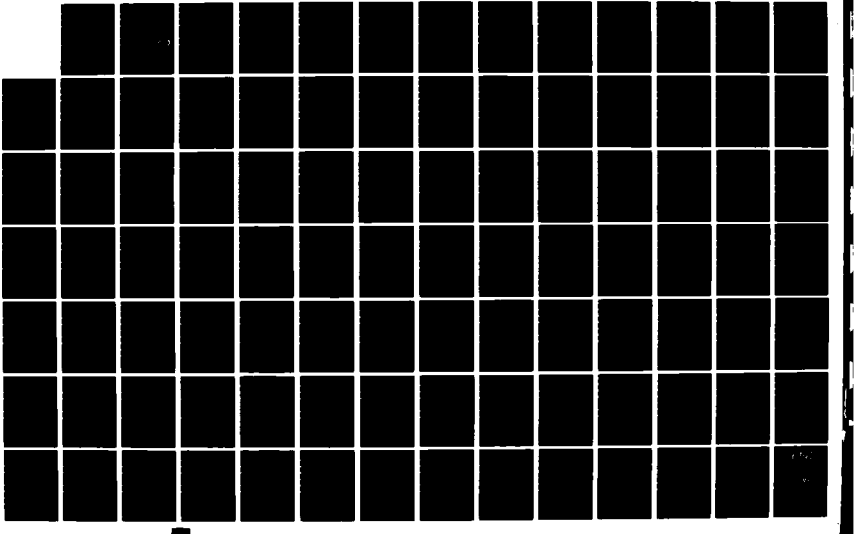
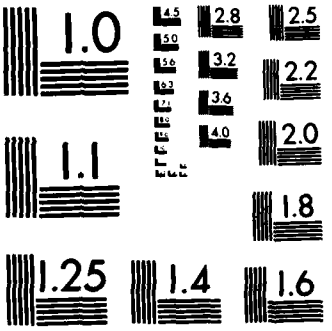
AD-A154 402 SURVEY OF ADA (TRADEMARK)-BASED PDLS (PROGRAM DESIGN LANGUAGE)(U) COMPUTER TECHNOLOGY ASSOCIATES INC MCLEAN VA JAN 85 N00163-84-C-0300 1/1

UNCLASSIFIED F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: N00163 - 84C - 0300
2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)
Survey of Ada*-Based PDLs
Final Report
(January 1985)
*Ada is a registered trademark of the U.S. Government-Ada Joint Program Office

5. TYPE OF REPORT & PERIOD COVERED

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)
Computer Technology Associates, Inc.
Advanced Software Methods, Inc.

8. CONTRACT OR GRANT NUMBER(s)

9. PERFORMING ORGANIZATION NAME AND ADDRESS
Computer Technology Assoc., Inc.
7927 Jones Branch Drive
Suite 600W
McLean, VA 22102

Advanced Software Methods
17021 Sioux Lane
Gaithersburg, MD 20878

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

11. CONTROLLING OFFICE NAME AND ADDRESS
Naval Avionics Center
21st Street and Arlington Avenue
Indianapolis, IN 46218

12. REPORT DATE
January 1985

13. NUMBER OF PAGES
86

14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)

15. SECURITY CLASS. (of this report)
Unclassified

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

DTIC ELECTE JUN 3 1985 S D A

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Unclassified

18. SUPPLEMENTARY NOTES

Final Report

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Ada Programming Design Language, Environmental Tools, AJPO, STARS NAC, PDL, SYNTAX, Ada-based, Methodology, ADA/SDP, ADAP, BTO Ada PDL, JPDL, LTH Ada PDL, PDL/Ada, RN ADAP, SAIC Ada PDL, Sanders Ada/PDL, TI Ada PDL, TRW Ada PDL, WIS Ada PDL, X Ada

DTIC FILE COPY

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The Naval Avionics Center (NAC) has published the results of its Programming Design Language (PDL) survey in the "Survey of Ada-based PDLs, Final Report, January 1985." A PDL is an English-like artificial language, sometimes called psuedo-code, used in documenting design logic in software systems development. Computer Technology Associates and Advanced Software Methods were contracted by the NAC to conduct a survey of Ada-based PDLs to update the 1982 NAC survey of Ada-based PDLs.

With the development and advancement of the Ada Programming language, many Ada-based

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

PDLs have been developed. These PDLs are designed to make full and correct use of the Ada language features and to employ the software engineering concepts contained within the Ada language.

The purpose of the 1984 NAC PDL survey was to identify and investigate the current set of Ada-based PDLs, updating the existing data collected in the 1982 survey. In addition, the results will provide a foundation for an Ada-based PDL evaluation and selection procedure that is being developed by the STARS Methodology Project.

The 1934 NAC PDL survey gathered information on 28 PDLs: 7 in the very early stages of development, 5 were disqualified because they failed to meet the requirements to be classified as "Ada-based" and 16 are analyzed in detail in the Final Report.

SURVEY OF Ada [TM]-BASED PDLS

FINAL REPORT
(January 1985)

Prepared for:

Naval Avionics Center
21st Street and Arlington Avenue
Indianapolis, IN 46218

Under contract:
N00163-84C-0300

By:

Computer Technology Associates, Inc.
7927 Jones Branch Drive, Suite 600W
McLean, VA 22102

and

Advanced Software Methods, Inc.
17021 Sioux Lane
Gaithersburg, MD 20878

ACKNOWLEDGEMENTS

The authors extend their gratitude to all those people who sent information concerning their PDL and patiently answered many questions. Their contributions were invaluable for the success of this survey.

They would also like to thank Becky Shannon (Computer Technology Associates) for her help in drafting all the tables in this report.

Finally, a special word of thanks goes to Dr. Lindley for his support and guidance during the course of the survey, and for his help in reviewing earlier drafts of this report.

TABLE OF CONTENTS

Byron™ is a registered trademark of Intermetrics, Inc.

TABLE OF CONTENTS (continued)

LIST OF TABLES

EXECUTIVE SUMMARY

The development of the Ada Programming Language has been followed by a profusion of Ada-based PDLs. These Ada-based PDLs are designed to fulfill several objectives. The primary objective is to maximize the potential benefits offered by correct use of both the Ada language features and the software engineering concepts inherent to Ada. Other technical concerns include easing the translation of detailed designs into Ada code and supporting learning of Ada itself. There is increasing evidence that the Air Force, Army, and Navy are actively supporting the use of these Ada-based PDLs. Consequently, another factor influencing the development of these PDLs is the need to be responsive to government standards and requirements.

The purpose of this survey was to identify and investigate the current set of Ada-based PDLs, updating an earlier NAC survey conducted in 1982. It also provides the foundation for an Ada-based PDL evaluation and selection procedure that is being developed by the STARS Methodology Project.

In this survey, information was acquired on 28 PDLs. Of these, 7 are in the very early stages of development and insufficient material is currently available to permit a detailed analysis. A further 5 PDLs are not considered to fit the categorization of "Ada-based." The remaining 16 PDLs are analyzed and discussed in some detail.

By and large, the Ada-based PDLs are intended for use in both preliminary design and detailed design activities. While a few are little more than Ada used for design purposes, the majority provide extensions that encourage expressing a design solution to a problem rather than a premature implementation. However, there is still little agreement as to the types of supplemental design information that should be captured by an Ada-based PDL. Some PDLs concentrate on describing interface specifications, while others address issues of requirements tracing and the specification of design constraints. In this area, the PDLs appear to be diverging.

In terms of coverage of Ada syntax and semantics, most Ada-based PDLs support full Ada but recommend deferring the use of certain features until the design text is translated into Ada code. In those cases where a PDL is expected to be used to develop software that is implemented in a language other than Ada, the developers generally provide guidelines illustrating use of the PDL in this context.

The Ada-based PDL user can expect more automated support than is customarily seen for design activities. Only a few tools are already available, but many of the PDL developers plan to extend this set within the next couple of years. Most of these tools address document and report generation, or are oriented towards the later part of the software life cycle.

There are relatively few plans to automate requirements tracking or design analysis. Consequently, while there is some improvement in the extent of the automated support provided, it does not match the potential of these PDLs. This can be attributed to the apparent desire of the Ada-based PDL developers to produce PDLs that are independent of any particular design method and so can be used with several. This attitude will have to change before more comprehensive automated support can be developed.

The effort so far has focused on defining the syntax and semantics of the Ada-based PDLs. While some attention is being paid to how these PDLs fit into the larger picture of software development and post-deployment support, this is not addressed extensively. Instead, it seems to be a reflection of the need to meet government standards and procurement practices, This means that the developers are making new PDLs available, and generally good ones, without providing much guidance in their use. This is not necessarily a criticism of the developers. It was necessary to start by focusing on one area of concern. Now that the syntax and semantic aspects are starting to stabilize, it is time to address the larger issues of design.

At the current time, it is difficult to single out any one PDL as being superior. A PDL which is strong in some respects is often weak in others. Therefore, a user seeking to adopt an Ada-based PDL is encouraged to select one as a basis, and then transplant the additional features needed from other PDLs. Moreover, new Ada-based PDLs continue to be developed and some of these may offer significant advances. Consequently, users are advised to review new PDLs as they become available.

In summary, this is an active field where much progress has been made in the last two years. Already these Ada-based PDLs do much to increase the visibility and accuracy of software designs. However, if there are to be radical improvements in software quality and cost, this progress must be continued. The next step is to put these PDLs back into the software life cycle and look at the software design process as a whole.

1.0 INTRODUCTION

The Ada-based PDL survey summarized in this report was performed for the Naval Avionics Center (NAC] under the direction of Dr. Lawrence Lindley. The survey was conducted by Dr. Paul Baker of Computer Technology Associates, Inc. and Mrs. Christine Youngblut of Advanced Software Methods, Inc.

Over recent years, Program Design Language (PDL) has gained wide acceptance as the preferred technique for expressing detailed software designs and there are many different types of PDLs in existence. All offer a mechanism for structuring the design of software, documenting the design so that it can be effectively criticized in reviews, and guiding the production of code.

The advent of the Ada Programming Language has had considerable impact on PDLs. Ada is a significant advance over previous higher order languages and offers many improvements for the software development and maintenance process. However, people are becoming increasingly aware that these potential benefits are unlikely to be realized in the absence of similar improvements in the methods used to develop software prior to the code and unit test phase. Consequently, the technical community has invested much effort in the development of PDLs that are specifically oriented towards the Ada language, or Ada-based PDLs.

In addition to encouraging the use of contemporary software engineering practices in design activities, these PDLs offer several other advantages. The similarity of the PDL syntax to that of Ada increases the ease of transitioning from the detailed design to Ada code. The software developer only has to master one language rather than two. Additionally, learning the Ada-based PDL supports learning Ada itself, and vice versa.

Various DoD organizations are encouraging the use of these PDLs. Until Ada compilers become more readily available, Ada-based PDLs present the best opportunity for complying with draft directive 5000.31. Indeed, the Army mandates the use of Ada-based PDL for the development of all management information systems (MIS) and mission critical systems, and the Air Force and Navy have mandated it for specific programs. Professional organizations are also active in this field. The IEEE has a working group developing a recommended practice for Ada-based PDLs [1] and the ACM has a special interest group devoted to Ada issues.

In order to have an early assessment of Ada-based PDLs, the NAC commissioned an initial survey in 1982 [2]. At that time, the advent of Ada had just begun to impact the technical community and only a few Ada-based PDLs were available. Since then, many more Ada-based PDLs have been developed and some of the earlier Ada-based PDLs have been modified and extended. This growth in the field has prompted the NAC to commission a new study. This report presents the results of that study.

The next section of this report describes how information about Ada-based PDLs was solicited and summarizes the response of the technical community. Section 3 identifies the characteristics that were used to determine whether a particular PDL can be classified as "Ada-based". This section also outlines the full set of characteristics that were used to analyze and compare these PDLs in a consistent manner. The fourth section presents detailed descriptions

of the Ada-based PDLs that were identified in the survey and the following section summarizes these descriptions in the form of a comparison. Section 6 briefly discusses those PDLs that are not included in the preceding description and comparison sections. These PDLs are either not yet fully developed or not classified as Ada-based.

The final section, Section 7, terminates the report with a discussion of the conclusions and recommendations that have arisen from the survey.

2.0 SCOPE OF THE SURVEY

The present survey is intended to be as inclusive as possible and every effort has been made to identify all current Ada-based PDLs. Each Ada-based PDL is evaluated from its written documentation; none are reviewed on the basis of interviews or outlines of forthcoming documentation. Consequently, the survey reflects those Ada-based PDL that are in use today.

