliability [] [] [] [] [] []
6.3. Efficiency [] [] [] [] [] []
6.4. Integrity [] [] [] [] [] []
6.5. Testability [] [] [] [] [] []
6.6. Portability [] [] [] [] [] []
6.7. Correctness [] [] [] [] [] []
6.8. Maintainability [] [] [] [] [] []
6.9. Reusability [] [] [] [] [] []
6.10. Interoperability [] [] [] [] [] []

7. Software Change History

Number Est. Est.
Changes DSLOC Personnel
Approved

- 1.1 Preliminary Design (C A to PDR)
- 1.2 Detailed Design (PDR to CDR)
- 1.3 Code & Debug (CDR to Start Test & Integr)
- 1.4 Test & Integration (Start T&I to EUT)
- 1.5 System Test (EUT to Contract End)

SOFTWARE DEVELOPMENT PROJECT SUMMARY DATA FORM

8. Comments:

DEVELOPMENT COMPUTER SYSTEM AND TOOLS DATA FORM

1. Development Contractor / Organization

1.1. Name: _____

1.2. Location: _____

2. Project Name: _____

3. Development System Attributes

3.1. Development Computer Mfg _____ Model _____

3.1.1. Main Memory Size (Words) _____

3.1.2. Word Size (Bytes) _____

3.1.3. Maximum Main Memory Size (Words) _____

3.1.4. Is this a Virtual Memory machine (Yes or No) _____

3.1.5. CPU Processing Speed (Mips) _____

3.1.6. Average number of people per terminal _____

3.2. Percentage of Source Instructions developed using each of the following access modes (Total=100%):

3.2.1. Batch _____ %

3.2.2. Dedicated Processor _____ %

3.2.3. Test Bed with High Priority _____ %

3.2.4. Test Bed with Low Priority _____ %

3.2.5. Interactive _____ %

3.2.6. Other: _____ %

3.3. Software Development System

Rating	Estimate			Actual
	Min	Most Likely	Max	
3.3.1. Turnaround Time No Recompile (hours)	_____	_____	_____	_____
3.3.2. Turnaround Time Recompile Required (hours).	_____	_____	_____	_____
3.3.3. Terminal Response (seconds)	_____	_____	_____	_____
Variable _____ or Consistent _____				
3.3.4. Number of Major Changes per Month	_____	_____	_____	_____
3.3.5. Number of Minor Changes per Month.....	_____	_____	_____	_____

3.4. Hours the Development System Operates _____

3.5. Percent of Development System Operation Hours Available to Development Organization _____

3.6. Development System Security Classification Level _____

DEVELOPMENT COMPUTER SYSTEM AND TOOLS DATA FORM

4. Development Computer System Tools (See instructions for rating scale definitions.)

	Manual Usage		Automated Usage		(1-5) Usage Freq.		(0-5) Tool Experience		(1-8) When Used		CSCIs Affected	Tool Name
Text editor												
Screen editor												
Line editor												
Syntax-directed editor ..												
Graphics editor												
Compiler code generator ..												
Compiler options												
Intermediate language generator												
Graphics generator												
Assembler												
Assembler options												
Linker												
Debugger												
Interactive debugger												
Symbolic debugger												
Data flow tracer												
Control flow tracer												
Execution timer												
Execution tracer												
Program tuner												
Interpreter												
Command language processor												
Code interface analyzer												
Code invocation analyzer												
Structure Checker												
Data type analyzer												
Code data flow analyzer												
Coverage/frequency analyzer												
Requirements language processor												
Requirements analyzer ...												
Requirements documentation generator												
Program design language ..												
Design language processor												
Design analyzer												
Design documentation generator												
Requirements prototyper ..												
Design prototyper												
Instruction set simulator												
Run-time environment simulator												

DEVELOPMENT COMPUTER SYSTEM AND TOOLS DATA FORM

Program (application)

Generator
 Test harness generator
 Body stub generator
 Code auditor
 Physical units analyzer
 Executable assertion
 checker
 Problem report analyzer
 Change request analyzer
 On-line assistance
 processor
 Cross reference
 Scanner
 Quality analyzer
 Sorter/merger
 Test condition analyzer
 I/O specification
 analyzer
 Constraint evaluator
 Regression tester
 Symbolic executor
 Formal verifier
 Change impact analyzer
 Statement profiler
 Memory dump
 Source converter
 Macro expander
 Formatter

Other Tools and Methodologies (specify)

Configuration Management Tools and Methodologies (specify)

Program management support tools (specify)

Program development (specify)

DEVELOPMENT COMPUTER SYSTEM AND TOOLS DATA FORM

5. Development Locations

5.1. Number of Development Sites _____

5.2. Development Computer and Site Locations

Site Locations

Computer Locations

6. List the CSCIs Developed on this Development Computer:

1. _____	2. _____	3. _____
4. _____	5. _____	6. _____
7. _____	7. _____	8. _____
10. _____	11. _____	12. _____

7. Comments

TARGET COMPUTER SYSTEM DATA FORM

1. Development Contractor / Organization

1.1. Name: _____

1.2. Location: _____

2. Project Name: _____

3. Target System Attributes

3.1. Target Computer Manufacturer _____ Model _____

3.1.1. Main Memory Size (Words) _____

3.1.2. Word Size (Bytes) _____

3.1.3. Number of Processors in the Target _____

3.1.4. Maximum Main Memory Size (Words) _____

3.1.5. Is this a Virtual Memory machine? (Yes or No) _____

3.1.6. CPU Processing Speed (Mips) _____

3.1.7. Reserve Memory Requirement (percent) _____

3.1.8. Reserve Timing Requirement (percent) _____

3.1.9. Programming Language(s) Implementation
(if different than the development computer) _____

3.1.10. Difference Between Development and Target Computer: _____

3.1.11. Accessibility to Target
(limited to freely accessible)..... _____

3.1.12. Target simulator Size, in words (if needed) _____

4. Changes to the Target Computer System During Software Development

Rating	Estimate			Actual
	Min	Most Likely	Max	

4.2.1. Number of Major Changes per Month _____

4.2.2. Number of Minor Changes per Month..... _____

4.3. Concurrent Hardware Development With the Software? Yes[] No[]

5. List the CSCIs Executed on the Target Computer:

1. _____	2. _____	3. _____
4. _____	5. _____	6. _____
7. _____	8. _____	9. _____
10. _____	11. _____	12. _____

Comments about the Target Computer, its Development, Use, Availability,
or other important points:

COMPUTER SOFTWARE CONFIGURATION ITEM SUMMARY DATA FORM

1. CSCI Identification

1.1. CSCI Name: _____ Date: _____

1.2. Development Contractor / Organization: _____

1.3. Project Name: _____

2. CSCI Functional Description: _____

2.1. Operating Environment

Production Center, Internally Developed []	Military Ground []
Production Center, Contracted Software []	Unmanned Space []
Military Mobile (Van or Shipboard) []	Manned Space []
Commercial Avionics []	Mil-Spec Avionics []
Other _____	

3. CSCI Schedule Data

3.1 Milestones

Milestone	Contract Date	Estimate Date	Actual Date
3.1.1. CSCI Development Start (Draft B5)	_____	_____	_____
3.1.2. S/W Specification Review (SSR)	_____	_____	_____
3.1.3. Preliminary Design Review (PDR)	_____	_____	_____
3.1.4. Critical Design Review (CDR)	_____	_____	_____
3.1.5. Start Coding/Debugging	_____	_____	_____
3.1.6. Complete Coding/Debugging	_____	_____	_____
3.1.7. Start Informal Test/Integration	_____	_____	_____
3.1.8. End Informal Test/Integration	_____	_____	_____
3.1.9. Preliminary Qualification Test (PQT)	_____	_____	_____
3.1.10. Test Readiness Review (TRR)	_____	_____	_____
3.1.11. Formal Qualification Test (FQT)	_____	_____	_____
3.1.12. Product Specification Approval	_____	_____	_____
3.1.13. Functional Configuration Audit (FCA)	_____	_____	_____
3.1.14. Physical Configuration Audit (PCA)	_____	_____	_____
Other.....	_____	_____	_____

3.2. Schedule Acceleration/Stretchout Assessment:

<75% []	75-85% []	86-95% []	96-105% []	106-115% []
	116-125% []	126-135% []	135-160% []	>160% []

COMPUTER SOFTWARE DEVELOPMENT IN THE SUMMARY DATA FORM

1. Title

2. Project Name

3. Project Number

4. Project Description

5. Test Program Name

6. Test Program Number

7. Test Program Description

8. Application Area

9. Project Status

10. Average Rate of Change

11.1 Formula for Rate of Change

11.2 Formula for Rate of Change

11.3 Formula for Rate of Change

11.4 Formula for Rate of Change

12. Peak Designer

13. Peak Programmer

14. Peak Tester

15. Maximum Staffing Rate

16. Overall % of Requirements

17. Reliability Requirements

Required Reliability

18. Complexity

18.1 Inherent Complexity of Application

18.2 Inherent Complexity of Data Structures

18.3 System Integration and Test

Simple [] Routine [] Difficult [] Complex []

COMPUTER SOFTWARE CONFIGURATION ITEM SUMMARY DATA FORM

Project: _____

System: _____ Integrated _____ (Item #) Size Information

Language: _____ (Does line 1 generate object code) []
 Page Returns [] Semiblocks [] PDL Lines []
 Other Specifications _____

Estimate Date: _____

	Initial Estimate			Current Estimate			
	Min	Most Likely	Max	Min	Most Likely	Max	Actual

Total Size _____

Excluding _____

Documentation _____

7.1.2 Documentation _____

7.1.3 Operational Response Requirements (% of 7.1.1.1.; Total = 100%)

Real Time _____ %	On-Line _____ %
Time Constrained _____ %	Non-Time-Critical _____ %

7.1.4 Source Statement Mix

7.1.4.1 Statement Types (% of 7.1.1.1.; Total = 100%)

Original _____ %	Command _____ %	Mathematical _____ %
Data Manipulation _____ %	Data Typing _____ %	Declaration _____ %
Ada Tasking _____ %	Invocation _____ %	

7.1.4.2 1961 Source Code Mix (Total Code = 100%)

Source Code Type	% Code	% New Design	% New Code
Operating Systems _____	_____	_____	_____
Interactive Operations _____	_____	_____	_____
Real Time Command & Control _____	_____	_____	_____
On-Line Communications _____	_____	_____	_____
Data Storage & Retrieval _____	_____	_____	_____
String Manipulation _____	_____	_____	_____
Mathematical Operations _____	_____	_____	_____
Other _____	_____	_____	_____
Other _____	_____	_____	_____

COMPUTER SOFTWARE CONFIGURATION ITEM SUMMARY DATA FORM

7.1.4. Target Computer Impact on Source Code

Rating	Estimate			Actual
	Min	Most Likely	Max	
7.1.4.1. Memory Constraint Percent	_____	_____	_____	_____
7.1.4.2. CPU Time Constraint Percent	_____	_____	_____	_____
7.1.4.3. Real-Time Operation Percent	_____	_____	_____	_____
7.1.4.4. Multi-processor Percent	_____	_____	_____	_____
7.1.4.5. Multi-target Percent	_____	_____	_____	_____

7.1.5. CSCI Reused Code From Other Projects

Size Component	Estimated	Actual
7.1.5.1. Total Pre-Existing	_____	_____
7.1.5.2. Total Deleted	_____	_____
7.1.5.3. Total Modified	_____	_____
7.1.5.4. Percent Re-Design Effort	_____	_____
7.1.5.5. Percent Re-Implementation Effort	_____	_____
7.1.5.6. Percent Re-Test Effort	_____	_____
7.1.5.7. List of projects which contained the re-usable code: _____	_____	_____

7.2. Function Point Data (if automated counting tool is available)

Parameter	Estimated	Actual
7.2.1. Number of Inputs	_____	_____
7.2.2. Number of Outputs	_____	_____
7.2.3. Number of Inquiries	_____	_____
7.2.4. Number of Data Files	_____	_____
7.2.5. Number of Interfaces	_____	_____
7.2.6. Total Number of Function Points	_____	_____

7.3. Size/Complexity Data (if automated counting tool is available)

7.3.1. Number of Ada Objects	_____	_____
7.3.2. Number of Ada Program Units	_____	_____
7.3.3. Number of Layers of Ada Program Units	_____	_____
7.3.4. Number of Blocks	_____	_____