This section describes how information was collected. It lists the sources that were used to identify the developers of Ade-based PDLs, describes how these people were contacted, and outlines the response received.

2.1 Identification of Ada-Based PDLs

To ensure completeness, several sources were consulted to compile a list of Ada-based PDLs. These were:

- Ada Programming Design Language Survey, Qctober 1982,
- A list of Ada-based PDL developers provided by the Naval Avionics Center,
- A notice placed in Commerce Business Daily,
- A notice placed in Ada Letters, and
- The matrix of Ada Design Language (DL) developers printed in the December issue of Ada Letters [3].

In total, fifty-one (51) potentially Ada-based PDLs were identified. A contact from the organization responsible for its development was found for each PDL. Letters were sent to these contacts requesting the following information:

"... detailed information concerning both the PDL and any support tools. Instruction manuals and reference manuals should provide a suitable level of detail. In particular, we require information on the semantic and syntactic structures of the PDL, the Ada language features which are supported, and the software methodology which the PDL supports. Examples of the PDL and the Ada code developed from the PDL would be of great value."

Table 2-1. Identification Aspects of PDLs Discussed in Report

PDL Abbreviation	Full Name (where applicable)	Organization
ACK	Ada Construction Kit	Edward F. Hoover, III
ADA/SDP	-	Mayda Software Engineering IL, Ltd.
ADAP	Ada Program Design Language	SofTech, Inc.
AIM/SEM	Ada Integrated Methodology/System Encyclopedia Manger	ISDOS, Inc.
ANNA	Annotated Ada	Stanford University
ATS Ada PDL	-	Advanced Technology Systems
BTO Ada PDL	-	Bell Technical Operations
Byron	-	Intermetrics, Inc.
CAEDE	Carleton Embedded Systems Design Language	Carleton University
CFG Ada PDL	-	Caine, Faber & Gordon, Inc.
GDADL	General Dynamics Ada-Based Design Language	General Dynamics, Pomona Division
Harris Ada PDL	-	Harris Corporation, GISD
Hazeltine Ada PDL	-	Hazeltine Corporation
IDL Ada PDL	-	IDL Tech
JPDL	JAMPS Program Design Language	U.S. Air Force Systems Command
LTH Ada PDL	-	LTH Systems, Inc.
PDL/Ada	-	IBM Federal Systems Division
RCA Ada PDL	-	RCA, Government Systems Division
RN ADAP	Regency Net Ada Program Design Language	SofTech, Inc.
SAIC Ada PDL	-	SAIC Comsystems Division
Sanders Ada/PDL	-	Sanders Associates, Inc.
SDC Ada PDL	-	SDC, A Burroughs Co.
TAP	Testable Ada Program Design Language	Reifer Consultants, Inc.
TI Ada PDL	-	Texas Instrument, Inc.
TRW Ada PDL	-	TRW
WIS Ada PDL	WWMCCS Information System (WIS) Ada PDL	WWMCCS, Hanscom AFB
XAda	-	GTE Sylvania Systems Group
York Ada PDL	-	York University, Great Britain

Some initial responses identified PDLs in the very early stages of development where written documentation is not yet available. In these cases, the contacts were encouraged to provide brief notes indicating type of PDL being developed.

2.2 Response to the Survey

Of the fifty-one people contacted, replies were received from thirty-nine (39). This is a response rate of approximately 76 percent.

Eleven (11) of these people indicated that their organization is not currently developing an Ada-based PDL. Included in these organizations are Norden Systems and the Ford Aerospace Company. Both these companies had previously engaged in Ada-based PDL development, but they are no longer supporting those efforts. Two others, Hughes Aircraft Company and the Naval Ocean Systems Center, are using the Byron Ada-based PDL developed by Intermetrics. The U.S. Army Communications and Electronics Command (CECOM) is not developing its own Ada-based PDL, but is concerned with evaluating and monitoring the PDLs developed by others. CECOM is currently undertaking an effort to develop Ada-based PDL guidelines [4].

The remaining twenty-eight (28) respondees provided information on their PDLs. Of these, seven (7) PDLs are still under development and insufficient information is available to permit a full analysis. Another five (5) do have sufficient documentation but do not fall under the classification of Ada-based. This leaves sixteen (16) Ada-based PDLs for which documentation was received, and these are the PDLs that are discussed in detail in this report.

The above twenty-eight PDLs are identified in Table 2-1. The names and addresses of the developing organization, and a contact at that organization, are listed in Appendix A for each of these PDLs. The documentation received for fully developed Ada-based PDLs and non Ada-based PDLs is identified in Appendix B.

3.0 PREPARATIONS FOR ANALYZING THE PDLS

In this survey, comparisons are drawn between members of a particular group of PDLs, namely those PDLs that have been specifically designed to support the development and maintenance of Ada software. Prior to beginning a survey of this nature, it is necessary to ask the following questions:

- How is group membership defined?
- Are the group members sufficiently alike to permit reasonable comparisons?
- Which characteristics reveal the relative strengths and weaknesses of the PDLs?

This section addresses these issues. It identifies two PDL characteristics that are sufficient to determine whether a design language is an Ada-based PDL, and a set of characteristics that permit describing and comparing these PDLs.

3.1 Classification of Ada-Based PDLs

A careful review of the current field of PDLs shows that there are two characteristics that can be used to test whether a particular design language is an Ada-based PDL: the similarity between the syntax and semantics of the PDL and those of Ada; and the support provided for particular phases in the software life cycle.

The first of these classification characteristics, relating to the syntax and semantics of PDL, is fairly obvious. An Ada-based PDL exhibits syntax and semantics that are consistent with Ada. With a few qualifications, an Ada-based PDL provides all the language features of Ada. The qualifications are that some Ada-based PDLs may discourage the use of language features that are regarded as too implementation oriented, for example, certain pragmas end representation specifications. These features may either not be supported, or a PDL may recommend deferring their use until the design text is translated into Ada code. Alternatively, the use of some features may be discouraged because the PDL is intended to develop designs which will be implemented in some programming language other than Ada. Even so, an Ada-based PDL always provides those features, such as packages, that are the essence of Ada.

Moreover, the members of this group are not PDLs where Ada is merely used in a PDL context; each member has one or both of the following attributes:

1) The PDL obligates the designer to follow rules and restrictions appropriate to the design activity.
2) The PDL is extended with features that assist in subsequent verification, documentation, or support of the design.

The second classification characteristic relates to the support provided by the PDL for particular phases in the software life cycle. All Ada-based PDLs share the same underlying conceptual view of the software life cycle and serve a similar purpose. They support the detailed design phase, and may additionally support preliminary design or other phases. In particuler, they should aid creation of the design, provide a mechanism for its expression, and serve as a template for writing code. Use of the PDL should also provide a common basis for communication among the members of a project team and support review of the design it captures. Finally, the PDL description should be capable of documenting the design.

Although there is some research into the use of graphical Ada-based PDLs, the progress to date does not justify inclusion of such graphical languages at this time. Consequently, the term Ada-based PDL will be used throughout this survey to denote a textual language.

In summary, a Ada-based PDL is a textual, procedural design language that follows Ada syntax and supports, at least, the detailed design phase in the software life cycle. These PDLs may be used to develop both Ada software and software written in other programming languages. They usually extend the Ada language with constructs that are desirable during the design activity. This classification excludes those PDLs that do not resemble Ada regardless of the programming language used to implement the design.

3.2 Characteristics Used to Describe Ada-Based PDLs

For the analysis portion of the survey, a list of PDL characteristics has been developed. These characteristics cover four broad areas of interest:

- Syntax/Semantic features,
- Automated support features,
- Methodology features, and
- Documentation features.

The characteristics in each area were used to determine the specific pieces of information that are recorded about each Ada-based PDL. These descriptions are presented in a series of tables in Section 4. The remainder of this section outlines the individual characteristics in each area. Underlines are used to highlight the initial reference to each piece of information.

3.2.1 Syntax/Semantic Characteristics

In order to compare Ada-based PDLs fairly, it is essential to recognize that the intentions of their developers may differ. For example, there is no universal agreement on whether or not these PDLs should remain completely faithful to the list of mandatory features in the Ada Language Reference Manual (LRM) [5]. With respect to the coverage of standard Ada features, there are two Ada coverage classes (these are sub-classes within the Ada-based PDL classification):

1) Full Ada: the PDL supports all Ada language features.
2) Subset Ada: certain features are not provided. For example, generics may be avoided if the target language is Pascal.

The coverage class of an Ada-based PDL provides an initial characteristic for this area of concern. It also leads to two other closely related characteristics. The first of these identifies the particular Ada language features not supported PDLs in the subset coverage class. The second records whether the developers of the Ada-based PDL recommend defferring the use of particular Ada language features difference between features not supported and

those whose use should be deferred is that deferral is not enforced. (If a subset Ada PDL is not supported by automated tools, then presumably reviews are used as the mechanism for enforcing restrictions to the syntax.)

Most Ada-based PDLs extend the Ada language with additional features specifically intended to support design activities. These extra features fall into three groups:

1) Mechanisms for indicating postponed decisions, such as TBD constructs and English narrative.
2) Mechanisms for expressing supplemental design information not expressible within the Ada language, often called annotations.
3) Additions to the PDL syntax, such as a new form of program unit or new types of commentary.

A number of characteristics are used to identify and describe the extensions provided within each group.

In the first group, postponed decisions, characteristics record the provision of mechanisms for postponing the implementation of types and postponing the implementation of operations.

Characteristics for the second group, supplemental information, record the ability of a Ada-based PDL to capture various types of supplemental design information. One type is configuration control information such as a version number, linking instructions, or identification of the system and Computer Program Configuration Item (CPCI) a module maps to. Reuse information can support the reuse of design modules by giving details that help to index the modules so that a library of reusable designs can be built. The component status, name of the original author, and a change log are also useful.

The PDLs may capture a general overview of the purpose and function of a module, and a description of the algorithm used. The usage of a module can be specified by preconditions and postconditions. Details about the error handling performed by the module are noted under exceptions raised or handled and error messages issued. Restrictions on a design module may include: timing constraints, sizing constraints, language constraints, hardware interfaces, and the software environment. Validation and verification of the design can be supported by specification of requirements met and testing requirements. The ability to measure certain aspects of the software, and the software developers, is facilitated by capturing metric data.

Information concerning how one module interacts with others includes task communication protocols, concurrent usage, interface specifications and information about external program unit references, external data references, enclosed data units, and enclosed program units.

Characteristics in the second group also record whether an Ada-based PDL allows expressing general notes and references to supporting literature end documents. Finally, some PDLs provide a keywords used or N/A feature that facilitates checking whether all required supplemental design information is present by indicating the types of supplemental information that are provided, or that are not applicable.

The third group of extra features that might be provided by an Ada-based PDL is chiefly concerned with additions to the PDL syntax. These may be: control constructs, simple statements, forms of commentary, or type definitions and operations for high level, abstract data types. The provision of format commands or other features only intended for interpretation by document and report generators are not included in this category.

In each case, the survey characteristics record whether any extra features are available. Additionally, the mechanisms used to provide the features are given. These mechanisms determine the compilability of an Ada-based PDL. Each extra feature will exhibit one of the following attributes:

1) Compatible with Ada: legal Ada syntex acceptable to the compiler.
2) Independent of Ada: never recognized by the compiler.
3) Incompatible with Ada: never acceptable to the compiler and must be removed before compilation.

In order for a design expressed in an Ada-based PDL to be compilable, without producing a host of extraneous error messages, all extra features must be either compatible or independent.

3.2.2 Automated Support Characteristics

The information of interest here is the type of automated support tools that are provided to facilitate the use of the Ada-based PDLs. At the current time, few developers have completed any tools. Those that are available are, of course, identified in this report. Many developers, however, provided lists of tools that are planned or under development. This information shows the direction that the development of automated tools is taking. Hence, planned tools are also included. A single characteristic is used to capture the tool status for each PDL. The host computers for the tools are also specified.

The remaining automated support characteristics were determined by compiling a list of desirable tools. A first version for this list was developed by merging references to all the tools mentioned in the received Ada-based PDL documentation, The resulting list was far too long, especially considering how few tools have actually been built. In the end, only the most important tools were selected for inclusion. The following remarks describe the nature of these selected tools.