8. Data Base Size (if automated counting tool is available)

Parameter	Estimated	Actual
8.1. Total Data Base Size (Words)	_____	_____
8.2. Total Unique Data Items.....	_____	_____
8.3. Total Number of Records	_____	_____
8.4. Unique Data Types	_____	_____

COMPUTER SOFTWARE CONFIGURATION ITEM SUMMARY DATA FORM

9. Special Display Requirements (Check 1)

Simple Inputs/Outputs _____
 User Friendly Error Recovery and Menus _____
 Interactive (With Pointing Device) _____
 Complex (Such as Cad/Cam) _____
 Other (Indicate) _____

10. Software Failure History (Errors Per Phase)

	Req'mnts Errors	Design Errors	Implement Errors	Total Errors
10.1. Preliminary Design (C/A to PDR)....	_____	_____	_____	_____
10.2. Detailed Design (PDR to CDR).....	_____	_____	_____	_____
10.3. Code & Debug (CDR to Start T & I)..	_____	_____	_____	_____
10.4. Test & Integration (Start T&I-FQT).	_____	_____	_____	_____
10.5. System Test/IOC (FQT to Cont. End).	_____	_____	_____	_____

11. Software Change History

	Number Changes Approved	Est. DSLOC +/-	Est. Personnel +/-
11.1. Preliminary Design (C/A to PDR).....	_____	_____	_____
11.2. Detailed Design (PDR to CDR).....	_____	_____	_____
11.3. Code & Debug (CDR to Start Test & Integ)...	_____	_____	_____
11.4. Test & Integration (Start T&I to FQT).....	_____	_____	_____
11.5. System Test/IOC (FQT to Contract End).....	_____	_____	_____

12. CSCI Development Attributes

12.1. Development Environment

Rating	Estimate			Actual
	Min Likely	Most Likely	Max	
12.1.1. Resource Dedication (percent)	_____	_____	_____	_____
12.1.2. Resource/Support Location (miles)	_____	_____	_____	_____
12.1.3. Security Level of CSCI	_____			
12.1.4. Contract Type	_____			
CPPP _____ CPIF _____ FFP _____ FPIF _____ OTHER _____				
Prime Contract _____ Subcontract _____				

COMPUTER SOFTWARE CONFIGURATION ITEM SUMMARY DATA FORM

12.2. Specific Development Goals (If They Impact Development)
(Rating Scale: No = No Emphasis Through 7hi = Critical Goal)

Goal	1 2 3 4 5 6 7						
	No	Wlo	Lo	Nom	Hi	Whi	
12.2.1. Maximum Maintainability	[]	[]	[]	[]	[]	[]	[]
12.2.2. Maximum Reuse of Pre-Existing S/W	[]	[]	[]	[]	[]	[]	[]
12.2.3. Maximum Reusability of CSCI Level End Products On Future Developments.....	[]	[]	[]	[]	[]	[]	[]
12.2.4. Maximum Reusability of TLCSC End Products On Future Developments	[]	[]	[]	[]	[]	[]	[]
12.2.5. Maximum Reusability of LLCSC End Products On Future Developments	[]	[]	[]	[]	[]	[]	[]
12.2.6. Maximum Output Clarity	[]	[]	[]	[]	[]	[]	[]
12.2.7. Maximum Use of Off the Shelf S/W.....	[]	[]	[]	[]	[]	[]	[]
12.2.8. Language/Tool/Method Evaluation	[]	[]	[]	[]	[]	[]	[]
Other.....	[]	[]	[]	[]	[]	[]	[]

13. Special Ada Features

Feature List	Avoided	Used	If Used, is there an Internal Standard? (Y or N)
Ada Tasking	_____	_____	_____
Derived Types	_____	_____	_____
Generics	_____	_____	_____
Overloaded Operators	_____	_____	_____
Other _____	_____	_____	_____
Other _____	_____	_____	_____

14. Special Problems or Comments:

SYSTEM LEVEL _____ or CSCI LEVEL _____ DOCUMENTATION FORM

1. Project Name:	2. Development Contractor / Organization:	Date:	3. CSCI Name if CSCI Level:	4. Document:	GPD or Provided	Written by Developer	Quality 1 - 5	Date Completed	Entered Number of Pages	Actual Number of Pages
				System/Segment Specification						
				Software Configuration Management Plan						
				Software Quality Evaluation Plan						
				Software Standards and Procedures Manual						
				Preliminary Operational Concept Document						
				Preliminary Software Requirements Specification						
				Preliminary Interface Requirements Specification						
				Interface Control Document						
				Software Top-Level Design Document						
				Version Description Document						
				Software Test Plan						
				Software Test Description						
				Software Test Procedure						
				Software Test Report						
				Computer System Operator's Manual						
				Computer User's Manual						
				Computer System Diagnostic Manual						
				Software Programmer's Manual						
				Program Maintenance Manual						
				Firmware Support Manual						
				Operational Concept Document						
				Computer Resources Integrated Support Document						
				Software Requirements Specification						
				Interface Requirements Specification						
				Interface Design Document						
				Data Base Design Document						
				Software Product Specification						
				Software Development Plan						
				Software Detailed Design Document						
				System/Subsystem Specification						
				Functional Description						
				Database Specification						
				Interface Specification						
				User's Manual						
				Program Specification						
				other						
				other						
				other						
				other						

COMPUTER SOFTWARE SIZE SUMMARY DATA FORM

1. Title _____ Date: _____

2. Department _____ Organization _____

3. Project Name _____

4. Software Description

4.1. Size Format: Source line of code (generates object code) []
 Carriage Returns [] Semicolons [] PDL Lines []
 Other (Specify) _____

4.2. CSC Size (or CSC If CSC Not Available)

CSC Name/Function _____

Total size excluding Comments and Documentation		_____
Comments and Doc.	Number of Machine Words	_____
Size Reused	Size to be Reusable	_____
Pre-existing Code Size	Meds to Pre-existing	_____
# of Function Points	Number of Ada Objects	_____
Number of Packages	Number of Tasks	_____
Number of Blocks	Layers of Blocks	_____
# of Ada Program Units	Layers Ada Program Units	_____
Number of Ada Statements	Language	_____
The values for this CSC are: Estimates		Actuals _____

CSC Name/Function _____

Total size excluding Comments and Documentation		_____
Comments and Doc.	Number of Machine Words	_____
Size Reused	Size to be Reusable	_____
Pre-existing Code Size	Meds to Pre-existing	_____
# of Function Points	Number of Ada Objects	_____
Number of Packages	Number of Tasks	_____
Number of Blocks	Layers of Blocks	_____
# of Ada Program Units	Layers Ada Program Units	_____
Number of Ada Statements	Language	_____
The values for this CSC are: Estimates		Actuals _____

CSC Name/Function _____

Total size excluding Comments and Documentation		_____
Comments and Doc.	Number of Machine Words	_____
Size Reused	Size to be Reusable	_____
Pre-existing Code Size	Meds to Pre-existing	_____
# of Function Points	Number of Ada Objects	_____
Number of Packages	Number of Tasks	_____
Number of Blocks	Layers of Blocks	_____
# of Ada Program Units	Layers Ada Program Units	_____
Number of Ada Statements	Language	_____
The values for this CSC are: Estimates		Actuals _____

COMPUTER SOFTWARE SIZE SUMMARY DATA FORM

CSC Name/Function _____

Total size excluding Comments and Documentation _____	
Comments and Doc. _____	Number of Machine Words _____
Size Reused _____	Size to be Reusable _____
Pre-existing Code Size _____	Mods to Pre-existing _____
# of Function Points _____	Number of Ada Objects _____
Number of Packages _____	Number of Tasks _____
Number of Blocks _____	Layers of Blocks _____
# of Ada Program Units _____	Layers Ada Program Units _____
Number of Ada Statements _____	Language _____
The values for this CSC are: Estimates _____ Actuals _____	

CSC Name/Function _____

Total size excluding Comments and Documentation _____	
Comments and Doc. _____	Number of Machine Words _____
Size Reused _____	Size to be Reusable _____
Pre-existing Code Size _____	Mods to Pre-existing _____
# of Function Points _____	Number of Ada Objects _____
Number of Packages _____	Number of Tasks _____
Number of Blocks _____	Layers of Blocks _____
# of Ada Program Units _____	Layers Ada Program Units _____
Number of Ada Statements _____	Language _____
The values for this CSC are: Estimates _____ Actuals _____	

CSC Name/Function _____

Total size excluding Comments and Documentation _____	
Comments and Doc. _____	Number of Machine Words _____
Size Reused _____	Size to be Reusable _____
Pre-existing Code Size _____	Mods to Pre-existing _____
# of Function Points _____	Number of Ada Objects _____
Number of Packages _____	Number of Tasks _____
Number of Blocks _____	Layers of Blocks _____
# of Ada Program Units _____	Layers Ada Program Units _____
Number of Ada Statements _____	Language _____
The values for this CSC are: Estimates _____ Actuals _____	

CSC Name/Function _____

Total size excluding Comments and Documentation _____	
Comments and Doc. _____	Number of Machine Words _____
Size Reused _____	Size to be Reusable _____
Pre-existing Code Size _____	Mods to Pre-existing _____
# of Function Points _____	Number of Ada Objects _____
Number of Packages _____	Number of Tasks _____
Number of Blocks _____	Layers of Blocks _____
# of Ada Program Units _____	Layers Ada Program Units _____
Number of Ada Statements _____	Language _____
The values for this CSC are: Estimates _____ Actuals _____	

COMPUTER SOFTWARE SIZE SUMMARY DATA FORM

CSC Name/Function _____

Total size excluding Comments and Documentation _____	
Comments and Doc. _____	Number of Machine Words _____
Size Reused _____	Size to be Reusable _____
Pre-existing Code Size _____	Mods to Pre-existing _____
# of Function Points _____	Number of Ada Objects _____
Number of Packages _____	Number of Tasks _____
Number of Blocks _____	Layers of Blocks _____
# of Ada Program Units _____	Layers Ada Program Units _____
Number of Ada Statements _____	Language _____
The values for this CSC are: Estimates _____ Actuals _____	

CSC Name/Function _____

Total size excluding Comments and Documentation _____	
Comments and Doc. _____	Number of Machine Words _____
Size Reused _____	Size to be Reusable _____
Pre-existing Code Size _____	Mods to Pre-existing _____
# of Function Points _____	Number of Ada Objects _____
Number of Packages _____	Number of Tasks _____
Number of Blocks _____	Layers of Blocks _____
# of Ada Program Units _____	Layers Ada Program Units _____
Number of Ada Statements _____	Language _____
The values for this CSC are: Estimates _____ Actuals _____	

CSC Name/Function _____

Total size excluding Comments and Documentation _____	
Comments and Doc. _____	Number of Machine Words _____
Size Reused _____	Size to be Reusable _____
Pre-existing Code Size _____	Mods to Pre-existing _____
# of Function Points _____	Number of Ada Objects _____
Number of Packages _____	Number of Tasks _____
Number of Blocks _____	Layers of Blocks _____
# of Ada Program Units _____	Layers Ada Program Units _____
Number of Ada Statements _____	Language _____
The values for this CSC are: Estimates _____ Actuals _____	

CSC Name/Function _____

Total size excluding Comments and Documentation _____	
Comments and Doc. _____	Number of Machine Words _____
Size Reused _____	Size to be Reusable _____
Pre-existing Code Size _____	Mods to Pre-existing _____
# of Function Points _____	Number of Ada Objects _____
Number of Packages _____	Number of Tasks _____
Number of Blocks _____	Layers of Blocks _____
# of Ada Program Units _____	Layers Ada Program Units _____
Number of Ada Statements _____	Language _____
The values for this CSC are: Estimates _____ Actuals _____	

COMPUTER SOFTWARE SIZE SUMMARY DATA FORM

CSC Name/Function _____

Total size excluding Comments and Documentation _____	
Comments and Doc. _____	Number of Machine Words _____
Size Reused _____	Size to be Reusable _____
Pre-existing Code Size _____	Mods to Pre-existing _____
# of Function Points _____	Number of Ada Objects _____
Number of Packages _____	Number of Tasks _____
Number of Blocks _____	Layers of Blocks _____
# of Ada Program Units _____	Layers Ada Program Units _____
Number of Ada Statements _____	Language _____
The values for this CSC are: Estimates _____ Actuals _____	

CSC Name/Function _____

Total size excluding Comments and Documentation _____	
Comments and Doc. _____	Number of Machine Words _____
Size Reused _____	Size to be Reusable _____
Pre-existing Code Size _____	Mods to Pre-existing _____
# of Function Points _____	Number of Ada Objects _____
Number of Packages _____	Number of Tasks _____
Number of Blocks _____	Layers of Blocks _____
# of Ada Program Units _____	Layers Ada Program Units _____
Number of Ada Statements _____	Language _____
The values for this CSC are: Estimates _____ Actuals _____	

CSC Name/Function _____

Total size excluding Comments and Documentation _____	
Comments and Doc. _____	Number of Machine Words _____
Size Reused _____	Size to be Reusable _____
Pre-existing Code Size _____	Mods to Pre-existing _____
# of Function Points _____	Number of Ada Objects _____
Number of Packages _____	Number of Tasks _____
Number of Blocks _____	Layers of Blocks _____
# of Ada Program Units _____	Layers Ada Program Units _____
Number of Ada Statements _____	Language _____
The values for this CSC are: Estimates _____ Actuals _____	

CSCI Total

Total size excluding Comments and Documentation _____	
Comments and Doc. _____	Number of Machine Words _____
Size Reused _____	Size to be Reusable _____
Pre-existing Code Size _____	Mods to Pre-existing _____
# of Function Points _____	Number of Ada Objects _____
Number of Packages _____	Number of Tasks _____
Number of Blocks _____	Layers of Blocks _____
# of Ada Program Units _____	Layers Ada Program Units _____
Number of Ada Statements _____	Language _____
The values for this CSC are: Estimates _____ Actuals _____	

RESOURCE EXPENDITURE DATA

1. Project Name:

2. CSCI or Subsystem Name:

3. Latest Month of Activity:

4. Department Manpower:

CSCI

WBS: DESIGN CODE & DEBUG TEST ENV TEST DOCUMENT 2.5

SYSTEM

WBS: RIM MGT TUNE & CTRL TEST ANAL INT & TEST

MONTHS

AFTER

CONTRACT

AWARD

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

RESOURCE EXPENDITURE DATA

20					
21					
22					
23					
24					
25					
26					
27					
28					
29					
30					
31					
32					
33					
34					
35					
36					
37					
38					
39					
40					
41					
42					
43					

[illegible]

A PLAN FOR COLLECTING ADA SOFTWARE DEVELOPMENT COST
SCHEDULE AND ENVIRONM. (U) TECOLOTE RESEARCH INC SANTA
BARBARA CA N J BRENNER ET AL. 02 APR 87 CR-0134/1

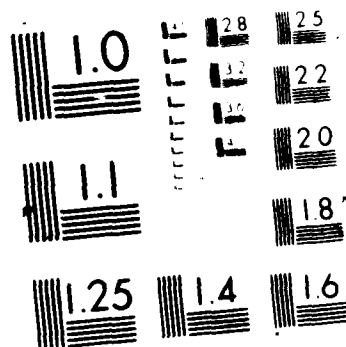
SCHEDULE AND ENVIRONMENT. (U) TECHNOLOGIE RESEARCH INC 3
BARBARA CA N J BRENNER ET AL. 02 APR 87 CR-0134/1

ESD-TR-87-167 F19628-84-D-0019

F/G 12/5

11

[illegible]



INSTRUCTIONS

ESD ADA SOFTWARE DEVELOPMENT DATA COLLECTION FORMS

SOFTWARE DEVELOPMENT PROJECT SUMMARY DATA FORM INSTRUCTIONS

1. Project Name and Date

Enter the name of the project and the date this form is being completed.

2. Development Contractor/Organization

Identify the company or organization which is actually performing the software design and development.

3. Project Description

3.1. Mission Description

Describe the overall mission or purpose of the system for which the software is being developed.

3.2. Major Hardware Interfaces

Identify the major hardware components which the software will interface. For example: radars, communications equipment, sensors, other embedded computer systems, etc.

3.3. Major System Functions

List the major functions performed by the system.

3.4. Major Software Functions

List the major functions performed by the software.

3.5. Number of CSCIs

Enter the number of Computer Software Configuration Items (CSCIs) into which the system is divided.

3.6. Computer Software Configuration Item Names

List all of the CSCI's which are a part of this project.

3.7. System User

Indicate the user for whom the system is being developed.

3.8. Relative Magnitude of the Software Effort

Often, software is a portion of a system development that includes hardware. Indicate the fraction of the system development cost allocated to software.

SOFTWARE DEVELOPMENT PROJECT SUMMARY DATA FORM INSTRUCTIONS

3.9. Software Development Standards

Indicate which DoD and individual service standards were applied to the software development.

4.1. - 4.15. Project Milestones

Identify the software milestones for the project. Enter the contract schedule date for each applicable milestone (enter N/A if a milestone is not applicable). If the milestone has not yet passed, enter the expected date. Otherwise, enter the actual date the milestone was passed. Although these milestones represent formal contractual activities in the Department of Defense software acquisition process, many non-defense projects will have milestones which are equivalent to these, e.g., contract award is equivalent to project start and critical design review is equivalent to completion of detail design.

If the formal milestones are not required in the project schedule, data for equivalent activities should be used. Definitions of these milestones are provided in Attachment A of these instructions. Unless otherwise indicated, the date should reflect the activity completion date. Where available, enter the actual date of completion for the milestone; for ongoing efforts, enter the current estimate for completion of the milestone.

5. Modern Development Method Use

Select the rating which best describes the use of listed Modern Development Methods.

0	No use of Method
1	Beginning, experimental use of Method
3	Reasonably experienced use of Method
5	Expert use of Method

6. Software Quality Required

Rate the software quality required for this project in each of the categories.

6.1. Usability

Usability is the extent to which the system is convenient and practical to use.

6.2 Reliability

Reliability is the probability that a system will satisfy its stated operational requirements for a specified period of time.

SOFTWARE DEVELOPMENT PROJECT SUMMARY DATA FORM INSTRUCTIONS

6.3. Efficiency

Efficiency is the extent to which a system fulfills its purpose without wasting resources.

6.4. Integrity

Integrity is the degree to which one subsystem can protect the operation of another subsystem.

6.5. Testability

Testability is the ease with which tests can be planned, specified, conducted, and analyzed for a system.

6.6. Portability

Portability is the extent to which a software system can be moved from one computer to another.

6.7. Correctness

Correctness is the degree to which a system fulfills its system requirements.

6.8. Maintainability

Maintainability is the degree to which a system facilitates the making of modifications during it's life.

6.9. Reusability

Reusability is the degree to which selected modules from one system can be used in another system.

6.10. Interoperability

Interoperability is the ease with which one software system can be coupled with another system.

7. Software Change History by Phase

Enter the number of changes which occurred during each completed development phase, the net increase/decrease in the total system delivered source lines of code count and the net increase/decrease in the estimated manpower for the software development effort.

If this information is available at the CSCI level, then provide the answers on the CSCI forms and skip this question.

SOFTWARE DEVELOPMENT PROJECT SUMMARY DATA FORM INSTRUCTIONS

8. Comments

Please provide any additional information about events that affected this software development effort as an aggregate. There is a separate comment section for events that affected individual CSCIs.

SYSTEM LEVEL OR CSCI LEVEL DOCUMENTATION FORM INSTRUCTIONS

This form is used to report the size and quality of the system level and CSCI level computer software documentation. Use one copy for the system level documentation, and additional forms for the documentation associated with each CSCI.

Check off whether this form is being used to report system level or CSCI documentation.

1. Supply the name of the project and the date that this form is being filled out.
2. Identify the development contractor / organization that is developing the software system or CSCI for which the documentation data are being reported.
3. If the data are being reported for a CSCI, identify the CSCI.
4. For each applicable document, check whether the document was provided by the Government or another agency, or whether it was written by the development contractor / organization. If the document was provided and then rewritten by the software developer, check both columns.

Indicate the quality of the document. The rating scale is:

Very Low	1
Low	2
Nominal	3
High	4
Very High	5

If the document was written by the development organization, indicate the date it was completed. Completion date here implies the document was available for use in the software development effort.

Finally, supply the estimated or actual number of pages.

If documents were produced that are not listed, list them under "Other."

DEVELOPMENT COMPUTER SYSTEM SUMMARY DATA FORM INSTRUCTIONS

1. Development Contractor / Organization

1.1. Name

Identify the company or organization which is actually performing the software design and development using this development system.

1.2. Location

Enter the location of the company or organization which is actually performing the software design and development using this development system.

2. Project Name

Enter the name of the project.

3. Development System Attributes

3.1. Development Computer Manufacturer and Model

Enter the development computer manufacturer and the model number of the computer.

3.1.1. Main Memory Size

Enter the main memory size, in words, for the development computer.

3.1.2. Word Size

Enter the word size (in bytes) of the development computer.

3.1.3. Maximum Main Memory Size

Enter the maximum main memory size, in words, for the development computer.

3.1.4. Is this a virtual memory machine? Answer yes or no.

3.1.5. CPU Processing Speed

Enter the CPU processing speed, in millions of instructions per second (MIPS), for the development computer.

DEVELOPMENT COMPUTER SYSTEM SUMMARY DATA FORM INSTRUCTIONS

3.1.6. Average Number of People Per Terminal

Enter the average number of people sharing a terminal.

3.2. Percentage of Source Instructions developed using each of the following access modes (Total=100%).

Enter the percentage of source instructions developed using each access mode defined below:

3.2.1. Batch

Processing of a group of items prepared or required for one or more related operations with no provision for unscheduled interruption.

3.2.2. Dedicated Processor

The processor is completely dedicated to this development.

3.2.3. Test Bed with High Priority

Development facilities set aside for target system simulation, developers have high priority access.

3.2.4. Test Bed with Low Priority

Development facilities set aside for target system simulation, developers have low priority access.

3.2.5. Interactive

Usage of a computer via a terminal where each line of input is immediately processed by the computer.

3.2.6. Other

Enter any other computer access modes used in the development of this CSCI, and the percentage of code developed in that mode.

3.3. Software Development System

For the following questions, respond with the range of the estimated average value, or the actual average value.

3.3.1. Turnaround Time No Recompile

Enter the amount of time, in hours, that it takes from logon until you receive a hard copy. This measures the time lost while dealing with the development computer (logging on, commanding actions, waiting for a response on a multi-user system, waiting for slow printers, delivery of

DEVELOPMENT COMPUTER SYSTEM SUMMARY DATA FORM INSTRUCTIONS

printouts from remote printers, etc.)). It does not include the productive time a project member may spend reading program listings, editing programs or text, etc.

3.3.2. Turnaround Time Recompile Required

Enter the amount of time, in hours, required to compile a program.

3.3.3. Terminal Response

Indicate the number of seconds from the time you hit the Return (Enter) key until the terminal responds. Also check whether the response time is variable or consistent.

3.3.4. Number of Major Changes per Month

These may be changes in the program editors, compilers or other tools, changes in the command languages, or changes in the target hardware. Each change may cause developers to lose time due to learning the system, changing their code, procedures, etc. Some virtual machines have been used for many years without changes, and no changes are expected.

Indicate the average number of major changes per month. Include fractional values.

3.3.5. Number of Minor Changes per Month

Indicate the average number of minor changes per month. Include fractional values.

3.4. Hours the Development Computer System Operates

Indicate the daily hours of operation of the development computer system.

3.5. Percent of Development System Operation Hours Available

Enter the percentage of hours indicated in 3.4. that the development computer will be available to the development organization.

3.6. Development System Security Classification Level

Indicate the highest security level the development computer system is cleared to.

4. Development Computer System Tools

This question seeks information on the automated tool sets available on the development system, and on methods used for which there was not an implementing computer program.

DEVELOPMENT COMPUTER SYSTEM SUMMARY DATA FORM INSTRUCTIONS

Indicate whether an item was used manually (i.e. a method) or as an automated tool.

If the tool or method was used, indicate the frequency of use using the following scale:

1	Very Low
3	Nominal
5	Very High

If the tool or method was used, rate the development team's experience with the tool or method as of the first use on this project.

0	No use of the tool
1	Beginning, experimental use of the tool
3	Reasonably experienced use of the tool
5	Expert use of the tool

If the tool or method was used, indicate the phase or phases when the tool or method was used. (Documentation is not a phase but is included because it is believed to be important).

1	Requirements Development
2	Preliminary Design
3	Detailed Design
4	Code and Unit Test
5	Test and Integration
6	System Test
7	Maintenance
8	Documentation

If the tool or method was used, list the CSCIs affected. Indicate the CSCI by using the number designation from question 6.

Finally, if the tool or method has a 'brand name', please identify it.