A syntax directed editor assists in correct entry of the PDL and supports adhering to PDL standards and conventions, thereby reducing time spent on syntax verification, text editing, and resubmission of the design text. A pretty printer reformats Adabased PDL text according to a uniform convention to improve the legibility of a design expressed in the PDL.

A flexible document generator develops reports by analysis of the PDL and is guided by format templates or format commands. This type of document generator can be used to support the development of various types of documents, including MIL-STD-490 C5-Level specifications. However, there are some document generators that are specifically intended for producing C-5 Level specifications. In this report, these tools are called C5 specification generators.

A dictionary generator creates a listing of all symbols used in the design text and their definitions. Cross-reference tools list references to such things as task and procedure calls, data element usages, data type dependencies, and generic instances. A design level enforcer determines if the use of Ada in the design text is consistent with the design level. For example, use of unchecked conversion or certain pragmas may be flagged as inappropriate. Design annotation summary tools extract non-Ada annotations for analysis purposes. A unit interface report generator describes how packages and modules interact. These generators consider not only data flows and module invocations, but may also report on event synchronization.

A library manager tracks the relation of structured comments and their associated code structures. It generally notes the interdependence of modules and can recognize those modules affected by a modification to the design. Libraries of PDL designs may also assist new projects in the identifiication of design units from previous projects that could be reused.

3.2.3 Methodology Characteristics

PDLs categorized as Ada-based follow the Ada language quite closely, and it is reasonable to assume that all these PDLs will support the software engineering principles inherent to Ada. That includes modularity, abstraction, localization, hiding, uniformity, completeness, and confirm abililty. (Booch [6] provides an excellent discussion of the principles and goals of software engineering.) Further, these PDLs can all be used to develop a design following hierarchical decomposition and stepwise refinement approaches. Since these attributes are considered common to all Ada decomposition and stepwise refinement approaches. Since these attributes are considered common to all Ada-based PDLs no characteristics are used to record this information.

In every case, the Ada-based PDLs are suitable for use with a variety of software development methods. However, some PDLs were developed with specific methods in mind, and the developers of others recommend the use of particular methods. Where available, this information is captured by characteristics that give the name of the methods supported and, to help identify some of the lesser known methods, the name of the person or organization that developed the method.

The life cycle coverage recommended by the developers of the Ada-based PDLs is also of interest. Characteristics are used to record whether the PDLs are intended for use in the software requirements analysis, preliminary design, and detailed design phases. (The particular software life cycle representation used is taken from the proposed standard DoD-STD-SDS [7].)

The code and unit testing phase is not included in this list. This phase is omitted because these PDLs chiefly record the detailed design that is the input to the phase. They do not support the coding or testing activities in other ways.

Similarly, a support phase is not included. Instead, support is regarded as a repetition of development phases such as preliminary and detailed design. Consequently, any Ada-based PDL that can be used for certain development phases can be used for the corresponding phases during support activities.

3.2.4 Documentation Characteristics

Projects using an Ada-based PDL, or any PDL for that matter, require documentation of the PDL that is adequate to train people in its use and support their daily design activities by resolving questions quickly as they arise. Training and support have different documentation requirements. Ideally, there should be two documents. One is a tutorial guide or overview document that enables new users to quickly acquire enough familiarity to begin working with the PDL. The other is a reference manual that is organized in a fashion that allows the users to obtain accurate and complete information on any topic that arises in the use of the PDL. These needs are reflected by characteristics that record the availability of these types of documentation.

An important factor in the quality of the documentation is the amount of illustrative material that is provided. Consequently, one characteristic is used to record whether worked examples are provided to show features of the PDL in action. A second characteristic is used to indicate whether one or more larger case studies are providied to illustrate the use of the PDL in its application to a design problem of nontrivial size. The documentation for some Ada-based PDLs also includes examples showing how to the PDL design text can be translated into languages other than Ada.

The design of many Ada-based PDLs has been influenced by particular methodological issues. This is reflected by a characteristic that indicates the inclusion of a method of use discussion in the documentation. This type of discussion explains how to use the PDL to follow recognized software engineering practices during design activities. These discussions may also explain how project standards and design documentation requirements are met. Another characteristic is used to specify whether the documentation includes style guidelines that apply to the use of the PDL.

Finally, there are characteristics that record whether documentation is provided for any tools that suppport use of the Ada-based PDLs. One characteristic indicates the provision of tool operation notes that explain how each tool is used. A second indicates if commercial tool packages have an installation procedures guide.

4.0 DESCRIPTION OF Ada-BASED PDLS

This section provides a detailed and thorough description of each Ada-based PDL. Since there are sixteen of these PDLs, a relatively large amount of data is recorded. In order to make this information easy to assimilate, it is presented in two forms. Initially, a set of tables capture the details of Ada-based PDLs in terms of the characteristics described in the previous section.

4.1 Summary Descriptions of the Ada-Based PDLs

A set of eight tables are used to outline the characteristics of the different Ada-based PDLs. These tables are also intended to allow contrasting the features provided by each of the PDLs.

The first table, Table 4-1, provides an overview of the syntax and semantic features of the PDLs. It records how closely each PDL abides by Ada syntax and semantics by: 1) indicating the Ada coverage class; 2) identifying the particular Ada features not supported by PDLs in the subset class; and 3) indicating whether recommendations for deferring the use of certain Ada features are given. This table also indicates the types of extra features that are provided and, as a consequence of these extensions, whether each PDL is compilable.

Most of the Ada-based PDLs provide features to supplement those of Ada and details about these extensions are given in the next four tables. Table 4-2 describes the mechanisms used to express postponed decisions. The mechanisms used to provide supplemental design information are presented in Table 4-3, and the particular types of supplemental design information that can be expressed by each PDL are given in Table 4-4. The last table in this group, Table 4-5, outlines any other extensions provided by the PDLs.

Table 4-6 identifies the automated support tools that are provided for each Ada-based PDL. It distinguishes between tools that are currently available, and those that are only planned.

Table 4-7 presents the methodology features of the PDLs. This information comprises the particular methods that a PDL is intended to be used with and the recommended life cycle coverage.

The final table, Table 4-8, lists the different forms of documentation that are provided to support use of the PDLs and any support tools.

Table 4-1. Overview of Syntax/Semantic Features

PDL	Ada Coverage Class	Ada Features Not Supported	Deferred Use of Ada Features	Extra Features				Compilable
				Postponed Types	Postponed Operations	Supplemental Information	Other Extra Features	
ADA/SDP	(1)	n/a	No	Yes	Yes	Yes	Yes	No
ADAP	Full	None	Yes	Yes	Yes	No	Yes	No
BTO Ada PDL	Full	None	Yes	No	Yes	No	Yes	Yes
Byron	Full	None	No	Yes	No	No	Yes	Yes
GDADL	Subset	Use clause	No	Yes	Yes	No	No	Yes
Harris Ada PDL	Full	None	No	No	Yes	Yes	Yes	No
JPDL	Subset	Pragmas: Inline, Surpress	Yes	Yes	Yes	Yes	Yes	Yes
LTN Ada PDL	Full	None	Yes	Yes	Yes	Yes	Yes	No
PDL/Ada	Subset	?	Yes	Yes	Yes	Yes	No	No
RN ADAP	Full	None	Yes	Yes	Yes	No	Yes	No
SAIC Ada PDL	Full	None	Yes	Yes	Yes	Yes	Yes	Yes
Sanders Ada/PDL	Full	None	No	Yes	No	No	No	No
TI Ada PDL	Full	None	Yes	Yes	Yes	No	Yes	Yes
TRW Ada PDL	Full	None	No	Yes	No	Yes	Yes	Yes
WIS Ada PDL	Full	None	Yes	Yes	Yes	Yes	Yes	Yes
XAda	Full	None	No	No	No	No	Yes	Yes

Table 4-1: Overview of Syntax/Semantic Features

Key: ? Information Not Provided
n/a Not Applicable

Notes: (1) Follows philosophy of Ada, rather than Ada syntax

Table 4-2. Mechanisms for Postponed Decisions (1 of 2)

PDL	Postponed Operations				
	Provided	Predefined General TBD Statement	Set of Predefined TBD Statements	Other	Compilability Attribute
ADA/SDP	Yes	No	No	String of undefined identifiers	Incompatible
ADAP	Yes	No	No	// <text> //	Incompatible
BTO Ada PDL	No	n/a	n/a	n/a	n/a
Byron	No	n/a	n/a	n/a	n/a
GDADL	Yes	No	No	--| <text>	Independent
Harris Ada PDL	Yes	No	No	Structured English	Incompatible
JPDL	Yes	No	Yes	No	Compatible
LTN Ada PDL	No	n/a	n/a	n/a	n/a
PDL/Ada	Yes	No	Yes	No	Compatible
RN ADAP	Yes	No	No	Undefined Ada identifier	Incompatible
SAIC Ada PDL	Yes	No	Yes	No	Compatible
Sanders Ada/PDL	Yes	No	No	Structured English or "tbd"	Incompatible
TI Ada PDL	No	n/a	n/a	n/a	n/a
TRW Ada PDL	No	n/a	n/a	n/a	n/a
WIS Ada PDL	Yes	No	Yes	No	Compatible
XAda	No	n/a	n/a	n/a	n/a

Key: n/a Not Applicable

Table 4-2. Mechanisms for Postponed Decisions (2 of 2)

PDL	Postponed Type Specifications				
	Provided	Predefined General TBD Type	Set of Predefined TBD Types	Other	Compilability Attribute
ADA/SDP	Yes	No	No	Undefined Ada identifier	Incompatible
ADAP	Yes	No	No	// <text> //	Incompatible
BTO Ada PDL	No	n/a	n/a	n/a	n/a
Byron	Yes	No	Yes	No	Compatible
GDADL	Yes	Yes	No	No	Compatible
Harris Ada PDL	No	n/a	n/a	n/a	n/a
JPDL	Yes	Yes	No	No	Compatible
LTN Ada PDL	Yes	No	No	Undefined term "tbd"	Incompatible
PDL/Ada	Yes	No	Yes	No	Compatible
RN ADAP	Yes	No	No	Undefined Ada identifier	Incompatible
SAIC Ada PDL	Yes	No	Yes	No	Compatible
Sanders Ada/PDL	Yes	No	No	Structured English or "tbd"	Incompatible
TI Ada PDL	Yes	No	Yes	No	Compatible
TRW Ada PDL	Yes	n/a	n/a	deferred unit binding	Incompatible
WIS Ada PDL	Yes	No	Yes	No	Compatible
XAda	No	n/a	n/a	n/a	n/a

Key: n/a Not Applicable

Table 4-3. Mechanisms for Supplemental Information

PDL	Supplemental Information Provided	Prologue Templates Defined	Mechanism Used	Compilability Attribute
ADA/SDP	Yes	Yes	Statement of the form: <keyword> <text>	Incompatible
ADAP	Yes	Yes	Structured comment: --//<keyword>: <formatted details> or --//<keyword>: {--// <formatted details>}	Independent
BTO Ada PDL	Yes	Yes	Structured comment: --* {<number>} <keyword> <text>} *	Independent
Byron	Yes	Yes	Structured comment: --| <keyword>: <text> or --| <keyword> {--| <text>}	Independent
GDADL	No	n/a	n/a	n/a
Harris Ada PDL	Yes	Yes	Structured comment: -- <keyword phrase>: or -- <keywork phrase> {-- <formatted details>} -- {<Ada code>}	Independent
JPDL	Yes	No	Predefined Ada program units: packages, pragmas, subprograms, generic functions	Compatible
LTN Ada PDL	Yes	Yes	Structured comment: -- <keyword>: <text>;	Independent
PDL/Ada	No	n/a	n/a	n/a
RN ADAP	Yes	Yes	Structured comment: --//<keyword>: <formatted details> or --//<keyword>: {--// <formatted details>}	Independent
SAIC Ada PDL	Yes	Yes	Structured comment: --* <keyword>: <text> {--* <text>}	Independent
Sanders Ada/PDL	No	n/a	n/a	n/a
TI Ada PDL	Yes	Yes	Structured comment: --| <keyword>: <text> or --| <keyword> {-- <text>} {--| <Ada code>}	Independent
TRW Ada PDL	Yes	No	unstructured comment or narrative	Independent
WIS Ada PDL	Yes	Yes	Structured comment: --% <keyword> {--% <formatted details>} {--| <text>}	Independent
XAda	Yes	No	Structured comment: --| <keyword>: <formatted details> or --| <keyword>: {--| <formatted details>}	Independent

Key: n/a Not Applicable

Note: Curly braces are used as BNF metasymbols to indicate repetition.