5. Development Locations

5.1. Number of Development Sites

Enter the number of different development locations for this CSCI.

5.2. Development Computer and Site Locations

List the site and computer location for each separate development site.

DEVELOPMENT COMPUTER SYSTEM SUMMARY DATA FORM INSTRUCTIONS

The tool definitions are:

Assembler -- translates a program expressed in an assembly language into object code

Assembler Options -- features of the assembler that provide additional capabilities such as listing assembler source, listing errors, inserting debug hooks, and collecting statistics

Body Stub Generator -- creates null bodies for specifications requiring bodies whose bodies have yet to be specified (i.e., top-down construction)

Change Impact Analyzer -- determines, for a proposed support or enhancement operation, the impact of proposed changes to the software system.

Change Request Analyzer -- analyzes change requests to determine necessity of the change, technical and economic impacts, and approach to accomplishing the change

Code Auditor -- examines whether predefined rules have been followed (such as coding standards)

Code Interface Analyzer -- checks the interfaces between coded program elements for consistency and adherence to predefined rules

Code Invocation Analyzer -- checks coded modules for determining the calling relationships between elements

Command Language Processor -- converts command language constructs into functions performed by an operating system

Compiler Code Generator -- transforms the intermediate language form of a computer program into machine language

Compiler Options -- features of the compiler that provide additional capabilities such as source listings, error listing, conditional compilation, inserting debug hooks, optimization, collecting statistics, reordering compilation units, tracing, allowing compiler or heap space

Constraint Evaluator -- generates and/or solves path input or output constraints for determining test input or for proving programs correct

Control Flow Tracer -- records the source statements and/or branches that are executed in a program in their execution order

DEVELOPMENT COMPUTER SYSTEM SUMMARY DATA FORM INSTRUCTIONS

- Coverage/Frequency Analyzer -- determines and assesses measures associated with the invocation of program structural elements to determine the adequacy of test run. Typically, coverage measures are in terms of statements, branches, paths, or modules executed by certain data sets.
- Cross Referencer -- logically references entities to other entities
- Data Flow Analyzer -- checks the sequential patterns of definitions and references of data, based upon the use of program control flow
- Data Flow Tracer -- monitors the current, actual state of variables in a program
- Data Type Analyzer -- evaluates whether the domain of values attributed to an entity is properly and consistently defined
- Debugger -- steps through a program, allowing the examination and setting of values
- Design Analyzer -- checks the interfaces between designed program elements for consistency and adherence to predefined rules, usually based upon the contents of the design database
- Design Document Generator -- collects information and generates documentation in a military-specification format for design documentation
- Design Language Processor -- transforms formal design language constructs into an internal database representation for subsequent analysis
- Design Prototyper -- rapidly constructs critical functions of a system to determine the best design approach
- Executable Assertion Checker -- checks user-embedded statements that assert relationships between elements of a program. An assertion is a logical expression that specifies a condition or relation among the program variables. Checking may be performed with symbolic or run-time data.
- Execution Timer -- reports actual CPU, wall-clock, or other times associated with executing parts of a program
- Execution Tracer -- monitors the historical record of program execution

DEVELOPMENT COMPUTER SYSTEM SUMMARY DATA FORM INSTRUCTIONS

- Formatter -- arranges text according to predefined and/or user-defined conventions
- Formal Verifier -- uses rigorous mathematical techniques to prove the consistency between an algorithmic solution and a rigorous, complete specification of the intent of the solution
- Graphics Editor -- has editing capabilities provided for graphical data
- Graphics Generator -- provides the input, construction, storage, retrieval, manipulation, alteration, and analysis of pictorial data
- Interactive Debugger -- performs debugging activities at the direction of a user
- Intermediate Language Generator -- transforms a source program into an intermediate representation
- Interpreter -- translates a source program into some intermediate data structure, then executes the algorithm by carrying out each operation given in the intermediate structure
- I/O Specification Analyzer -- analyzes the input and output specifications in a program, usually for the generation of test data
- Line Editor -- has editing capabilities that require input of a line number and editing function indicator
- Linker -- creates a load module from one or more independently translated object modules or load modules by resolving cross-references among the object modules, and possible by relocating elements
- Macro Expander -- augments instructions in a source language with user-defined sequences of instructions in the same source language
- Memory Dump -- the contents of storage (or a part of storage) for a specific purpose, such as a safeguard against faults, or in connection with debugging
- On-line Assistance Processor -- a user interface feature that is part of the input/output process of a programming support environment (such as error assistance, on-line tutoring, etc.)

DEVELOPMENT COMPUTER SYSTEM SUMMARY DATA FORM INSTRUCTIONS

- Physical Units Analyzer -- determines whether the units or physical dimensions attributed to an entity are properly defined and consistently used
- Problem Report Analyzer -- analyzes problem reports for the purpose of determining the validity of the reported problem and corrective action
- Program (Application) Generator -- constructs computer programs using translation or interpretation, based upon rules for data structures and control (as in 4th generation languages)
- Program Tuner -- determines what parts of a program are being executed the most
- Quality Analyzer -- measures specified quality factors for use during the evaluation of software products (and prediction of software quality) at key milestones during development. Factors to be analyzed include: efficiency, integrity, reliability, survivability, usability, correctness, maintainability, verifiability, expandability, flexibility, interoperability, portability, and reusability.
- Regression Tester -- reruns test cases which a program has previously executed correctly, in order to detect errors spawned by changes or corrections made during software development and maintenance
- Requirements Analyzer -- checks formally stated requirements to determine their consistency and completeness
- Requirements Documentation Generator -- collects information and generates documentation in a military-specification format for requirements documentation
- Requirements Language Processor -- transforms formal software requirements statements into an internal database representation for subsequent analysis
- Requirements Prototyper -- rapidly constructs critical functions of a system early in the life cycle for purpose of understanding the requirements. The constructed code may be thrown away.
- Scanner -- examines an entity sequentially to identify key areas or structure
- Screen Editor -- has screen-oriented editing capabilities, using cursor keys or pointing device to move around a full screen at a time

DEVELOPMENT COMPUTER SYSTEM SUMMARY DATA FORM INSTRUCTIONS

- Sorter/Merger -- arranges items in a specific order
- Source Converter -- modifies an existing program to enable it to operate with similar functional capabilities in a different environment
- Statement Profiler -- analyzes a computer program to determine statement types, number of occurrences of each statement type, and the percentage of each statement type in relation to the complete program
- Structure Checker -- detects structural flaws with a program, such as improper loop nestings, unreferenced labels, unreachable statements, etc.
- Symbolic Debugger -- performs debugging activities at the source-language level
- Symbolic Executor -- reconstructs logic and computation along a program path by executing the path with symbolic, rather than actual, values of data
- Syntax-Directed Editor -- has editing capabilities that are sensitive to the context in which they are applied, using specific programming language templates based upon syntactical inputs
- Test Condition Analyzer -- processes formal requirements language to determine data values to be examined and the mechanisms to be used in the verification of test results
- Test Harness Generator -- produces programs that provides input to and encapsulates outputs from a testable program unit
- Text Editor -- has editing capabilities for textual data
6. List the CSCIs developed on this development computer system.
7. Provide any comments that will further aid our understanding of the effect of the development computer system on cost and schedule.

TARGET COMPUTER SYSTEM DATA FORM INSTRUCTIONS

If more than one target computer system will be used, complete a separate form for each.

1. Development Contractor / Organization

1.1. Name

Identify the company or organization which is actually performing the software design and development using this development system.

1.2. Location

Enter the location of the company or organization which is actually performing the software design and development using this development system.

2. Project Name

Enter the name of the project.

3. Target System Attributes

3.1. Target Computer Manufacturer and Model

List the manufacturer and model number of the target computer for this CSCI.

3.1.1. Main Memory Size

Enter the main memory size, in words, for the target computer.

3.1.2. Word Size

Enter the word size of the target computer.

3.1.3. Number of Processors in the Target

Enter the number of processors in the target computer.

3.1.4. Maximum Main Memory Size

Enter the maximum main memory size, in words, for the target computer.

3.1.5. Indicate if this is a virtual memory machine. Answer yes or no.

3.1.6. CPU Processing Speed

Enter the CPU processing speed, in millions of instructions per second (MIPS), for the target computer.

TARGET COMPUTER SYSTEM DATA FORM INSTRUCTIONS

3.1.7. Reserve Memory Requirement

Enter the reserve memory requirement for the target computer as a percentage of main memory size (3.1.1.).

3.1.8. Reserve Timing Requirement

Enter the reserve timing requirement for the target computer as a percentage.

3.1.9. Programming Language(s) Implementation

List the programming language(s) for the target computer if they are different or if the implementation is different from that used on the development computer.

3.1.10. Differences between Development and Target

List the differences between the development and target computer.

3.1.11. Accessibility to the Target

Indicate the access that the development team will have to the target computer.

3.1.12. Target Simulator Size

Enter the size, in words, of the target simulator.

3.2. Changes to the Target Computer System During Development

For the following questions, respond with the range of the estimated average value, or the actual average value.

These may be changes in the program editors, compilers or other tools, changes in the command languages, or changes in the target hardware. Each change may cause developers to lose time due to learning the system, changing their code, procedures, etc. Some virtual machines have been used for many years without changes, and no changes are expected.

3.2.1. Number of Major Changes per Month

Indicate the average number of major changes per month. Include fractional values.

3.2.2. Number of Minor Changes per Month

Indicate the average number of minor changes per month. Include fractional values.

TARGET COMPUTER SYSTEM DATA FORM INSTRUCTIONS

3.3. Indicate if the target computer was developed concurrently with the software.

4. List the CSCI's executed on this Target Computer.

5. Add any comments that will explain any unusual cost impact that can be attributed to the use of this target computer.

COMPUTER SOFTWARE CONFIGURATION ITEM SUMMARY DATA FORM INSTRUCTIONS

1. CSCI Identification

1.1. CSCI Name and Date

Enter the name of the CSCI and the date the form is being completed.

1.2. Development Contractor/Organization

Identify the company or organization which is actually performing the software design and development.

1.3. Project Name

Enter the name of the project containing this CSCI.

2. CSCI Functional Description

Give a brief description of the functions and purpose of the CSCI.

2.1. Operating Environment

Select the operating environment which best describes this CSCI.

3. CSCI Schedule Data

3.1. Milestones

3.1.1. - 3.1.14. Individual Milestones

Enter three dates for each milestone; the date specified in the contract, the estimated date and the actual date (if the CSCI has reached the milestone). If one or more of the milestones listed do not apply to this CSCI, enter N/A (Not Applicable). For a detailed description of each milestone, please see Attachment A.

3.2. Schedule Acceleration/Stretchout Assessment

Indicate the degree of schedule acceleration or stretchout that the original (contract) schedule dates in 3.1. represent relative to the normal time required to develop this CSCI. For example, if the specified schedule is 24 months and the normal development time is estimated at 30 months, the schedule acceleration/stretchout is 80%.

COMPUTER SOFTWARE CONFIGURATION ITEM SUMMARY DATA FORM INSTRUCTIONS

4. Personnel

4.1 Average Quality and Experience

4.1.1. Analyst Quality

Rate the Project analysts' performance as a team against all other analysts, using the following scale:

15th Percentile	Non-Functioning Team
35th Percentile	Functional but Not Very Effective
55th Percentile	Functional and Effective
75th Percentile	Extraordinary
90th Percentile	Nearly Perfect

Analysts perform the following functions:

- Defines the software architecture
- Creates preliminary design specifications
- Solves requirements or design errors
- Assists test planning and software testing
- Assists software integration
- Assists software/hardware integration

4.1.2. Programmer Quality

Rate how well the programmers working on the project will perform as a team compared to all other programming teams in the world. Use the following table as a guideline:

15th Percentile	Non-Functioning Team
35th Percentile	Functional but Not Very Effective
55th Percentile	Functional and Effective
75th Percentile	Extraordinary
90th Percentile	Nearly Perfect

A programmer performs the following tasks:

- Defines the design details, ("code-to" design, PDL, flow charts, etc.)
- Develops the programming language code
- Integrates the modules (units) into the software systems

4.1.3. Team Programming Language Experience

Enter the average number of years of programming language experience for the entire CSCI development team. Programming experience measures how much similar experience the team has acquired on the same or similar languages by the start of Full-Scale Development.

The experience must relate to the same type of language as the one being proposed. Language experience on unrelated languages must not be counted in this experience evaluation.

COMPUTER SOFTWARE CONFIGURATION ITEM SUMMARY DATA FORM INSTRUCTIONS

4.1.4. Development Methods Experience

Enter the average number of years of experience the software development team has with the modern development methods which are being used during the development of this CSCI.

4.1.5. Development Virtual Machine Experience

Enter the average number of years of experience the software development team has with the virtual machine. The virtual machine includes the computer, the operating system, job control languages (commands given to the computer to perform some task), automated tools such as text editors, language compilers, etc. and all the things the developers will use to develop the software. The virtual machine can also include the target computer.

4.1.6. Applications Area Experience

Enter the average number of years of experience the analyst team has with the applications area for this CSCI. Application experience is the analyst team's relevant experience in designing with similar applications. This includes the team's experience at the time of system design review (when the software requirements are reviewed).

An application is considered similar if it has similar types of functions, goals, or inherent problems and the experience will be useful during the project.

4.1.7. Support Software/Tools Experience

Enter the average number of years of experience the software development team has with the software tools that will be used during the development of this CSCI.

4.2. Average Formal Training

4.2.1. Formal Programming Language Training Days

Enter the average number of days the software development team has had in formal programming language training in the language that will be used on this CSCI.

4.2.2. Formal Development Methods Training Days

Enter the average number of days the software development team has had in formal modern development methods training.

4.2.3. Formal Tools Training Days

Enter the average number of days the software development team has had in formal tools training.

COMPUTER SOFTWARE CONFIGURATION ITEM SUMMARY DATA FORM INSTRUCTIONS

4.2.4. Formal Development System Training Days

Enter the average number of days the software development team has had in formal development system training.

4.3. Peak Designer Staff

Enter the maximum number of people available for the designer staff.

4.4. Peak Programmer Staff

Enter the maximum number of people available for the programmer staff.

4.5. Peak Tester Staff

Enter the maximum number of people available for the test staff.

4.6. Maximum Staffing Rate

Enter the maximum rate, in persons per year, that people can be added to the staff of this CSCI.

4.7. Overall Personnel Availability

Enter the percentage of time the development staff will be working on this CSCI.

5. Reliability Requirement

Indicate the required reliability of the CSCI by marking the box which best describes the reliability requirement.

Very Low	0	No Reliability requirement
Low	1	Low reliability
Nominal	2	Mil-Spec reliability
Very High	3	Reliability for high potential Loss
Extra High	5	Risk of Loss to Human Life

6. Complexity

6.1. Inherent Difficulty of Application

Complexity specifies the relative complexity (inherent difficulty) of the specific software component. This complexity is independent of the developer's ability to implement the CSCI.

COMPUTER SOFTWARE CONFIGURATION ITEM SUMMARY DATA FORM INSTRUCTIONS

Select a value from the table below which best describes the complexity of this CSCI.

Software Complexity Criteria

Type Rating	Control Operations	Computational Operations	Device-dependent Operations	Data Management Operations
Very Low	Sequenced code with a few non-tested SP operators: DOs CASEs, IF THEN-ELSEs. Simple predicates.	Evaluation of simple expressions, e.g. $A=B+C*(D-E)$	Simple read, write statements with simple	Simple arrays in main memory
Low	Straightforward nesting of SP operators. Mostly single predicates	Evaluation of moderate level expressions e.g., $D=SQRT(B**2-4.*A*C)$	No cognizance needed of particular processor of I/O device characteristics. I/O done at GET/PUT level, no cognizance of erlap	Single file subsetting with no data structure changes, no data edits, no intermediate files
Nominal	Mostly simple nesting. Some intermodule control. Decision tables	Use of standard math and statistical routines. Basic matrix and vector operations	I/O processing includes device selection Status checking and error processing	Multiple input and single file output. Simple structural changes simple edits
High	Highly nested SP operators with many compound predicates. Queue and stack control. Considerable intermodule control	Basic numerical analysis: multivariate interpolation, ordinary differential equations. Basic truncation roundoff concerns	Operations at physical I/O level(physical storage address translations, seeks, reads, etc.) Optimized I/O overlap	Special purpose subroutines activated by data stream contents. Complex data restructuring at record level

COMPUTER SOFTWARE CONFIGURATION ITEM SUMMARY DATA FORM INSTRUCTIONS

Very High	Reentrant and recursive coding. Fixed priority interrupt handling	Difficult but structured numerical analysis: near-singular matrix equations, partial differential equations	Routines for interrupt diagnosis, servicing, masking. Communication line handling	Generalized parameter-driven file structuring routine. File building, command processing, search optimization
Extra High	Multiple resource scheduling with dynamically changing priorities. Microcode level control	Difficult and unstructured numerical analysis: highly accurate analysis of noisy stochastic data	Device timing-dependent coding, micro-programmed operations	Highly coupled dynamic relational structures. Natural language data management

6.2. Inherent Complexity of Data Structures

Rate the complexity of the data structures which will be used in this CSCI using the following table:

Very Low	0	Simple data structures
High	3	Complex data structures
Extra High	5	Very complex data structures

6.3. System Integration and Test

Select the type of system integration and test which best describes the integration and test activities for this CSCI.

7. Software Size Description

7.1. Computer Software Configuration Item Size Information

Check the size format that will be used to answer all portions of 7.1. If data are available for more than one format, xerox the pages of the data collection form and repeat the questions for each format available.

7.1.1. Total Size

7.1.1.1. Enter the total size of the code, excluding documentation. Enter the initial estimate, and either the current estimate or the actual values if the project is complete.

COMPUTER SOFTWARE CONFIGURATION ITEM SUMMARY DATA FORM INSTRUCTIONS

7.1.1.2. Enter the size of the documentation. Enter the initial estimate, and either the current estimate or the actual values if the project is complete.

7.1.2. Operation Response Requirements

Indicate the response mode required in the operational system using the following guidelines:

Real-time - The software must complete processing in response to an event prior to the occurrence of the next event. Arrival of the data and the occurrence of events is not under the control of the software and extra effort in the design, test and implementation of the software is required to satisfy time and processing requirements.

On-line - Software in this category must respond within a human compatible time frame, usually within a few seconds. Also requires additional development effort, but not the extra level required for real-time software.

Time-constrained - Software in this category must complete processing within a specified time frame which is not as restrictive as real time or on-line requirements. Time lines are in the order of minutes or hours; sometimes a clock time is specified for completion of processing.

Non-time-critical - There is no time constraint for completion of processing for this category of software.

Indicate the type of format used to determine the size of this CSCI.

7.1.3. Source Statement Mix

7.1.3.1. Statement Types

Enter the percentage of the delivered lines of source code for this CSCI for each of the statement types listed using the following guidelines:

Logical - statements which control the execution sequences in the program and include constructs such as IF-THEN-ELSE, DO WHILE, DO UNTIL, CASE, GO TO or CALL.

Command - statements which direct the system software to perform specific functions or to create the environment required to support the software. These statements are generally written in a language specific to the computer hardware.

Mathematical - statements which perform computations. This category includes coded equations for algorithms, vector algebra, modeling, index computation, etc.

COMPUTER SOFTWARE CONFIGURATION ITEM SUMMARY DATA FORM INSTRUCTIONS

Data Manipulation - statements which perform input and output, as well as the storage, movement and modification of data. Format statements are also included.

Data Declaration - statements which are non-executable and define the characteristics and values of the data contained in the program.

Data Typing - statements which are non-executable and define the characteristics of the data contained in the program.

Ada Tasking - statements which are written to execute Ada tasking.

Invocation - calls to CSC modules and system procedures.

Indicate the type of format used to determine the size of this CSCI.

7.1.3.2. CSCI Source Code Mix

Enter the percent of the deliverable lines of source code that performs each of the categories of operation defined below: ('% Code' is the percentage of the entire CSCI size. '% New Design' is the percentage of only the new code. '% New Code' is the percentage of the new code only)

Operating Systems - Task management. Memory management. Heavy hardware interface. Many interactions. High reliability and strict timing requirements.

Interactive Operations - Real-time man/machine interfaces. Human engineering considerations and error protection are very important.

Real-Time Command & Control - Machine-to-machine communications under tight timing constraints. Queuing not practicable. Heavy hardware interface. Strict protocol requirements.

On-Line Communications - Including machine-to-machine communications with queuing allowed. Timing restrictions not as severe as with real-time command and control.

Data Storage & Retrieval - Operation of data storage devices, data base management, secondary storage handling, data blocking and deblocking, hashing techniques. Primarily hardware oriented code.

String Manipulation - Instructions dealing with movement, manipulation and creation of consecutive characters.

Mathematical Operations - Routine mathematical applications with no overriding constraints.

COMPUTER SOFTWARE CONFIGURATION ITEM SUMMARY DATA FORM INSTRUCTIONS

7.1.4. Target Computer Impact on Source Code

For the following questions, supply a range estimate or the actual value.

7.1.4.1. Memory Constraint Percent

Memory Constraint evaluates the anticipated effort to reduce memory usage. Such economy is usually seen in overlapping/segmentation, special coding, common memory management, and performance trade-offs.

Memory includes the computer's main storage for loading and executing programs and temporary storage of data. Memory includes Random Access Memory (RAM) Read Only Memory (ROM), core memory and similar program memory storage/execution devices only. Memory does not include magnetic tapes, disks, bubble memory used as a storage device, etc.

Indicate the percent of the CSCI that is subject to memory conservation techniques.

7.1.4.2. CPU Time Constraint Percent

These time constraints are often required to meet overall system performance requirements. For example, the software may be required to respond to a user's request within one second. In such a case, the code would be tuned to enhance performance in the areas that affect a request from the user until such a request could be handled in under one second.

Indicate the percent of the CSCI that must have special attention to ensure adequate time performance (not real time).

7.1.4.3. Real-time Operation Percent

If real time software does not perform within the time limits placed on it by an outside source (usually hardware), information will be lost or changed before the software can process it, or data may not be correctly output to the device. Examples of this information are: a signal that is received 1,000 times per second, information that must be received from a communication network or lost, etc.

Indicate the percentage of the CSCI that must perform its processing function within absolute time constraints (may be measured in 1,000ths or 10,000ths of a second or even less).

7.1.4.4. Multi-processor Percent

Indicate the percent of the CSCI that must handle multi-processors.

7.1.4.5 Multi-Target Percent

Indicate the percent of the CSCI that is specially written to run on multiple targets.

COMPUTER SOFTWARE CONFIGURATION ITEM SUMMARY DATA FORM INSTRUCTIONS

7.1.5. CSCI Reused Code from Other Projects

7.1.5.1. Total Pre-existing

Enter the size of the pre-existing code completed and tested prior to this CSCI development. If this CSCI is a totally new development, enter N/A.

7.1.5.2. Total Deleted

Enter the size deleted from an existing CSCI if this CSCI is reusing code from another project. Otherwise, enter N/A.

7.1.5.3. Total Modified

Enter the total size modified. (Modifications made to the preexisting code.) If this CSCI is NOT reusing code from another project, enter N/A.

7.1.5.4. Percent Re-Design Effort

Enter the amount of design effort required to produce the CSCI when compared with all new development. If the CSCI is all new, the design effort is 100%.

When rebuilding or reusing software, the design effort can account for additional work required to understand existing designs. This may make reuse more costly than it first appears.

Reuse of poor, undocumented, or obsolete designs can justify a design effort rating of greater than 100%.

7.1.5.5. Percent Re-Implementation Effort

Enter the implementation effort to code and unit test the CSCI as compared to all new implementation. Just like the design effort and test effort, the implementation effort percentage code is determined by the anticipated effort in comparison to a new effort. A complete implementation is 100%.

7.1.5.6. Percent of Re-Test Effort

Enter the test effort required to revalidate the CSCI when compared with all new development. This includes all test effort after code and unit test until the CSCI is ready for delivery to system integration with no known liens on performance. This test effort does not include CSCI-to-CSCI integration or hardware/software testing.

If the CSCI under estimation is all new, the test effort is 100%.

7.1.5.7. List of Projects which contained the Re-usable Code

List the projects which contained the code which is being reused.

COMPUTER SOFTWARE CONFIGURATION ITEM SUMMARY DATA FORM INSTRUCTIONS

7.2. Function Point Data

Enter the following if an automated counting tool is available.

7.2.1. Number of Inputs

Enter the estimated or actual number of unique input types that change data.

7.2.2. Number of Outputs

Enter the estimated or actual number of external outputs.

7.2.3. Number of Inquiries

Enter the estimated or actual number of input/output combinations where an input causes or generates an immediate output.