Table 4-4. Types of Supplemental Information Provided (1 of 2)

	Configuration Control Information	Reuse Information	Component Status	Author	Change Log	Overview	Algorithm	Pre Post-Conditions	Exceptions Raised	Error Messages Used	Timing Constraints	Sizing Constraints	Language Constraints	Hardware Interfaces
ADA/SDP	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ADAP	-	-	-	Yes	Yes	Yes	-	Yes	Yes	-	Yes	Yes	Yes	Yes
BTO Ada PDL	Yes	-	-	Yes	Yes	Yes	-	-	-	-	-	-	-	-
Byron	-	-	-	-	-	Yes	Yes	-	Yes	Yes	Yes	Yes	-	-
GDADL	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Harris Ada PDL	Yes	-	-	-	Yes	-	-	-	Yes	-	Yes	Yes	Yes	-
JPDL	-	-	-	-	-	-	-	Yes	Yes	-	Yes	Yes	-	-
LTH Ada PDL	Yes	-	Yes	-	-	-	-	-	-	-	-	-	Yes	-
PDL/ADA	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RN ADAP	-	-	-	Yes	Yes	Yes	-	Yes	Yes	-	Yes	Yes	Yes	Yes
SAIC Ada PDL	Yes	-	-	Yes	Yes	Yes	-	-	-	-	-	-	-	-
Sanders Ada/PDL	-	-	-	-	-	-	-	-	-	-	-	-	-	-
TI Ada PDL	Yes	-	-	-	-	Yes	-	-	-	-	-	-	Yes	-
TRW Ada PDL	-	-	-	-	-	-	-	-	-	-	-	-	-	-
WIS Ada PDL	Yes	Yes	-	-	-	Yes	-	-	Yes	-	-	-	-	-
XAda	-	-	-	-	Yes	Yes	Yes	-	Yes	-	-	-	-	-

Table 4-4. Types of Supplemental Information Provided (2 of 2)

	Software Environment	Requirements Met	Testing Requirements	Metric Data	Task Communication Protocols	Concurrent Usage	Interface Specification	External Program Units References	External Data References	Encapsulated Data Units	Encapsulated Program Units	Notes	Technical References	Keywords Used or N A
ADA/SDP	-	-	-	-	-	-	Yes	-	-	-	-	-	-	-
ADAP	-	Yes	-	-	Yes	-	-	Yes	Yes	Yes	Yes	-	-	-
BTO Ada PDL	Yes	-	-	-	-	-	Yes	-	-	-	-	-	-	-
Byron	-	-	-	-	-	-	Yes	-	Yes	-	-	Yes	-	Yes
GDADL	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Harris Ada PDL	-	Yes	Yes	-	-	-	Yes	Yes	Yes	-	Yes	-	-	-
JPDL	Yes	-	-	-	Yes	Yes	-	-	-	-	-	-	-	-
LTH Ada PDL	-	-	Yes	-	-	-	-	-	-	-	-	-	Yes	-
PDL/ADA	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RN ADAP	-	Yes	-	-	Yes	-	-	Yes	Yes	Yes	Yes	-	-	-
SAIC Ada PDL	-	Yes	-	-	-	-	Yes	Yes	-	-	-	-	-	-
Sanders Ada/PDL	-	-	-	-	-	-	-	-	-	-	-	-	-	-
TI Ada PDL	-	Yes	-	-	-	-	-	Yes	Yes	-	Yes	Yes	Yes	-
TRW Ada PDL	-	-	-	-	-	Yes	Yes	-	-	Yes	-	-	-	-
WIS Ada PDL	-	Yes	-	Yes	-	-	Yes	Yes	Yes	-	-	Yes	-	-
XAda	-	Yes	-	-	-	-	Yes	Yes	Yes	-	-	Yes	-	Yes

Key: - Not Provided

Table 4-5. Other Extra Features Provided (1 of 2)

PDL	Name of Extra Feature	Purpose	Mechanism Used	Compilability Attribute
ADA/SDP	Text Unit	A block of text used as commentary to the design	Set of statements of the form: TEXT <name> {<text>} ENDTEXT	Incompatible
	Natural Language Text	To specify an operation in readable natural language	Unstructured statement of the form: <text>	Incompatible
	Defining Sentence	Describe the action and parameters of a program unit	Structured statement of the form: <text> {$<format parameters>$<text>}	Incompatible
Harris Ada PDL	Call & Engage Keywords	Used to differentiate between sequential and concurrent processing	Extensions to subprogram and task entry calls: call <subprogram call> engage <task entry call>	Incompatible
	I/O Keywords	Provide high level I/O operations	Structured statements corresponding to Ada standard packages: INPUT_OUTPUT, TEXT_IO, LOW_LEVEL_IO	Incompatible
JPDL	Abstract Description	Facility for abstractly specifying iteration over a sequence	Predefined generic package called: JPDL_Generic_Sequence_Package	Compatible
	Task Communication	Extension for task communication through abstract streams	Predefined package called: JPDL_Generic_Stream_Package	Compatible
	Periodic Loop	For use in delay statements to allow high level description of periodic loops	Predefined function: function Until_End_Of_Interval (Of_Length : Duration) return Duration;	Compatible
	External File Sequence	Allows abstract specification of iteration over each item in a file	Predefined generic function: generic type File_Type is limited private function JPDL_Internal_File_For (External_File_Name : String) return File_Type;	Compatible

Note: Curly braces are used as BNF metasymbols to indicate repetition.

Table 4-5. Other Extra Features Provided (2 of 2)

PDL	Name of Extra Feature	Purpose	Mechanism Used	Compilability Attribute
LTH Ada PDL	Call & Send Statements	Intended to increase the readability of procedure and task calls	Extension to Ada statement: call <procedure call> send <task entry>	Incompatible
	System & Subsystem Statements	Identifies key logical units	Modifies Ada statement by replacing "package" with "system" or "subsystem"	Incompatible
	Predefined Types & Operations	Provides various high level types and operations	Predefined library program units, grouped into 6 packages called: VDT_LIB MATH_LIB PROG_MGT_LIB PDL_TO_LIB ERROR_MGT_LIB STRING_LIB	Compatible
PDL/Ada	Logical, Data & Descriptive Commentary	Used to describe such things as predicate conditions and functions. Provided in addition to ordinary comments	Structured comments of the form: -- { <logical | data commentary>} -- { <* descriptive commentary *>}	Independent
	Anonymous Data Types & Processes	Provides high level types and operations for: stack, queue, sequence, set, list, character string	Set of generic packages that extend the Ada package STANDARD	Compatible
SAIC Ada PDL	Predefined Types & Operations	Provides high level types and operations for: message queue, stack; and high level operations to exchange items	Set of predefined packages called: MESSAGE_QUEUE, STACK, and EXCHANGE	Compatible
TRW Ada PDL	Module Construct	Used to describe a functional unit before determining whether it is a package, task, or subprogram	Structured construct similar to TRW Ada PDL package,task, and subprogram units	Incompatible
	Design Narrative	Used to describe any aspect of a design. Provided in addition to ordinary comments	Unstructured statements of the form: <text>	Incompatible
WIS Ada PDL	Group Annotation	Allows grouping related sets of declarations and associating a name and commentary with the group	Structured comments of the form: --% Group <title> --% Description {--| <text>} { <basic_declarative_items>} --% End Group <title>	Independent

Note: Curly braces are used as BNF metasymbols to indicate repetition.

Table 4-6. Automated Support Features

PDL	Tool Status	Host Computer	Syntax Directed Editor	Pretty Printer	Flexible Document Generator	CS Specification Generator	Dictionary Generator	Cross Reference Tool	Design Level Checker	Design Annotation Summary	Unit Interface Report	Library Manager
ADA/SDP	A	VAX, DG	-	Yes	-	-	Yes	Yes	-	-	Yes	-
ADAP	D	-	Yes	-		-	-	-	-	-	-	-
BTO Ada PDL	N	-	-	-	-	-	-	-	-	-	-	-
Byron	A	VAX	-	-	Yes	Yes	Yes	Yes	-	(1)	Yes	Yes
CDADL	A	VAX, HP1000, Tektronix 8560	-	Yes	-	-	(3)	Yes	-	(2)	-	-
Harris Ada PDL	A	VAX	-	-	-	-	-	Yes	Yes	-	-	-
JPDL	N	-	-	-	-	-	-	-	-	-	-	-
LTN PDL	N	-	-	-	-	-	-	-	-	-	-	-
PDL/Ada	N	-	-	-	-	-	-	-	-	-	-	-
RN ADAP	N	-	-	-	-	-	-	-	-	-	-	-
SAIC Ada PDL	D	-	-	Yes	-	-	-	Yes	-	(1)	-	-
Sanders Ada/PDL	A	VAX	-	Yes	-	-	-	Yes	-	-	-	-
TI Ada PDL	D	-	-	Yes	-	Yes	Yes	Yes	Yes	-	YES	-
TRW Ada PDL	A	VAX UNIX	-	Yes	D	D	Yes	Yes	-	-	-	Yes
WIS Ada PDL	D	-	Yes	Yes	Yes	-	Yes	-	Yes	Yes	-	Yes
XAda	D	-	-	-	Yes	-	-	-	-	-	-	-

Key: A Available
D Under Development or Planned
N Not Mentioned
- Not Provided

Notes: (1) The flexible document generator could probably produce the desired annotation summary.
(2) The CDADL tool can remove "nondesign" Ada lines leaving only design annotation plus skeletal Ada statements.
(3) Mentioned in the preliminary documentation but no [illegible]

Table 4-7. Methodology Features

PDL	Specific Methods Supported		Life Cycle Coverage		
	Name of Method	Name of Developer	Software Requirements Analysis	Preliminary Design	Detailed Design
ADA/SDP	None	n/a	No	Yes	Yes
ADAP	Structured Design	Edward Yourdon	No	Yes	Yes
BTO Ada PDL	None	n/a	No	No	Yes
Byron	Modular Program Construction	Barbara Liskov	No	Yes	Yes
CDADL	Disciplined Software Design Approach	General Dynamics	No	Yes	Yes
Harris Ada PDL	Integrated Software Methodology	Harris Corporation	Yes	Yes	Yes
JPDL	Jackson System Development Structured Requirements Analysis Structured Design Higher Order Software	Michael Jackson Ken Orr Edward Yourdon HOS, Inc.	Yes	Yes	Yes
LTN Ada PDL	Software Data Flow Diagrams Software Functional Hierarchy Diagrams	MIL-STD MIL-STD	No	Yes	Yes
PDL/Ada	IBM FSD Design Methodology	IBM-FSD	No	Yes	Yes
RN ADAP	Structured Design	Edward Yourdon	No	Yes	Yes
SAIC Ada PDL	None	n/a	No	No	Yes
Sanders Ada/PDL	Structured Analysis Structured Design	Tom DeMarco Edward Yourdon	No	Yes	Yes
TI Ada PDL	Software Work Breakdown Figure-8 Design Process Three-Leaved Rose Design Process	MIL-STD ? ?	Yes	Yes	Yes
TRW Ada PDL	None	n/a	No	Yes	Yes
WIS Ada PDL	None	n/a	(1)	Yes	Yes
ZAda	None	n/a	No	Yes	Yes

Key: ? Information Not Provided
n/a Not Applicable

Notes: (1) Also intended for use in system design activities

Table 4-8. Documentation Features

PDL	Tutorial or Overview Material	Reference Material	Method of Use Discussion	Worked Examples	Case Study	Translation Examples to Non-Ada Languages	Tool Operation Notes	Style Guidelines	Installation Procedures Notes
ADA/SDP	-	Yes	Yes	Yes	Yes	-	-	-	-
ADAP	Yes	Yes	Yes	Yes	-	-	-	Yes	-
BTO Ada PDL	-	Yes	-	Yes	-	-	-	-	-
Byron	-	Yes	Yes	Yes	-	-	Yes	Yes	Yes
GDADL	-	Yes	Yes	-	-	-	Yes	-	-
Harris Ada PDL	Yes	Yes	Yes	Yes	Yes	Assembler	Yes	Yes	-
JPDL	-	Yes	Yes	Yes	-	Fortran, C	-	-	-
LTN Ada PDL	-	Yes	Yes	Yes	-	-	-	Yes	-
PDL/Ada	-	Yes	Yes	Yes	Yes	-	-	-	-
RN ADAP	Yes	Yes	Yes	Yes	Yes	Pascal	-	Yes	-
SAIC Ada PDL	Yes	-	Yes	Yes	Yes	-	-	Yes	-
Sanders Ada/PDL	-	Yes	-	-	-	-	-	-	-
TI Ada PDL	-	Yes	Yes	-	-	-	-	-	-
TRW Ada PDL	Yes	Yes	Yes	Yes	-	Fortran, J73	Yes	Yes	Yes
WIS Ada PDL	Yes	Yes	Yes	-	-	-	-	-	-
XAda	-	-	Yes	-	-	-	-	-	-

4.2 Detailed Descriptions of Ada-Based PDLS

The tables in the previous subsection only give an overview of the Ada-based PDLs. For example, they list the type of automated support tools that are available or planned, but do not differentiate between the cross-reference tools provided for one PDL and another. Similarly, the tables identify the different types of documentation that are provided, but do not capture the extent of each piece of documentation.