7.2.4. Number of Data Files

Enter the estimated or actual number of logical files that are generated, used, or maintained by the program.

7.2.5. Number of Interfaces

Enter the estimated or actual number of internal files passed or shared.

7.2.6. Total Number of Function Points

Enter the estimated or actual total number of function points.

7.3. Size/Complexity Data

Enter the requested counts if an automated counting tool is available.

8. Data Base Size (An automated counting tool may be needed)

8.1. Total Data Base Size

Enter the estimated or actual total data base size (in target machine words).

8.2. Total Unique Data Items

Enter the estimated or actual number of total unique data items. A data item is a field in a record.

8.3. Total Number of Records

Enter the estimated or actual total number of records (a record is a heterogeneous set of data items in a named data group).

COMPUTER SOFTWARE CONFIGURATION ITEM SUMMARY DATA FORM INSTRUCTIONS

8.4. Unique Data Types

Enter the estimated or actual number of unique data types (e.g. integer, double, floating point, etc.).

9. Special Display Requirements

Special display requirements rate the amount of extra effort required to interface with the user. Many programs require only simple inputs and outputs (and no special display requirements).

Special displays are the user's window into the computer. (This is how the users see what is happening and view results). It is often a video display terminal of some sort, although it could be a hard copy printer and keyboard or another device that allows the user to interact with the software.

Indicate the special displays implemented in this CSCI.

10. Software Failure History (Errors by Phase)

Enter the number of requirements errors, the number of design errors, the number of implementation errors, and the total number of errors for each phase listed. Errors should be unique in every phase (i.e. an error which is discovered in code and debug should not be in subsequent error counts even though it may not be fixed yet). An error is a reported anomaly for which a resolution is a change to the software or specification.

11. Software Change History by Phase

Enter the number of changes which occurred during each completed development phase, the net increase/decrease in the total system delivered source lines of code count and the net increase/decrease in the estimated manpower for the software development effort.

If this information is available only at the system level, then answer this question on the SOFTWARE DEVELOPMENT PROJECT SUMMARY DATA FORM only.

12. CSCI Development Attributes

12.1. Development Environment

12.1.1. Resource Dedication

Resource dedication is the availability of the virtual machine to the software development project. If the machine is used by other organizations of the company, then the availability probably decreases. Even dedicated engineering resources may not be fully available due to conflicts between teams, hardware developers, etc.

Enter the percentage of time 0 - 100% that the development computer will be available to the CSCI development team.

COMPUTER SOFTWARE CONFIGURATION ITEM SUMMARY DATA FORM INSTRUCTIONS

12.1.2. Resource/Support Location

Resources are things like terminals or manuals. Support includes system consultants, programmers language support, and development tool support. Access may be limited by physical distance (terminals in other buildings, consultants located in other states) or by procedural constraints (difficult procedures, or by personnel problems - uncooperative people).

Indicate the distance (miles) of the development resources and support from the development site.

12.1.3. Security Level

Enter the level of security required for this CSCI (e.g. unclassified, classified, secret, etc.).

12.1.4. Contract Type

Indicate the type of contract the CSCI was developed under.

CPFF = Cost Plus Fixed Fee
CPIF = Cost Plus Incentive Fee
FFP = Firm Fixed Price
FPIF = Fixed Price Incentive Fee

12.2. Specific Development Goals

Rate the items listed from 0 - 5 if they will affect the development of the CSCI. The items are described in greater detail below.

12.2.1. Maximum Maintainability

The CSCI is designed and coded for easy maintainability.

12.2.2. Maximum Reuse of Pre-existing Software

As much code as possible is used from a pre-existing CSCI.

12.2.3. Maximum Reusability of CSCI Level Products

CSCI end-products are coded for re-use.

12.2.4. Maximum Reusability of TLCSC Products

Top-Level CSC Products are designed and coded to be reused.

COMPUTER SOFTWARE CONFIGURATION ITEM SUMMARY DATA FORM INSTRUCTIONS

12.2.5. Maximum Reusability of LLCSC Products

Low-Level CSC Products are designed and coded to be reused.

12.2.6. Maximum Output Clarity

The CSCI's output (i.e. displays and reports) is developed for maximum readability.

12.2.7. Maximum Use of Off-the-Shelf Software

The CSCI uses off-the-shelf software to the greatest extent possible.

12.2.8. Language/Tool/Method Evaluation

The goal is to evaluate the use of a specific language tool or method rather than just developing software.

13. Special Ada Features

Enter the names of any special Ada language features which were/will be explicitly avoided during the development of this CSCI.

If there are special Ada features used in the CSCI, indicate if there is a development organization internal standard regarding the use of the special features.

14. Special Problems or Comments

List any special problems or comments which are important to the development of this CSCI, but have not been covered in this document.

COMPUTER SOFTWARE SIZE SUMMARY DATA FORM INSTRUCTIONS

1. CSCI Name and Date

Enter the name of the CSCI and the date the form is being completed.

2. Development Contractor/Organization

Identify the company or organization which is actually performing the software design and development of this CSCI.

3. Project Name

Enter the name of the project which contains this CSCI.

4. CSCI Size Description

4.1. Size Format

Indicate the size format which most accurately describes the method used to determine the size of this CSCI. SLOC is Source Lines Of Code. Note that this size measure is only used for the Total Size, Comments and Doc. (sometimes), Size Reused, Size to be Reusable, Pre-existing Code Size, and Mods to Pre-existing.

4.2. CSC Size

List each CSC and its size values (list CSCI size if CSC sizes are not available). If a size value does not apply to the CSC enter "N.A." If the size values are available for more than one size format, fill out a separate form for each size format available.

The number of words is measured using target computer words.

The pre-existing code size is the size of a pre-existing CSC that was rebuilt or modified.

Mods to pre-existing is the number of lines added, deleted, or modified in the pre-existing CSC.

Size reused is the size of a CSC reused without modification.

Size to be reusable is the amount of the CSC that was designed to be reusable code.

Language is asked for because the CSC may not be in Ada. The code could be reused from a project using another language, or this CSCI could be designed in Ada and implemented in another language.

RESOURCE EXPENDITURE DATA FORM INSTRUCTIONS

This form is designed to collect time-phased manpower data for the software development project at the lowest level of detail available. Attach a copy of the cost/work breakdown structure used to collect manpower data for software activities on this contract/development project.

1. Project Name

Enter the project name.

2. CSCI or Subsystem Name

If the resource expenditure data is for an identified CSCI, state the name of the CSCI here and report expenditure data in accord with the WBS elements identified in the row, "CSCI WBS." Six WBS elements are shown in this row which breaks down the development labor into categories of design, code and debug, test documentation, test, program documentation, and verification and validation. If the CSCI resource data are not available at this level, report the totals for these activities and note on the sheet which if any of these activities may not be included in the resource expenditure data.

If the resource expenditure data is for system level programmatic activities necessary to the integration of two or more CSCIs within one hardware subsystem, state the subsystem name and report resource expenditure data according to the WBS elements identified in the row titles "System WBS."

The software engineering may include, at this level, elements of program management, configuration control, systems analysis, and CSCI integration and test as part of the software subsystem development. Thus, the second row of WBS elements allows for this resource expenditure data to be reported apart from the efforts associated with developing each CSCI.

3. Latest Month of Actuals

Enter the latest month after contract award or project start for which actual manpower data is available. This number should reflect the months after the date for contract award entered in Item 4.1 for the Contract Award milestone on the Software Development Project Summary Data Form.

4. Units of Manpower

Enter the units of measure used for the manpower figures, that is, man-hours, man-days, man-months or man-years. Indicate the number of hours that the unit is based on, if you are not entering man-hours.

In each of the subsequent rows enter the manpower expended during the month after Contract Award indicated in the left-hand column. Space is provided for up to five years of data. For ongoing projects enter the latest estimate of resource requirements for those months for which actual data is not available.

GLOSSARY

Ada Tasking

An Ada construct in which a program unit may operate in parallel with the main program.

Bottom-Up

A term usually used to describe design or testing strategies in which the lowest level components are designed or tested first, then the next level components, and so on, until the highest level component is designed or tested. For dynamic bottom-up testing, drivers for the individual or set of low level components must be built in order to provide their input and observe their output.

Central Processing Unit (CPU)

The CPU is the computer's main processor, as opposed to other potential co-resident special purpose (e.g., math handling, graphics handling, etc.) processors.

Code and Unit Test (CUT) Milestone

The milestone when the software coding and unit testing is completed, and the software is ready for integration.

Configuration Management

Includes task configuration identification, change control, configuration status accounting, and configuration auditing to ensure proper configuration control.

Contract Award

The point when the development is actually funded or the go-ahead for development is received. Some requirements work may have been done prior to this time, but not necessarily.

Correctness

Correctness is a software quality factor which indicates the degree to which software satisfies its requirements.

Critical Design Review (CDR)

A review conducted for each configuration item when the detail design is essentially complete to determine if the detail design satisfies the requirements established in the specification and to establish the exact interface relationships with other parts of the system.

Cross-Assembler

A computer program that accepts symbolic instruction mnemonics for a selected target computer and generates target computer machine code while hosted on another computer. A cross-assembler thus allows code written on one computer to be assembled on another. A symbolic language translator that operates on one type of computer to provide machine code for another type of computer.

CSCI (Computer Software Configuration Item)

An aggregation of computer software which satisfies an end-use function and is designated for configuration management.

Data Base

In its broadest sense, a data base is the complete collection of information (machine-readable data--both external data files and internal, hard-coded data items--and written documentation) associated with a software program. Generally, data base refers to a specific set of machine-readable data files.

Derived Type

An Ada object type whose operations and values are taken from an existing object type.

Design/Code, Walkthroughs/Inspections

A step-by-step, detailed examination of design/source code by a small group of qualified personnel.

Development Staff

Software design and software programming personnel.

Development Test and Evaluation Milestone

The milestone when software has been completely integrated with other software and hardware, and has passed system tests.

Development Test and Evaluation (DT&E)

Test and evaluation that focuses on the technological and engineering aspects of the system or equipment items.

Efficiency

Efficiency is a software quality factor indicating how productively the software uses its computer resources.

Formal Qualification Test (FQT)

A formal test conducted in accordance with approved test plans, descriptions, and procedures after a CPCI has been integrated to validate that each function of the CPCI satisfies the specified software requirements and applicable interface requirements.

Formal Test

A test which is conducted in accordance with test procedures approved by the procuring activity, is witnessed by an authorized representative, and is documented in a test report for procuring agency review.

Function Points

Functions points is a measurement and calculation technique used for estimating productivity. The technique calls for classifying and counting five types of functions: external inputs, external outputs, logical internal files, external interface files, and external inquiry types.

Functional Configuration Audit (FCA)

The formal examination of functional characteristics test data for a configuration item to verify that the item has achieved the performance specified in its functional or allocated configuration identification.

Functional Specifications

A specification of a component as a set of functions defining the output for any input. The specification emphasizes what the program is to do, rather than how to do it. However, an algorithmic specification can be considered functional if it is not used to dictate the algorithm to be used. Describe a system in terms of its principle functions and their interrelationships; i.e., the functional relationships of the parts.

Generics

Generics are Ada subprograms that can process parameters of more than one type. The classes of parameters that are acceptable to the subprogram are specified in a generic clause. Before using a generic subprogram, the types that it is to process are specified during generic instantiation.

HIPO (Hierarchy Plus Input-Process-Output)

A graphical technique that defines each component by its transformation on its input data sets to its output data sets. This part

the second part of the hierarchy chart is a list of hierarchical elements corresponding to the process blocks. The hierarchy chart is a tree of blocks, similar to an organization chart, showing each level of the division of functions. For each function or subfunction, an input-process-output chart, roughly similar to the block diagram, is included. It shows the inputs and outputs and the processes joining them. The acronym is Input-Process-Output. It is a graphic design technique. I/O diagrams describe functions in terms of the input to a process. They show a system, subsystem, or program functionally (i.e., the functions that it performs), answering the question "What does it do?" Since these diagrams are visual, they are easier to understand than most documentation which is narrative. Although flowcharts are another graphic design technique, they show organization and logic in contrast to function.

Instruction simulator

A computer program used to simulate the execution characteristics of target computer using a sequence of instructions of a host computer. The instruction simulator provides bit-for-bit fidelity with the results that would be produced by the target computer following the same operations and initial conditions.

Inquiry

In context with the function-points technique, an inquiry is a unique input-output combination, where an input causes and generates an immediate output, as an external inquiry type.