This subsection provides this additional information. For each Ada-based PDL, it addresses the four areas of interest discussed in Section 3.2.

4.2.1 ADA/SPD (Mayda Software Engineering, IL Ltd.)

ADA/SDP is based on an earlier non-Ada PDL called SDP and, as a result of this, is somewhat different than the other Ada-based PDLs. It uses a simplified Ada syntax to facilitate a more natural and readable design description and allows the software developer to choose the extent of detail appropriate for the task in hand. This is intended to reflect the human factors considerations of the design process. Unlike many Ada-based PDLs tools, the automated support provided for ADA/SDP can accept and process natural language.

4.2.1.1 Syntax/Semantic Features of ADA/SDP

ADA/SDP differs from Ada in four ways. First, the end of line terminator is significant in delimiting statements and use of the semicolon is optional. When statements must continue over multiple lines, a continuation character (\) is used. Second, in the source text file, subprogram parameters are surrounded by a sentinel character ($), but in the pretty printed output these parameters are underlined instead. Type and mode information is treated in a similar way. Third, keywords are recognized in a given line only if the line begins with a keyword. Fourth, identifiers may consist of more than one word.

Natural language statements are allowed wherever a simple statement or expression is allowed.

ADA/SDP divides a design into design units which correspond to Ada program units. Each of these design units is introduced with one of the appropriate Ada keywords: PROCEDURE, FUNCTION, PACKAGE, or TASK. ADA/SDP also provides a new keyword TEXT which introduces a unit consisting only of commentary. The Ada keywords are followed by a construction which is the unique semantic innovation of ADA/SDP. Where Ada follows a program unit keyword with the name of the unit and a list of identifiers, ADA/SDP uses a "defining sentence" which is a specially annotated natural language sentence describing the action of the unit. The sentence consists of action words and object names. In order to make the sentence easily machine processible, names that refer to the parameters of the unit are delimited with the sentinel character. Thus, an ADA/SDP design unit is introduced by a statement that describes its purpose and simultaneously defines its formal interface with other units.

Type definition facilities in ADA/SDP are nearly identical to those of Ada. However, the PDL accepts incomplete type definitions supplemented by narrative description. Also the visibility rules differ from Ada in that the WITH statement implies both the Ada WITH and USE statements.

ADA/SDP cannot be directly compiled.

4.2.1.2 Automated Support Features of ADA/SDP

ADA/SDP is currently supplied with a single tool that performs text processing, syntax checking, and cross-referencing functions. At present, there is no separate documentation for the tool and the remarks here are largely based on inspection of the large case study furnished by the developer. The tool will be available initially for Unix systems, VAX/VMS, and on Data General machines.

The standard document format produced by the tool features a numbered table of contents, a listing on a separate page for each element of the design, and separate cross-reference tables for modules, objects, types, and packages.

The page format for the ADA/SDP listing surrounds the PDL with a box that facilitates associating line numbers with the PDL lines. Outside the box, the listing shows page references for identifiers that occur on the corresponding line inside the box. These cross-reference annotations are supplemented by the cross-reference tables where references show both the page and line number.

The text processor prints keywords in boldface and repeats the identification of the PDL text block at the top of the page for ease of reference. It also performs text rearrangement for items embedded between sentinel characters. For example, the phrase:

```
function user confirms $user action: request type$
     return boolean is separate
```

is rendered in the printed document as:

function user confirms user action **return** boolean
    user action : request type
    amount of money : money
return boolean **is separate**

A tool that would render such phases into compiler compatible Ada text is under development and should be included in the second version of the PDL tool.

Each function and procedure in ADA/SDP is defined by a sentence that describes the design unit interface. When the documentation tool assembles the module dictionary and cross-references, these sentences are repeated and serve the purpose of a unit interface specification document.

4.2.1.3 Methodology Features of ADA/SDP

The ADA/SDP PDL was developed for use in preliminary and detailed design. It is not intended for use with any particular software method.

4.2.1.4 Documentation Features of ADA/SDP

The PDL is supplied with a single reference manual that is marked as a draft version. The manual describes both the syntax that derives closely from Ada and the syntax that pertains to the earlier SDP design language. Various points are illustrated with examples that are especially useful in illustrating the narrative flavor of the Ada-based PDL. However, the draft version does not contain a complete tutorial on the use of ADA/SDP during design activities. In particular, it would be helpful to have a discussion on how to convert the narrative PDL to code acceptable to the Ada compiler.

4.2.2 ADAP (SofTech, Inc.)

This PDL is one of the three Ada-based PDLs developed by SofTech that are discussed in this report, the others being RN ADAP and JPDL. ADAP was developed before RN ADAP and the high degree of similarity between these two PDLs suggests that ADAP provided the basis for RN ADAP. There are no significant similarities between these two PDLs and JPDL.

RN ADAP will be discussed in Section 4.2.11. However, in comparison to ADAP, it should be noted that RN ADAP provides 1) a different approach to deferred specification of identifiers; 2) more specific style recommendations; and 3) guidelines for conversion of ADAP to Pascal.

4.2.2.1 Syntax/Semantic Features of ADAP

ADAP allows the use of complete and correct Ada syntax. The PDL design text consists of Ada interspersed with English prose that provides the design narrative. In this PDL, the English prose is delimited by a special symbol (//). The English prose can be substituted for statements, expressions, or type definitions.

ADAP is not directly compilable without textual preprocessing to replace the English prose. However, if legal subtitutions are made for the delimited prose statements, then the text should be compilable.

4.2.2.2 Automated Support Features of ADAP

No automated support tools are provided for ADAP.

4.2.2.3 Methodology Features of ADAP

ADAP is intended to address the software design principles of modularity, localization, abstraction, information hiding, reliability, and completeness. In particular, it supports the Constantine/Yourdon Structured Design method, accompanied by top-down development and stepwise refinement.

4.2.2.4 Documentation Features of ADAP

The ADAP manual provides an excellent example of quality instructional material that can help new users of PDL. The manual introduces each feature with a careful explanation and a clear example. The manual appears to be intended to teach good design practices in Ada, and the PDL is explained in the framework of this objective.

Style guidelines are provided in the form of a discussion on standard structured header commentaries. Together with the many examples, this guide defines a consistent usage style for the PDL.

4.2.3 BTO Ada PDL (Bell Technical Operations)

The Ada-based PDL described here was developed for use in the design of the Vehicle Integrated Defense System (VIDS). It was jointly developed by Bell Technical Operations and Dalmo-Victor. Both these companies have established guidelines for developing PDLs. In the case of BTO Ada PDL, the Ada-based aspect of the PDL was only one of several design requirements.

4.2.3.1 Syntax/Semantic Features of BTO Ada PDL

BTO Ada PDL is another representative of the class of Ada-based PDLs that use complete Ada for design purposes. In this case, the normal Ada comment statement is used to delimit design narrative. Since there is no distinction between design narrative that will eventually be replaced by code and commentary that will remain to explain the design, readers must distinguish between the two based on context.

The documentation of BTO Ada PDL does not explicitly comment on whether the design text is compilable, but most is. For example, Example 4.4 from the manual is compilable:

```
loop
     -- get a reply to operator query
     -- when the reply is
     -- a 1, 2, 3 or 4 exit
     -- the loop
     -- else request a retry
end loop;
```

Here comments are used where zero or more statements are required in Ada. To the Ada compiler, the above loop has no statements. A limitation of this technique is that where the compiler requires at least one Ada statement, the comments must be interspersed with Ada statements in order to avoid compiler errors.

Example 4.5 shows one of the cases that are not compilable:

```
loop
    -- read next   cord;
    exit when --endfile marker;
    -- process input data;
end loop;
```

However, this example could be rewritten in a compilable form as:

```
loop
    -- read next record
    exit; -- when endfile marker
    -- process input data
end loop;
```

4.2.3.2 Automated Support Features of BTO Ada PDL

No automated tools are provided.

4.2.3.3 Methodology Features of BTO Ada PDL

This PDL is intended as a design aid for use in describing a software implementation. The developers of the PDL recognize the dangers of using a PDL that allows highly detailed language dependent representations to be produced. They caution that the PDL should be used to formalize a definition of the solution to a problem, and not used to code a solution.

There are no indications that this PDL was designed for use with any specific design methods.

4.2.3.4 Documentation Features of BTO Ada PDL

The reference manual is devoted to explaining what features of Ada a designer must know in order to perform the PDL phase of a design. The manual will be of most use to the reader who is already familiar with the use of PDL in design and who requires guidance on how Ada features should be used.

4.2.4 Byron (Intermetrics, Inc.)

Byron was the first Ada-based PDL to be supported by a powerful set of automated tools and is now well established. It is one of the few PDLs that was developed as a marketable product, and is the only PDL identified on this survey that is in use by several organizations.

Byron provides a uniform means of expression for all phases of a project. It is intended not only to ease communication, but also to facilitate the provision of automated support for each software development activity. One of its objectives is to help ensure that documentation is consistent with the executable code. This is achieved by keeping all information about a program unit in a single source file.

4.2.4.1 Syntax/Semantic Features of Byron

Byron uses the full Ada languaqe for design purposes. In addition, it provides three design constructs called Byron text, Byron directives, and Byron markers. Each is preceded by the Byron prefix symbol (--|). This symbol causes them to be ignored by the Ada compiler but allows them to be easily distinguishable by the human reader.

Byron text consists of the prefix symbol followed by English narrative. It is used immediately after the declaration of an identifier and causes the descriptive text to be associated with that identifier in the program library.

Byron directives are used to annotate the design with supplemental information. They consist of the prefix symbol followed by one of eleven Byron keywords and appropriate text. The PDL reference material indicates those directives that are required and those that are optional. It also provides guidelines concerning the order in which directives should be given.

The final additional construct is the Byron marker. The previous two types of constructs are used in the declarative part of the PDL. The Byron marker, however, is used in the statement part. The effect of this construct is to mark lines of code or comments as Byron text. This allows the report generator to include these lines in the reports it produces. The Byron marker consists of the prefix symbol. It can be used at the end of a line to mark the entire line, or earlier in the line to mark the following code or comment.

All of the Ada keywords and user defined Ada identifiers are significant in the PDL text. This text establishes a design structure and a dictionary of known program units, objects, and types. When Byron design annotations are encountered they are assumed to apply to an object or control structure as determined by their location with respect to the Ada text. In this fashion, Byron adds new attributes to the entities in Ada. These attributes are then extracted and summarized in the reports derived from the design text.

In summary there are three semantically different ways in which a Byron design may be read:

- The Ada compiler's reading,
- A linear reading of the fully annotated design text to understand the Ada code in the context of the extra Byron design information, and
- A reference reading in which entities are listed separately from their source text location, but together with their Ada and Byron defined attributes.

Byron has a predefined package called TBD which contains Ada type definitions that can be used to defer the full specification of data types.

4.2.4.2 Automated Support Features of Byron

The Byron PDL is provided with an extensive set of tools designed to operate on a VAX machine. Although the set does not cover all of the categories in Table 4-6, the coverage is impressive.

The library manager is a useful tool that has an especially important application because Byron design annotations are recognized in the framework established by the legal Ada statements. This allows Ada packages to inherit design information through their association with other packages. The library manager ensures that changes to the design information of one package will be reflected in all the packages related to the changed one. In essence, it offers the same facilities for incremental and independent extraction of design information that the Ada compiler offers for incremental and independent compilation of modules.

The flexibility of the Byron document generators support the use of Byron to develop government required documentation such as the C-Level Specifications defined in DoD-STD-490.

4.2.4.3 Methodology Features of Byron

Byron can be used with a variety of design techniques. The developers recommend that users become familiar with several techniques so that they can select those most appropriate for each project. The factors that should be considered in this selection include: availability of resources, organization of the project team, and the structure of problem to be addressed.

Particular attention is given to the use of Byron with the Modular Program Construction approach developed by Barbara Liskov. This design method is an extension of the stepwise refinement technique. It views a program as a high-level procedural abstraction that can be constructed from lower level procedural, data, and control abstractions.

4.2.4.4 Documentation Features of Byron

Documentation for Byron is extensive and covers not only the PDL but also the large number of tools that are furnished with the language. The Byron developers reflect their interest with the methodology work of Barbara Liskov by including a reprint of one of her papers with the documentation. The methodology is also summarized adequately in the Byron manual itself. Rounding out the documentation set is an installation guide that can help new users install the Byron tools on their machine.

What is missing in the current documentation is an introduction for the relatively inexperienced user. If one imagines the situation of learning the Liskov method, the Ada language, the Byron PDL, and half a dozen tools at the same time, the need for a gradual, moderately paced introduction becomes apparent.

4.2.5 GDADL (General Dynamics)

The General Dynamics PDL consists of a combination of Ada code and pseudo code design statements. Specifically, it uses Ada to describe the program structure and data structure, and design statements to describe the control flow of the program.

4.2.5.1 Syntax/Semantic Features of GDADL

A design in GDADL contains three types of text: full legal Ada, comments, and design statements. The types are never mixed in any fashion that would prevent compilation of the code. Thus, a GDADL design should always be compilable except for the following case.

The visibility rules of GDADL do not require a WITH statement. Instead, all packages are simultaneously accessible. However, in order to be compilable, WITH clauses must be given. Similarly, the USE statement has no effect in GDADL. Consequently, fully disambiguated names must be used for the benefit of the GDADL processor.

The declarations of program units, objects, types, task entries, and generic parameters are written in full Ada. These declarations are used both by the compiler and the PDL processor.

Design information concerning the bodies of the program units is written in PDL design statements. These statements are Ada comments that begin with a special symbol (--|). They may use Ada keywords to express control structure information, but the intent is to use English narrative for high level expression of the algorithm. The design statements are searched by the PDL processor for references to objects, types, and program units defined in the Ada declarations.

When the design is implemented, Ada code is added so that code and design statement alternate in small groups of lines and the relationship of design and code can be easily determined.

GDADL displays one lexical restriction that is not present in Ada. When a keyword begins a standard construct, for example the IF in IF .. THEN .. ENDIF, the keyword must be the first nonblank symbol on a line. On the other hand, GDADL will overlook the omission of a semicolon in a design statement.

4.2.5.2 Automated Support Features of GDADL

GDADL is supported by a single tool that combines the functions of a pretty printer and a cross-reference generator. The tool is presently hosted on VAX and HP1000 computers, in addition to a Textronix 8560 workstation.

The pretty printing function has the usual features and one uncommon one. The pretty printed output is annotated with "flow lines." These lines mark references to other procedures and package declarations by means of a line that points to the margin of the text where the page and line reference numbers are recorded. Although the same function is served by the cross-reference tables, the flow line annotation is a useful feature.

Many software developers feel that the RENAMES and USE Ada language features are best not used in Ada code, much less PDL text. The GDADL tool reflects this concern by not recognizing these constructs. In the case of RENAMES this means that contraction of a long unambiguous name to a short

locally valid name is not acceptable. The decision not to support the USE construct means that fully disambiguated names must be used and infix operators cannot be overloaded.

The GDADL tool also provides a mechanism to extract design documentation and exclude non-design Ada statements from the printed report. This feature is helpful for maintaining accurate, up to date documentation, particularly when the design is iterated in the face of test or analysis results obtained late in the software life cycle.

4.2.5.3 Methdodlogy Features of GDADL

GDADL is intended to improve productivity and quality during the software development process. However, its use is expected to yield the most benefits during maintenance and enhancement activities.

GDADL was developed to support the Disciplined Software Design Approach (DSDA) used at the Pomona Division of General Dynamics. This methodology uses a real-time version of Tom DeMarco's classical Structured Analysis to define the software requirements. These requirements are expressed and verified using the PSL/PSA tool developed by the ISDOS project. Then a stepwise refinement approach is used to define the design.

GDADL is intended for use in three stages of design refinement called: architectural design, executive design, and detailed design.

4.2.5.4 Documentation Features of GDADL

Documentation for GDADL consists of a users manual, a reference manual, and a discussion of the DSDA methodology. The manuals are in an early draft stage, but their tables of contents indicate that the planned documentation will be more than adequate. However, the available material does not specify whether case studies and a graduated series of introductory tutorial material will be included.

4.2.6 Harris Ada PDL (Harris Corp.)

Harris Ada PDL was one of the first Ada-based PDLs to be developed. It evolved from an earlier non-Ada PDL and the experience gained with the previous design language was used to guide the development of the Ada-based PDL. The objective of the new PDL is to provide a form of communication that is easy to use, easy to understand, and that simplifies definition, design, and review.

The PDL is only one part of the Harris Integrated Software Methodology (ISOMET). It is termed a Process Description Language, not a Program Design Language, and intended to support both requirements definition and design activities.

4.2.6.1 Syntax/Semantic Features of Harris Ada PDL

Harris Ada PDL uses standard Ada syntax extended with predefined templates for embedding supplemenal design information in Ada specification parts.

The supplemental design information is written in the form of Ada comments and each item is introduced by a keyword. Some of these annotations just flag the following Ada text and others provide additional design details. Where details are required, Harris Ada PDL defines the format that should be used for each annotation. The specification templates also indicate where additional descriptive commentary should be given. This commentary is written in the form of Ada comments, but is distinguished by preceding the text with the ":" character.

There is one major semantic difference between Harris Ada PDL and Ada. In order to identify critical access and control logic, Harris Ada PDL introduces some additional PDL keywords. Task and procedure entries are distinguished by the non-Ada keywords ENGAGE and CALL, respectively; and OPEN, CLOSE, CREATE, DELETE, READ, WRITE, GET, PUT, SEND-CONTROL, and RECEIVE-CONTROL are used in input/output statements. Except for these extra keywords, Harris Ada PDL is compilable.

The developers recommend using the PDL in conjunction with Ada in a split page approach. This means that the PDL is moved to the right as a block of commentary and Ada is then written to the left of the PDL. Following this approach, compilation is not attempted until all of the non-Ada keywords have been moved into Ada comments.

4.2.6.2 Automated Support Features of Harris Ada PDL

Harris Ada PDL is supported by several tools which are available only for use on Harris projects. However, the documentation for these tools reveals an interesting feature that deserves mention.

The ISOMET methodology recognizes that system designs undergo progressive decomposition or partitioning during design and implementation activities. When a component belonging to one partition of the system is decomposed further, the internal components of the new design detail must be checked for consistency. The Harris cross-reference tool has the capability to scan its data base and perform a "vertical validation" to ensure that the design is consistent after partitioning.

Documentation was also provided on some Harris automated support tools that are not Ada specific but nevertheless provide useful assistance in drawing and plotting graphical design information.

4.2.6.3 Methodology Features of Harris Ada PDL

Use of the Harris PDL is just one of the software development practices defined by Harris's Integrated Software Methodology (ISOMET). This methodology is an integrated set of policies, guidelines, and techniques that support the entire software life cycle. The principles on which ISOMET is

based include: documentation from inception through maintenance; software analysis techniques based on top-down partitioning and progressive elaboration; interface description tools; data description methods; and validation and verification of the software throughout the life cycle.

The PDL is the tool used for communication of software requirements, and preliminary and detailed software designs. It is also used as in-line commentary during implementation. Hence it is an integral part of ISOMET.

4.2.6.4 Documentation Features of Harris Ada PDL

The fact that the Harris PDL has been in use for several years is most evident in the extent of the PDL documentation. In addition to complete reference documentation, training material is also available.

The main PDL manual contains both reference material and a users guide. The original version, developed in 1982, was updated in 1984 to bring the PDL into full compliance with MIL-STD-L8L5A. The manual includes recommendations for style, design methodology, and translation of the PDL text into ADA. Appendices to the manual provide templates that can be used as guidelines for achieving a uniform PDL style.

The Harris documentation uses a story board format in which each page explains a point from the outline at two levels of detail. Thus the reader is encouraged to move quickly through familiar material and focus only on what is most relevant to each reading of the text.

4.2.7 JPDL (U.S. Air Force Systems Command)

JPDL is designed for use in the reimplementation of the JINTACCS Automated Message Preparation System (JAMPS). Hence the name JAMPS Program Design Language, or JPDL. The JAMPS software will be implemented in Ada. However, the JAMPS data base will be used by systems implemented in both C and FORTRAN. This multilingual environment has influenced the design of JPDL only to the extent that conversions from JPDL constructs to both FORTRAN and C are provided. The PDL is primarily intended to develop Ada software.

JPDL was developed by SofTech for the U.S. Air Force Systems Command at Hanscom AFB.

4.2.7.1 Syntax/Semantic Features of JPDL

JPDL is based on Ada but uses several major semantic extensions to provide mechanisms for postponing design decisions, specifying abstract operations, and expressing supplemental design information.

Deferred specification of computations is supported by a package called JPDL-Value-Description-Package. The functions contained in this package take the notation (+ " descriptive text") and return a legal value. This effect is accomplished by overloading the "+" operator to yield each predefined Ada type

when applied as a prefix operator to a string. No calculation is performed, instead this mechanism allows calculations to be described in English text.

JPDL allows the use of incomplete type definitions by means of a type To-Be-Determined that is defined in a JPDL package. According to the documentation, this type is defined as private so that no operations on objects of type To-Be-Determined can be expressed in the PDL.

JPDL also provides a abstract construct for looping on a sequence. This was motivated by a similar facility in the language CLU. A predefined package containing sequence functions is used to express this idea in Ada. For example, to code the notion present in the following lines:

```
for all E in S loop                    -- where E is an element in
     Process-Element (E);              -- a sequence of elements S
end loop;
```

a designer can write the following in JPDL:

```
while not End-of-Sequence(S) loop
     Get-Next-Element (S,E);
     Process-element (E);
end loop;
```

Similar provisions are made to describe periodic loops and streams of values to, or from, external files.

Another predefined JPDL procedure provides for communication streams between processes. This extension is intended to allow an abstract view of intertask communication during design. Currently, only single tasks can communicate through these streams. However, the documentation states that future work will address the issue of allowing more flexible networks of communicating tasks.

All the JPDL extensions are compatible with Ada and, therefore, the design text is compilable. However, they need to be removed or implemented in Ada during the coding phase of the software development.

4.2.7.2 Automated support Features of JPDL

No tools are provided with JPDL.

4.2.7.3 Methodology Features of JPDL

JPDL was developed to support the preparation of Design Specifications in accordance with DOD-STD-1679, and its proposed successor DOD-STD-SDS. The JPDL text can be used in the requirements section of a design document to provide an additional representation of information traditionally expressed in schematic and block diagrams.

JPDL is intended to address the issues of programming-in-the-small and treats data, data flow, and program control simultaneously. It extends the

support provided by Ada for programming-in-the-large with additional notations for concurrent programming.

Additionally, the developers of JPDL say it can be used in B-Level Specifications, though this is not its primary purpose. Specification languages and design languages serve very different purposes and are usually structurally different. Consequently, the designers caution users that JPDL is not a suitable vehicle for schematic or system level block diagrams, but should be used for functional and nonfunctional descriptions of the software to be designed, including data and interface definitions. Furthermore, at this level, JPDL must be used in conjunction with a method that guides the specification of requirements.

It is intended to support a variety of software methods, including Jackson System Development (JSD), Ken Orr's Structured Requirements Definition, Constantine/Yourdon's Structured Design, and the Higher Order Software (HOS) method.

4.2.7.4 Documentation Features of JPDL

The documentation consists of a single reference manual. In order to support the use of the PDL in a multi-lingual environment, the manual provides conversion techniques showing how to transform any arbitrary PDL construction into FORTRAN and C. Additionally, the manual provides conversion of the new JPDL loop abstractions and interface abstractions into Ada, as well as FORTRAN and C.

In many cases, the JPDL document eschews fixed standards. Instead, it provides discussions on a variety of issues such as naming conventions and documentation. This is intended to make it possible for individual projects to develop their own standards that meet their special needs.