Interface

1. A type of a software quality factor describing the level of communication authorized access to operations and data.

Interface

2. In context of the function point technique, an (external) interface is a file posse, or shared between applications.

Interoperability

Interoperability is a software quality factor which indicates how well the system can interface with other systems.

Iterative refinement

Iterative refinement of software design. Iterative enhancement means developing the entire system with very little detail at first, and then adding more detail with each design iteration.

Maintainability

Maintainability is a software quality factor which indicates the degree of effort it takes to locate and fix an error. High maintainability means a low degree of effort.

Monitor

A monitor determines which of two or more processes competing for control, in order to execute has priority. It allows that which has priority to take control and execute and places the other process(s) on a queue to await their turn to take control and execute.

Operational

The status given a software package once it has completed contractor testing and it is turned over to the eventual user for use in the applications environment.

Operational Evaluation

The analysis of a system operating in its real-life environment.

Operational Test and Evaluation (OT&E)

Testing designed to give results related to the service environment in which the systems will be operating. It is accomplished by service operational and support personnel of the type and qualification of those expected to use and maintain the equipment. OT&E relates not only to technical suitability, but also operational effectiveness and suitability, including maintainability, reliability, training, and logistics.

Operational Testing

Performing tests on software in its normal operating environment.

Overloading Operator

Overloading an Ada operator means that it has been given more than one meaning or function. The particular meaning of the overloaded operator depends upon the type it receives at a given time.

Peak Staff

The maximum number of project and development staff during full scale development.

Physical Configuration Audit (PCA)

The formal examination of the "as coded" configuration of a CPCI against its technical documentation in order to establish the initial product configuration identification.

Portability

Portability is a software quality factor which reflects the maximum effort required to transfer an implemented system from one hardware or software system environment to another.

Preliminary Design Review (PDR)

A review prior to the start of the detailed design process to evaluate progress and technical adequacy of the selected design approach, to determine the design compatibility with the performance requirements of the CPCI development specification, and to establish the existence and compatibility of the interfaces between the configuration item and other elements of the system.

Preliminary Qualification Test (PQT)

A test conducted during the integration of a CPCI to evaluate the performance of those CPCI functions which are critical, as determined by time-critical or performance-critical requirements.

Procedural Specifications

A specification of a component in some algorithmic manner (e.g., using PDL or a flowchart). The specification says how the program is to work.

Program Design Language (PDL)

A design tool used to facilitate the translation of functional specifications into computer instructions. Intended to be comparable to the blueprint in hardware, programming design languages strive to communicate the concept of the software design in all necessary detail, using a formal or structured version of English.

Program Librarian

A program librarian is someone who is responsible for controlling all of the software and technical documentation pieces of a project. This role is usually associated with configuration management.

Program Management

Includes direct labor software management. It does not include hardware management highest level program management, etc.

Program Support Library

A software system which provides tools to organize, implement and control software development.

Proof of Correctness

Proof of correctness is a software testing technique in which the program is treated like a theorem. Usually the proof consists of symbolically evaluating all of the expressions from an input statement to an appropriate output statement, assuming a given input value. Alternatively, the proof can start from an output statement and evaluate the expressions backwards to an appropriate input statement. The purpose is to prove that the program both terminates and produces the expected output for a specific input domain.

Quality Assurance

Includes the quality engineering functions (ensuring that quality is built into the product and developing appropriate standards), and quality control inspection and audits.

Rapid Prototyping

Software rapid prototyping involves quickly building a system or model of the system, usually for the purpose of demonstrating feasibility, clarifying stated requirements, or trying out design concepts. The prototype may not be the basis of the eventual system (i.e., a throw-away, once the purpose is served) or it may be a skeleton of the eventual system.

Record

In the context of Ada, a record object type is made up of components that usually have different named types or subtypes.

Reliability

Reliability is a software quality factor which indicates how consistently an implemented system performs its intended function for its specified input domain.

Requirements

A system specification written by the user to define a system to a developer. (A statement of what the user (purchaser) expects the system to include among its capabilities.)

Requirements Analysis

Analysis performed to assure that the developer's software requirements are completely and correctly defined. As part of this activity, analysts check each requirement for consistency with other requirements and trace software requirements to their source.

Requirements Specification

Translation of an operational (or application) requirement into a statement of the functions to be performed.

Requirements Specification Language

A language used to specify a software system which is sufficiently formal in the mathematical sense, that conclusions concerning consistency and completeness may be drawn from the system's specifications expressed in such languages.

Reusability

Reusability is a software quality factor which indicates how easily a software unit or system can be used in another application.

SLOC (Source Lines of Code)

The SLOC represents the number of "card-image" lines of compilable source code. It does not include comments; they are not compiled. It does include data declarations. If a single programming statement takes more than one line to express, then each line is counted separately. However, if more than one programming statement is entered on the same line, each statement is counted as a line (since a compiler treats the statements separately).

Software Design

This includes the definition of the software architecture to implement the software requirements, the preparation of architectural design specifications, design reviews, the layout of physical data structures, interfaces, and additional design details to implement the requirements.

Software Programming

This includes the actual coding, unit testing, maintaining appropriate unit documentation, and test driver development for the individual software modules/units.

Software Specification Review (SSR)

A review to demonstrate to the contracting agency the adequacy of the Operational Concepts Document, Software Requirements Specification, and, if applicable, Interface Requirements Specification(s). Specific details regarding the SSR process are contained in MIL-STD-1521.

Software Test

Includes preparing test plans and procedures, running tests, and preparing test reports. This includes software related test only.

Specification-Driven Testing

Input test data derived for the purpose of testing each specified requirement or design element is known as specification-driven testing.

Staffing Rate

The rate at which people can be added to the project per year.

Static Analysis

The analysis of a program without executing the program. Specific methodologies include desk checking, peer code review, and structural analysis.

Structure-Driven Testing

Using input test data derived for the purpose of testing each logical path in a program.

Structured Design

Design technique that involves hierarchical partitioning of a modular structure in a top-down fashion, with emphasis on reduced coupling and strong cohesion.

Structured Code

Structured code usually indicates that only well-behaved control constructs are used. This means that logic is forward flowing and each control block has a single entry (and possibly a single exit). In Ada, all control constructs, except the goto statement, represent structured code. The use of the exit statement, however, does introduce a multiple exit construct for the loop control block.

Structured Programming

The activity of programming with a limited set of constructs. The key constructs in structured programming are: (A) each program is allowed only one entry and one exit. (B) only three basic control structures are sufficient: do-while, if-then-else, else, and sequence. (C) other sequences are sometimes allowed, the most popular ones being do-until and case. (D) The restricted constructs are often augmented with the following practices: * hierarchical and modular block structures * limits on the size of modules * indentations and formatting a system design, implementation and computer programming technique encompassing the following concepts: (1) Top-down design in which overall program logic is designated first, each major component before any of its subcomponents, etc. (2) Chief programmer team managerial approach to program production incorporating as a nucleus a chief programmer, a back-up programmer, programming secretary and

defined relationships among any additional specialists. (3) Top-down programming in which overall program logic is coded and tested before any of its subcomponents, etc. (4) Programming using only the three logic structures of a simple sequence of two or more operations; a conditional branch to one, or more operations and a return (if A, then B, else C); and a repetition of an operation while a condition is true (do-while). (5) Programming with limited or no "Go To" logic. (6) Picture-on-a-page technique in which the overall program logic is represented on the first page, each major component is represented on a subsequent page, each subcomponent on a still later page, etc.

Structured Requirements Analysis

An analysis technique that involved hierarchical partitioning of the requirements in a top-down fashion, with emphasis on functionality.

System Design Review (SDR)

A review conducted when the definition effort has proceeded to the point where systems requirements and design approach are more precisely defined. The review ensures that there is a technical understanding between the contractor and the procuring agency on the system segments defined in the system specifications and the configuration items defined in the configuration item performance specifications.

System Requirements Review (SRR)

A review conducted when a significant portion of the system functional requirements have been established to determine the adequacy of the contractor's efforts in defining system requirements.

Terminal Response Time

Terminal response time is the elapsed wallclock time from the moment the return or enter key is hit until a response is shown on the terminal's screen.

Testability

Testability is a software quality factor which indicates the maximum effort required to ensure that the system performs its intended functions.

Top-Down

Top-down is the reverse design or testing strategy of bottom-up. The top-down approach calls for designing or testing the highest level components first, and working downward toward the lowest level components. In top-down testing, stubs (i.e., nearly vacuous components that merely model the actual components) must be built in order to simulate the values that the lower level components would return, if invoked.

Turnaround Time

Turnaround time is the elapsed wallclock time involved in a particular activity. If compilation turnaround time is of interest, it would be the elapsed time required from the moment the source code is made available to the compiler until an indication is given that compilation is complete.

Usability

Usability is a software quality factor which indicates the level of human engineering requirements for the system. These requirements include the maximum time and effort required to learn the human interface, prepare input, and interpret output of the system.

Validation

The act of confirming that the design specifications and contractual commitments have been met and that operational capabilities of the ship/systems have been demonstrated to be satisfactory.

Virtual Machines

The complex of software and hardware that the software being developed calls upon to accomplish its tasks.

Walkthrough

A walkthrough is a technique of desk-checking a completed item. It usually refers to the manual analysis of source code as a review check. It is often undertaken by an independent analyst, not the author of the code.

APPENDIX D
DATA COLLECTION PLAN

INTRODUCTION

This plan defines both the general approach and specific requirements for the Ada software data collection effort. This effort will provide more knowledge about the actual costs, schedules, and development environments of Ada-related projects and expand the ESD/ACC software data base.

Data will be collected using the data collection forms created during this research study (Appendix C of the Task One ESD Ada Software Cost Study). These forms are an enhancement of the software data collection forms currently being used by ESD. They are designed to collect the data requested by the original forms, and additional data that are felt to be of value in future modelings of the costs and schedules of Ada software projects.

The forms have also been somewhat reordered to group data that apply to a specific phase or topic. The reordering makes the individual forms more flexible in their ability to be tailored to a particular software development.

The data to be gathered will help calibrate existing software cost models and provide guidance for potential new model development. Additionally, the data will provide insights into Ada project management.

Actual data collection will be performed during SPO contacts, on-site interviews, document reviews, and evaluation. The on-site data collection will ensure that data are collected consistently and completely, within the definitions and scope intended. The data will be collected according to the procedures outlined in CR-0136, "Procedures for Future Data Collection and Update," a document produced under Tecolote's ESD Software Database Expansion task.

The Tecolote data collection will be coordinated with the two SBIR contractors that will be collecting similar data in order to develop an Ada cost model. ACC will be the focal point of the coordination. The purpose of the coordination will be to ensure that we all operate as a team and provide the government with the best possible return for its research dollar.

Candidate projects that were identified during the research study are identified in this plan.

SCOPE OF THE DATA COLLECTION EFFORT

Candidate software development projects must be Ada-related, either by using Ada as the implementation language, using Ada tools and methods, or as prior non-Ada implementations of current Ada developments (such as the previous implementation of a simulator that is currently under reimplementation in Ada).

Both large and small Ada-related programs are included as candidates, although more large projects are preferable. Actual developments, with typical personnel and functional deliverables, are also highly desired candidate project traits. Data from several experimental projects (highly motivated teams of experts who are constantly being measured) will be collected. This data will be specially identified to ensure they are used with caution in cost model calibration.

DATA COLLECTION APPROACH

Most data collection will be performed on-site, at the developer facilities. Additionally, some data will be collected from SPOs (or other appropriate organizations for non-Air Force projects), both to save time and to determine how it correlates with detailed project data. Experienced software professionals/cost analysts will perform the on-site data collection. This will ensure experimental project shortcuts are identified and reduce the potential of contractors providing unreasonable data.

Data collection will be performed as follows:

- Arrange visits with program offices.
- Prepare program overviews before on-site visits.
- Collect initial data.
- Validate initial findings.
- Clean up initial data.

Additionally, on incomplete projects where data have been collected, arrangements for final data to be collected upon project completion will be made. This is in accordance with the procedure for the Software Data Base Expansion.

These steps are detailed in the following sections.

3.1 ARRANGE VISITS WITH PROGRAM OFFICES

SPO contacts will be made as a matter of courtesy, to help ensure contractor cooperation during data collection and as a primary source of project information. First we will contact the SPOs via letter, introducing ourselves, explaining the project, and requesting cooperation in the data collection activity. The letter will be written and signed by the appropriate ACC representative. This will be followed with a telephone contact. Over the telephone, we will reintroduce ourselves, explain the

data collection goals, and clarify the information contractors should assemble before our on-site visits.