As a result of the amount of information it contains, the JPDL manual is very large and might be intimidating to new users of JPDL. However, in those cases where only one of the target languages will be used, there is an easy way to resolve this problem. Projects using JPDL need to review the document to establish a set of project standards, and this provides an excellent opportunity for developing a project specific manual. This manual would recommended choices for standards and elaborate only those points applicable to the target language used on the project.

4.2.8 LTH Ada PDL (LTH Systems, Inc.)

The LTH Ada PDL was developed under contract to the Center for Tactical Computer Systems (CENTACS), U.S. Army Communication and Electronics Command (CECOM). Originally intended as an interim guide for the use of Ada-based PDLs, the report actually provides a definition of an Ada-based PDL.

The above remarks are intended to provide an historical perspective for the PDL discussed here. They are not meant to imply that this PDL has been, or will be, adopted in any CECOM policy.

4.2.8.1 Syntax/Semantic Features of LTH Ada PDL

The LTH Ada PDL is defined in the following manner in Section 3.4 of the reference manual:

> "The Ada Based PDL is a top-down textual representation of the system's software 'B5' and 'C5' specifications utilizing the Ada Based compositional elements which are expressed in the Ada Based (ABDDD or PPDLD and ABPDD or DPDLD) documents' format based upon the project's required PDL type."

(Note: ABDDD = Ada Based Development Design Document
PPDLD = Preliminary Program Design Language Document
ABPDD = Ada Based Product Design Document
DPDLD = Detailed Program Design Language Document)

In effect, the definition says that the PDL is a function of the design phase and the project type. LTH distinguishes two phases, preliminary design and detailed design; and three project types, pure Ada implementations, mixed Ada with other language implementations, and pure non-Ada implementations. In total, the LTH Ada PDL actually consists of six recommendations from which one is selected depending upon circumstances. However, the discussion here of syntax/semantic features is primarily concerned with the "Ada Based compositional elements" which are common to all six variations of LTH Ada PDL.

The LTH Ada Based compositional elements consist of the following six statement types:

1) Ada: statements conforming to MIL-STD-1815A. In addition, style recommendations are given and the use of pragmas is discouraged;
2) Ada Based PDL: Ada statements extended by non Ada keywords: CALL, SEND, SYSTEM, and SUBSYSTEM;
3) Machine Code: assembler level instructions. In the interest of portability, the use of descriptive operation code names is encouraged;
4) Other Language: instructions written in some language other than Ada;
5) User Defined: statements following a project specific formalism.
6) English: narrative statements introduced by a special symbol (!!) and ending in a semicolon.

Only the first of these compositional elements is standard Ada. Moreover, LTH Ada PDL discourages the option to mark PDL design statements as Ada comments. Therefore, the PDL is not compilable.

4.2.8.2 Automated Support Features of LTH Ada PDL

No support tools are provided. The reference manual refers in several places to a discussion of "Applicable Automated Tools". However, the relevant section was omitted from the final report.

4.2.8.3 Methodology Features of LTH Ada PDL

The LTH Ada PDL is intended for use in preliminary design and detailed design. It recommends that PDL text is accompanied by Software Data Flow Diagrams (SDFD) and Software Functional Hierarchical Diagrams (SFHD). The SFHDs provide graphical illustration of the hierarchical structure of the logical/program functional units. The SDFDs provide a graphical representation of the data flows between the logical/program units. Although these diagramming techniques are sometimes associated with particular software methods, the recommendation of their use is not meant to endorse a particular method.

LTH PDL text is designed to be consistent with various government standards. These standards include: MIL-STD-483, MILSTD-490, DoD-STD-1679A, ANSI/MIL-STD-1815A, and the proposed standard DoD-STD-SDS. Additionally, it meets DARCOM 70-16R and CECOM Policy 8-81 requirements.

4.2.8.4 Documentation Features of LTH Ada PDL

The LTH PDL is documented by a single reference manual. Due to the developers concern with military software procurement procedures, the manual discusses the various types of documentation standards and format guidelines that may be required in a given project in unusual depth.

The text is also strong in the area of explaining the purpose of PDL and its intended use in the software development process. A final noteworthy feature is a section describing six predefined library units that the software developer can use in the development of a design.

4.2.9 PDL/Ada (IBM Federal Systems Division)

PDL/Ada is administered by the Federal System Division's (FSD) Ada Coordinating Group (ACOG). Prior to the development of PDL/Ada, the FSD had trained approximately 2300 programmers in the use on a non-Ada PDL, called simply PDL. In an effort not to waste the experience these programmers had already acquired, and to facilitate their learning a new design language, PDL/Ada has been gradually evolved from PDL.

4.2.9.1 Syntax/Semantic Features of PDL/Ada

PDL/Ada conforms closely to Ada although there are some differences that may be the remnants of the earlier PDL terminology. For example, PDL/Ada allows concurrent assignments. These are groups of assignments that would affect each other if executed sequentially. For example, SWAP(x,y) might be written as two concurrent assignments x := y and y := x. Since these are not semantically correct Ada, they are written as Ada comments. During coding, the concurrent assignments are replaced by a set of assignments using temporary variables, for example, t := x; x := y; y := t;. PDL/Ada also

deviates from Ada when the condition in a loop statement occurs in the middle of the loop. Compare the following:

Ada	PDL/Ada

loop
  statement-group-1
exit when
  condition;
  statement-group-2
end loop;
	loop
	  statement-group-1
	exit when
	  condition
	  or else
	  statement-group 2;
	end loop;

PDL/Ada provides several mechanisms to allow deferred design decisions. There is a standard procedure, ST, and standard Boolean function, CD, that can be substituted for statements or conditions respectively while the purpose of the designer is contained in an adjacent comment. Additionally, the Ada package STANDARD is extended to provide generic data types and operations for stacks, queues, sequences, lists, and strings. This allows these object to be treated as abstract types.

The PDL also allows postponing decisions on how to implement design units. Initially, designs are structured in terms of a unit called a module. At some point in the design process, the Ada program units PACKAGE, PROCEDURE, TASK, or FUNCTION are substituted for each module.

Commentary in PDL/Ada is structured according to its positional association with Ada statements. In addition to ordinary commentary, PDL/Ada provides logical and descriptive comments. Logical comments are divided into three types: invariant, status, and data. Guidelines are provided to specify when each type of commentary is used, and the form the commentary will take.

The developers of PDL/Ada recommend that any expressions that deviate from Ada should be retained as comments when correct Ada code is written. In the common situation that more than one line of Ada is written for a single line of PDL, the developers recommend the use of the Ada BEGIN/END block to mark the Ada lines generated from the PDL line.

PDL/Ada is not compilable until all non-Ada PDL lines have been replaced.

4.2.9.2 Automated Support Features of PDL/Ada

No automated tools are provided.

4.2.9.3 Methodology Features of PDL/Ada

PDL/Ada is intended to be used with the FSD system design methodology. In particular, the PDL is used to support the FSD modular design, data design, and program design techniques.

Following these practices, the FSD approach defines four levels of design. The first level forms the user contract. It is concerned with using state

machine models to design Computer Program Configuration Items (CPCIs). State machine diagrams and PDL/Ada are used to record the CPCIs. In Level 2, functional and data design techniques are used to refine the PDL/Ada CPCI descriptions into Computer Program Components (CPCs). The remaining levels use stepwise refinement to elaborate the PDL/Ada functional and data designs. The products at the third level are independent of the operational environment, whereas Level 4 products are fully targeted to the operational environment.

IBM FSD has used PDL/Ada in the B-Level and C-Level specifications defined in DoD standards.

4.2.9.4 Documentation Features of PDL/Ada

The single large reference manual for PDL/Ada assumes a familiarity with IBM FSD software development methodology and builds on this background to describe the use of PDL/Ada features in design. The text makes good use of examples to illustrate each feature as it is introduced. The methodology makes use of predefined abstractions such as stacks, queues, sequences, sets and lists. The use of these abstractions is explained and the appendices define Ada package interfaces for the library packages corresponding to these abstractions.

The manual also includes a BNF definition of the PDL syntax.

4.2.10 RN ADAP (SofTech, Inc.)

RN ADAP is the version of SofTech's ADAP used for the Regency Net program. Consequently, the discussion here focuses on the differences between the two Ada-based PDLs.

4.2.10.1 Syntax/Semantic Features of RN ADAP

RN ADAP belongs to the group of Ada PDLs that employ Ada syntax without major modification. However, RN ADAP adds two significant semantic constructions.

The first extension concerns the use of structured headers that precede each design unit to capture supplemental design information. There are three different kinds of headers: Procedure/Function Headers, Task Headers, and Compilation Unit Headers. Each header identifies the information that must be supplied for the types of design unit it accompanies. The examples given for the headers indicate that information is recorded according to formatting rules that not only make the information easy to read, but would facilitate automated processing of the contents. Each line in the headers start with the special symbol (--//). Consequently, they are ignored by the compiler.

RN ADAP differs from ADAP by introducing "abstract identifiers" to allow postponing design decisions. According to the reference manual:

> "These are syntactically-correct Ada identifiers that represent operations or definitions intended to (be) more fully defined in alater iteration."

However, these abstract identifiers are undefined and, if compiled, would be reported as errors by the compiler. Thus each abstract identifier must be removed during coding.

4.2.10.2 Automated Support Features of RN ADAP

No automatic tools are provided for RN ADAP. An appendix to the reference manual mentions techniques by which the "abstract identifiers" can be made acceptable to the Ada compiler. An Ada compiler could then be used to check the syntax and develop a cross-reference table.

4.2.10.3 Methodology Features of RN ADAP

RN ADAP was designed for use in both the preliminary and detailed design phases. Like ADAP, its developers recommend that it is used with the Structured Design method developed by Constantine and Yourdon.

4.2.10.4 Documentation Features of RN ADAP

The RN ADAP manual is derived from the ADAP manual described earlier in Section 4.2.2.4. The manuals have in common the same approach to PDL and the same concern for clear explanation.

There are semantic differences between ADAP and RN ADAP concerning deferred specification of identifiers. Consequently, the two manuals provide differing recommendations in several places. In addition, all of the examples taken from ADAP have been rewritten in RN ADAP to reflect the RN ADAP style.

All other changes concern additional material that is found in the RN ADAP manual only. Each feature of RN ADAP is coupled with an explanetion of how that feature might guide the implementation of Pascal code. In addition, the RN ADAP manual provides guidance on project naming conventions. Finally, the manual closes with a large case study and shows how B5 and C5 specifications are developed from the RN ADAP design text.

4.2.11 SAIC Ada PDL (SAIC)

The design of SAIC Ada PDL was influenced by three primary objectives. These objectives were the following: to capitialize on the software engineering advances offered by Ada; facilitate the transformation of a design into Ada code; and promote the reusability of software by establishing an Ada PDL library that contains PDL descriptions of common functions. Where those

functions are implemented in Ada, the library also holds the corresponding Ada packages.

4.2.11.1 Syntax/Semantic Features of SAIC Ada PDL

SAIC uses standard Ada syntax for their Ada-based PDL and refers users to the Ada LRM for syntax rules. However, the use of certain Ada language features is deferred during design activities. The PDL also provides predefined packages that allow treating message queues and stacks as abstract structures. Unlike many Ada PDLs, the SAIC Ada PDL is intended to be compilable at all times. For this reason, all deferred design decisions and additional design remarks are entered as Ada comments in the PDL.

SAIC Ada PDL uses a standard header format, called a unit preamble, to capture supplemental design information. Each line in the unit preamble starts with the special symbol (--*). Consequently, the preamble is ignored by the compiler, but easily distinguishable by the human reader. It records configuration control information, identifies the requirements met by the unit, and specifies the unit interface.

4.2.11.2 Automated Support Features of SAIC Ada PDL

In addition to the Ada PDL described here, SAIC markets a design language known as SAI-SDDL. The Software Design and Documentation Language (SDDL) has been used and refined over a period of time beginning with an initial development at the Jet Propulsion Laboratory. SAI-SDDL is supported by an automated tool and can be used with four languages, FORTRAN, FORTRAN-77, Pascal, and SIMSCRIPT.

A set of Ada keywords has been developed for SAI-SDDL but the keyword set has not yet been released. Consequently, it has not been possible to evaluate the use of the SAI-SDDL tool set for SAIC Ada PDL. Rather than omit the features of the tool, SAI-SDDL is treated as a special case in the tabular tool comparison in Table 4-6. In this table, the features of the SAI-SDDL tool are listed but the status column shows that they are not yet available for SAIC Ada PDL. Since the features are operational in another PDL context, they may be available for the Ada-based PDL shortly.