The usefulness of the data collection to the Air Force and the SPO's mission will be emphasized. We will refer SPO personnel to our Air Force contact if necessary to resolve any concerns. Additionally, we will inform the SPOs of our non-disclosure agreement with ACC and communicate our willingness to sign additional non-disclosure agreements.

3.2 PREPARE PROGRAM OVERVIEWS

These program overviews will ensure the scope and goals of identified projects are understood before on-site visits. Project case files will be created to ensure data is well organized and that researchers have project familiarity before the on-site contractor visits.

3.3 COMPILE PROJECT CASE STUDY FILES

A case study file will be prepared for each candidate program. These files will be updated as additional information becomes available throughout the data collection process. The following information will be included as available:

- Program name and major hardware elements
- Program managers
- Program mission/Application type
- Development schedule
- Related programs or projects
- Program goals (production, experimental, etc.)
- Contractor's Ada experience level
- Contractor and subcontractor names
- Ada environment summary (such as Rational or VAX)
- CSCI names, numbers, and sizes
- RDT&E funding profiles
- History of program modifications
- Project-level data collection forms
- Other pertinent data

3.3.1 PROGRAM OVERVIEW INFORMATION SOURCES

The sources that will be accessed to develop the program overviews are:

- Program management directives (ESD or others when applicable to the project).
- RDT&E summaries of budgetary submits.
- ESD or other applicable cost library documents dealing with recent cost estimates.
- Telephone contacts with knowledgeable ESD, MITRE, and other appropriate personnel.
- Telephone conversations with SPOs or other Government personnel.
- Computer Resource Integrated Support Plans (CRISPS) where possible to provide major software information.

3.3.2 VISITS TO SPO TO COLLECT PROJECT FILE DATA

If visits to SPO are required to obtain the requested data, we will attempt to arrange them to minimize the number of trips. These visits will be made if SPOs cannot provide necessary project file data via mail, or if we feel a visit will allay SPO misgivings about this effort.

3.4 ARRANGE CONTRACTOR VISITS

Each specific contractor will be contacted after the appropriate SPO has approved the data collection. During the contractor contact, we will set appointment dates. Appointments will be coordinated so we collect data on projects that are farther along the development cycle first. This will give other projects more time to develop before we do data collection. We will plan one to two weeks on-site per project, depending on the project size, development cycle phase, contractor cooperation, and likelihood of collecting meaningful data.

Second visits may be recommended for projects that are useful data sources where more actual (rather than estimated) data or results of recent changes are deemed valuable during the first visits.

3.4.1 CONTRACTOR PROPRIETARY DATA SECURITY

Non-disclosure agreements can be executed with specific contractors if needed. Some entries in the form may be sanitized to ensure contractor confidentiality. The contractor will be asked to indicate which data are proprietary. The Air Force will receive a master data set with original data.

3.4.2 DISTRIBUTION OF SANITIZED DATA

We recommend ESD/ACC offer participating contractors sanitized versions of the data collected from all projects during this task. We believe this will provide an extra incentive for cooperation. If ACC agrees, we will include this offer in our initial contacts. We will assure contractors that only sanitized data will be distributed.

3.5 COLLECT DATA AT CONTRACTOR SITES

3.5.1 LOCATE DATA SOURCES ON-SITE

We will identify the following personnel and data sources:

- Program managers to provide the project level information, goals, personnel attribute information, and to coordinate with other project personnel.
- Software CSCI-level managers to provide more detailed size, difficulty, personnel, and environment data.
- Software engineers involved with each CSCI who can provide us with the detailed data and assist in our independent document evaluation.
- Quality assurance personnel, testers, and others who have performed independent project analysis and reviews.

- Actual plans, software specifications, manuals, listings, unit development folders, and other information for review.

3.5.2 INTERVIEW APPROPRIATE PERSONNEL AND REVIEW DOCUMENTATION

3.5.2.1 Program Manager Interviews

Our first contractor site interview will be the program manager for one to two hours, filling in top-level project data, verifying any unclear information within our project file and gaining commitment for arranging meetings with CSCI level managers. If the project is too large to allow data collection on each CSCI, we will identify the most appropriate CSCIs for data collection.

3.5.2.2 CSCI Level Manager Interviews

Appropriate CSCI level managers will be interviewed next. Each initial interview will take about two hours. During these interviews, we will acquire more detailed size, difficulty, personnel, and other data. Additionally, we will coordinate our interviews with software engineering and technical personnel primarily through these CSCI level managers. We will interview CSCI level managers again to fill in any missing data or resolve inconsistencies in data obtained from technical personnel.

3.5.2.3 Technical Personnel Interviews

During these technical reviews, we will collect detailed CSCI level data. Additionally, we will validate any questionable data received from other interviews. The combination of software engineers actually doing the work, testers, quality assurance, and other personnel should provide us with a well-rounded, accurate picture of development.

3.5.2.4 Review CSCI Documentation

During our documentation reviews, we will validate information received from project personnel. Additionally, we will review documents for additional insights. For example:

- The Software Development Plan review should provide significant data.
- Design specification reviews should show the use of design level tools and methods.
- Code reviews of chosen packages should provide indications of Ada conventions used.
- Unit development files may provide a wide variety of information.
- Test plans, procedures and reports (and software problem and change reports) may provide information regarding the ease of test and number of errors.

3.5.3 RECORD DATA IN DATA COLLECTION FORMS

Most data will be entered into the collection forms during the interviews and reviews. Some details may be filled out in between interviews to ensure respondent's time is optimized.

After each interview, the major data sources will be noted.

Post-interview reviews by data collection personnel may highlight areas where more data should be captured or where clarification of inconsistencies is required. If so, we will attempt to clarify the information while on-site.

3.5.3.1 Estimated Data Collection Versus Actual

Since project data will be collected at differing points within the development cycle, some data may be estimated, rather than actual. In those cases where a range of data is appropriate, we will collect the minimum, most likely, and maximum estimates of these data. The minimum and maximum case estimates will be verified with both management and technical personnel when appropriate. All estimated data collected will include information identifying the source.

3.5.3.2 Data Sanitizing

If desired, data will be sanitized after the on-site visit is completed. Sanitation will remove information allowing specific project identification. In order to allay contractor concerns about adequate sanitization, a completed copy of the data collection forms will be provided to the contractor. The contractor will then indicate which data are proprietary.

3.6 VALIDATE FINDINGS VIA INDEPENDENT PROJECT OR SPO INTERVIEWS

After the data collection forms are completely filled out, we will attempt to verify the accuracy of high-level data independent of the contractor. Validation will concentrate on areas that appear to be outside of norms, overall project goals, and the specific Ada cost drivers issues and impacts.

3.7 CLEAN UP DATA AND TEXT DEFINITION

During this step, staff will ensure forms are neatly filled in without blanks, that the cross-referencing between forms is appropriate, and that any textual descriptions within the forms are complete, cohesive sentences. The forms will be suitable for a clerk to enter the data into an automated data base. If we feel it will add clarity, we will: (1) include a second copy of the form with relevant text or qualifying information; or (2) add a separate sheet of caveats at the end of the form.

CANDIDATE ADA PROGRAMS

The following lists candidate programs for data collection. To date, project sources have been reluctant to provide much information without a formal introduction from the Air Force. The usual reason given was the high visibility of the early Ada projects and the desire to minimize the number of requests for information. There is a concern that the number of potential data requests can interfere with the Ada projects' progress. Thus, when ACC approves of this data collection plan and provides a letter of introduction, the state of the various projects can be assessed and a priority list can be generated.

There are many non-Air Force projects on the project list, and they should be considered for data collection to ensure we obtain as much information on Ada costs as possible. Hence, ACC should explore the possibility of having the AJPO support this effort with their own cover letter. Additional support could come from RADC, SSCAG, and other high-level organizations. The more official backing the project has, the easier it will be to gain the cooperation of government managers and private contractors.

Once the data collection task begins, the SPOs' and contractors' willingness and enthusiasm for data collection will be factored into the candidate list. Data relating to previous implementations of current Ada developments will be given priority to contrast with the new Ada systems.

A few non-Ada projects are in the list. These are the original functions that are being reimplemented in Ada at SIM SPO.

We are still working to identify additional projects. As more information becomes available, it will be provided.

4.1 ASD/SIMSPO REBUILD - BOEING

Summary: Rebuild of existing simulator for Ada data collection purposes.

Program Manager: Bill Lloyd

Telephone Number: (513) 255-7177

Project Phase: Past PDR

4.2 ASD/SIMSPO REBUILD - PERTEC

Summary: Rebuild of existing simulator for Ada data collection purposes.

Program Manager: Bill Lloyd

Telephone Number: (513) 255-7177

Project Phase: Past PDR

4.3 NSA MINSTREL - GTE ROCKVILLE

Summary: NSA project developed by GTE. Originally targeted for language development. Language was changed to Ada project after contract award. Ada training is currently underway.

Telephone Number: (301) 294-8603 ???

4.4 ASD ECSP0 1750 COMPILER

Summary: Compiler with real time, size and performance constraints under development. Project is in trouble. SPO personnel asked that we not collect data.

Program Manager: Robert Earnest

Telephone Number: (513) 255-5945

4.5 ROME AIR ADA-INTEGRATED ENVIRONMENT

4.6 MILSTAR GROUND TERMINAL - LOCKHEED

Program Manager: Colonel Lindberg

4.7 NSA - GTE MOUNTAIN VIEW

Summary: 10,000 lines developed to NSAM standards. Recently completed under budget.

Program Manager: Margaret Meseimans (GTE Mountain View)

Project Phase: Complete

4.8 SIG ADA EXPERIMENT

Program Manager: Tony Alben (TRW Contact)

Telephone Number: (213) 535-1624

Project Phase: Completed

4.9 WORKSTATIONS - INTELLIMAC ROCKVILLE

Summary: Developed Ada environment workstations.

Program Manager: Dave Dikel or Mr. Richardson

4.10 GOULD

Program Manager: Bob Thibaeau (Gould Fort Lauderdale)

4.11 JPL

Program Manager: Ed Colbert (Absolute Software Consultant)

Telephone Number: (213) 545-0567

Project Phase: Various

4.12 SofTech

Summary: SofTech developed the Ada Language System (ALS), which is a compiler and APSE for the VAX family. The ALS was sponsored by the Army. SofTech also developed the Ada Compiler Validation Suite (ACVS), a set of about 4000 small Ada test programs.

4.13 MAGNAVOX FORT WAYNE

Summary: Ada project was developed to assess 2167 Ada impacts.

4.14 WIS PROJECTS

The government contact is Lt. Jeff Siegal.

4.15 PHASE 2 SECURE OPERATING SYSTEM - TRW

4.16 ARMY ALBUQUERQUE PROJECT

4.17 MILSTAR (IBM PORTION)

4.18 NASA Langley

Summary: Note Ed Dean has been out of town for over a month. He is now back. Galorath made one contact and should make another to get project information.

4.19 CCPDSR

Summary: About 200-300K lines replacement

Program Manager: ESD Steve Patey or Colonel Yonkers

Project Phase: The RFP will be released in September. We were asked to make contact then.

4.20 LIST OF ADA PROJECTS IN THE ADA INFORMATION CLEARINGHOUSE NEWSLETTER

Ada-AIMES
Ada Compiler System (ACS)
Ada Designed/x.25/VLSI/VHSIC Chip
AdaEDIT
Ada Language System
Ada SAM Missile Simulation
Ada Test Tools
Advanced Field Artillery Tactical Data Systems (AFATADS)
Air Force Support to MEECN
Automated Test Procedure Generator for Ada
Concurrency Control Method for Database Indices (MU)
Design Evaluation Tool (DET)
E-48 Message Processor System
Flexible Ada Simulation Tool (FAST)
Flir Mission Payload Subsystem (FMPS)
GRAMACT
Intermediate Forward Test Equipment (IFTE)
Maneuver Control System (MCS)
Mobile Information Management System (MIMS)
MSOCC Ada Study
NOSC Tools

Regency NET
RELATE/3000, Project Alert
Relational Database System
Single Channel Objective Tactical Terminal (SCOTT)
817 Fuze Tester

4.21 ADA JOINT PROGRAM OFFICE

AJPO has responded to our request for information on projects with a list that includes project names, descriptions, sizes, and points of contact. AJPO requested that we restrict distribution of this list, so it is not included in this report. ESD/ACCR has a copy of the list.

END

12-87

DTIC