Additional tools specifically intended for SAIC Ada PDL are under development. These include an automated configuration control manager, a documentation tool to extract and list information from source code and commentary, and additional text formatters.

4.2.11.3 Methodology Features of SAIC Ada PDL

This PDL is intended to support the detailed design phase. The developers recommend that it is used in conjunction with some design method, but do not specify any particular ones. They also stress the importance of stepwise refinement and structured walkthroughs.

4.2.11.4 Documentation Features of SAIC Ada PDL

SAIC Ada PDL is described by a single manual that incorporates tutorial information integrated with examples, style guidelines, and a case study. This manual also lists a few standard packages that can be assumed by the designer.

Since SAIC Ada PDL uses the full Ada syntax, reference material concerning syntax and semantic features is not required. The PDL user is referred instead to the Ada LRM. The SAIC Ada PDL manual does provide two useful tables that classify Ada features as 1) appropriate, 2) limited use, or 3) inappropriate depending upon whether the design phase is in an early or late stage of development. These tables capture succinctly the difference between Ada as a PDL and Ada as a coding language.

A complete manual is available for the current SAI-SDDL PDL automated support tool. Some of this material will be applicable to the Ada-based PDL; for example, the manual documents the extensive set of formatting directives that control the flexible document generator.

4.2.12 Sanders Ada/PDL (Sanders Associates, Inc.)

This Ada-based PDL was designed to utilize Ada as much as possible, and allow flexibility in the use of the PDL for any given development effort.

4.2.12.1 Syntax/Semantic Features of Sanders Ada/PDL

The Sanders PDL syntax comprises full Ada plus structured English narrative. A structured English statement may be substituted wherever an Ada statement, expression, or type definition is expected. Alternatively, the abbreviation TBD may be used. This symbol TBD is treated as a reserved word.

Sanders Ada PDL cannot be compiled so long as the design text contains TBDs or English narrative. However, the text is compilable when these are replaced during coding activities.

4.2.12.2 Automated Support Features of Sanderss Ada/PDL

The Sanders Federal Systems group has developed the following tools: a lexical/syntactic analyzer, a pretty printer, and a cross-reference utility. A semantic analyzer is also under development. At present, no documentation is available for these tools. Consequently, Table 4-6 shows only those tools that seem to correspond in an obvious way to the categories of the table.

4.2.12.3 Methodolgy Features of Sanders Ada/PDL

The use of Sanders Ada/PDL is not restricted to any particuler software development method. Indeed, one of the requirements for the PDL documentation was that it should promote the portability of the PDL to a variety of methods. However, the developers do discuss the use of Structured Analysis and Structured Design to specify and design the software expressed in Sanders Ada/PDL.

Use of the Sanders Ada/PDL is recommended for preliminary design and detailed design activities.

4.2.12.4 Documentation Features of Sanders Ada/PDL

Only a reference manual is provided at present. The organization of the manual follows exactly the outline of the Ada LRM. Therefore, the reader can easily check which syntax features are supported by the PDL.

At the time of this writing, no documentation has been released for the automated tool support environment.

4.2.13 TI Ada PDL (Texas Instruments, Inc.)

The TI Ada PDL was developed to meet government requirements for use of Ada-based PDLs and the developers of the PDL have paid considerable attention to how the PDL can used with various government standards. This is reflected in the both the features that the TI Ada PDL provides for capturing supplemental design information and the PDL documentation.

The PDL is currently undergoing review by the Ada Technology Branch, Advanced Computer Systems Lab at Texas Instruments.

4.2.13.1 Syntax/Semantic Features of TI Ada PDL

At this time, the syntax of TI Ada PDL follows Ada without deviation. However, a package of TBD type definitions is currently being developed which will allow postponing the implementation of data types during design. Since the main concern of the PDL designers seems to be the smooth integration of the PDL with Texas Instrument's project management and system design methods, other major changes in syntax are unlikely prior to final approval.

TI Ada PDL follows the Byron approach to the capture of supplemental design information and annotations are expressed as structured comments starting with the special symbol (--|). These structured comments are used to express information on the relation of each design unit to the work breakdown structure (WBS) of the project and other project-oriented data.

TI Ada PDL also uses structured comments to flag logical groups of Ada design text, For example, one annotation flags the WITH clauses used to import externally defined program units, and another precedes the declaration of types and objects used in a global sense within a compilation unit.

Standard templates are provided to identify the different kinds of supplemental design information that must be given for system, CPCI, CPC, and subprogram design units.

4.2.13.2 Automated Support Features of TI Ada PDL

The developers of TI Ada PDL plan an extensive tool development program. Not only will these tools cover most of the categories listed in Table 4-6, but there will be many additional tools. These other tools fall into three

general categories: structure analysis tools, requirements tracking tools, and project visibility tools.

One of the tools planned by Texas Instruments is a text conversion tool. There are no statements in TI Ada PDL that require text conversion owing to a variance with Ada syntax. Instead, the purpose of this tool will be to convert Ada commentary into comments acceptable by processors for other languages such as C or Fortran.

4.2.13.3 Methodology Features of TI Ada PDL

The TI Ada PDL was developed for use in the development of DoD software and this is reflected throughout the design and documentation of the PDL, In particular, it is designed to meet the government visibility requirements defined in the Work Breakdown Structure (WBS) and can support any of the three (WBS) formats defined in MIL-STD-881A.

Additionally, the PDL is intended to support all the contemporary design concepts, with particular focus on abstraction, decomposition, information hiding, stepwise refinement, and modularization. It provides support for four levels of software system description:

1) Software System.
2) Computer Program Configuration Items (CPCIs).
3) Computer Program Components (CPC).
4) Subprogram.

The software developers at Texas Instruments take a realistic view of the software life cycle and assumes a cyclic development process. Thus, a project may start with the development of an Advanced Development Model (ADM). This allows the feasibility of the system to be investigated and provides a prototype that can be analyzed. The information gathered from the ADM is then used to develop the Engineering Development Model (EDM). The EDM is eventually fielded as the final system.

Two design processes are recommended for designing the software in the ADMs and EDMs. These are the Figure-8 Design Process and the Three-Leaved Rose Design Process. Both define a number of iterative steps and milestones. The TI Ada PDL can be used to express the designs developed using these processes.

4.2.13.4 Documentation Features of TI Ada PDL

Currently, only a preliminary form of the reference manual for TI Ada PDL is available. The table of contents indicates that the completed manual will exhibit all the documentation characteristics outlined in Section 3.2.3, but much of the corresponding text has yet to be written.

The current version of the manual includes an extensive explanation of software development methodology and the proposed relationships between the methodology, PDL, and tools. It provides an explanation and justification of the development process that is suitable for a knowledgeable reader, but does not constitute instructional text for a new user. However, it can be expected that suitable introductory material will be provided when development of the PDL is more advanced.

4.2.14 TRW Ada PDL (TRW)

Experience with an earlier non-Ada PDL was used to guide the determination of the requirements for TRW's Ada PDL. Use of the earlier Caine, Farber, Gordon (CFG) PDL had resulted in many benefits, including a 15% productivity gain. The new Ada-based PDL is designed to keep the syntactic simplicity of the CFG PDL, but include those Ada design features that support the preliminary and detailed design phases.

4.2.14.1 Syntax/Semantic Features of TRW Ada PDL

TRW Ada PDL differs lexically from Ada in that Ada PDL statements are significant only at the start of a line. If a line does not start with a statement, it is assumed to contain only narrative design text.

Designs in TRW Ada PDL are subdivided into units which may be declared as PROCEDURE, FUNCTION, TASK, and PACKAGE or which may be declared as a MODULE or as a FRAGMENT in order to defer a decision on the kind of subunit. Following the philosophy of Ada, the subunits consist of a specification and a body. Unlike Ada, the body parts of nested units may not be nested together. Instead, the IS SEPARATE alternative must be used to separate unit bodies. The use of IS SEPARATE facilitates the enforcement of separation of design levels.

The specification part consists of a subunit name, parameter list, and type and object declarations. The names and parameter lists follow Ada syntax, The remaining declarations either follow Ada syntax or use a narrative description. Incomplete type definitions are allowed. Deferred name declarations are allowed; a 'deferred name' is one used in a recognizable context (e.g., type, object, and unit names) where Ada would require prior declaration; this leads to implicit declaration in Ada PDL and is regarded as a recognized name for further design analysis.

The body part consists of design narrative that is organized by PDL keywords. These keywords provide the full structured constructs of Ada. Within the design narrative text, the PDL processor will recognize and report the use of defined object names and types.

The visibility rules of TRW Ada PDL are the same as Ada. However, the PDL also provides an IMPORT declaration which is similar to the combined effect of WITH and USE.

Although in its broadest definition, the TRW Ada PDL is not directly compilable, it can be used in a compilable form with no loss of capability by imposing certain restrictions on its usage.

4.2.14.2 Automated Support Features of TRW Ada PDL

The TRW Ada PDL is currently supported by a tool set which, using a building block approach, can be configured as a single tool or as a group of tools which can execute concurrently. If desired, the user can disable some of the documents normally produced on a single pass over the design code. The support tool also maintains a library of design modules that have passed syntax

tests. TRW emphasizes the library aspect of the tool as the key to multiple run execution efficiency, project coordination, and future software design reusability.

In view of the non-compilability of TRW Ada PDL, an initial translator would be useful. Such translators are under development for Fortran, Jovial, and Ada. Since the PDL syntax omits Ada features that are not design oriented, there is no need for a design level enforcer. Nevertheless, there are plans to produce a "Deferred-Development Report" which may be useful for testing the degree of completion of a design. Additional tools under development include: a global interface consistency checker, graphical structure chart tools, and C5 specification generators.

A rehost of the VAX/Unix tool software to a VAX/VMS operating system is in progress.

4.2.14.3 Methodology Features of TRW Ada PDL

TRW Ada PDL is specifically designed to provide support for both the preliminary and detailed design phases in the software life cycle. Though the PDL is currently intended to be independent of any particular software design approach, TRW is undertaking several activities that will provide guidelines for using the PDL with certain methods and methodologies. For example, one of TRW's contracts is addressing the integration of the Ada-based PDL with TRW's Distributed Computing Design System (DCDS) methodology.

The PDL can be used to develop documentation in accordance with various government standards.

4.2.14.4 Documentation Features of TRW Ada PDL

The TRW Ada PDL is described in a single, comprehensive document. The organization of the material and the accurate table of contents simplify reference to any given topic.

In Section 2, the major reference section, the TRW manual combines tutorial material, reference material, and PDL examples in a way that should be effective in teaching the use of the Ada PDL. A formal definition of the PDL grammar is given in an appendix.

The table of contents of the manual reviewed for this survey lists a few sections that are not included in the text. These sections should appear in future updated versions. The material in question includes design case studies and language compatibility and translation suggestions for Ada, Assembler, C, CMS-2, COBOL, FORTRAN, JOVIAL, and Pascal.

4.2.15 WIS Ada PDL (WWMCCS Information System)

The WIS Ada PDL Standard provides guidelines for using an Ada-based PDL in the development of software for the World Wide Military Command and Control System (WWMCCS) Information System, or WIS. Ada-based PDL are intended to be used for the development and support of WIS software in accordance with the

standard. The procuring organization is expected to tailor the standard appropriately for each individual acquisition.

This PDL was developed for the WIS program by the Strategic Systems Division of GTE Government Systems Division. Currently it is the only PDL that provides special features to capture information specifically pertaining to the reuse of designs and measurement of various software attributes.

4.2.15.1 Syntax/Semantic Features of WIS Ada PDL

WIS Ada PDL uses Ada syntax and adds annotation through structured commentary. Its overall approach is therefore similar to Byron. WIS Ada uses the symbol (--%) to introduce lines containing annotation keywords and the symbol (--|) to precede lines that provide the details appropriate for eech annotation. Some of the information given in these structured comments is formatted to facilitate automated processing. Standard templates are provided to identify which pieces of supplemental information should be recorded with each type of Ada program unit.

An interesting innovation is that the WIS Ada PDL is concerned with providing information that can facilitate reuse of design units. This has two consequences. One is that an initial set of package classification is given. These classifications are compatible with an object-oriented design approach and identify a package as: a declaration group, an operational abstraction, a state machine, an abstract type, or an abstract object. Standard templates are given for these package classifications.

The second addition made to promote reusability is the provision of a notation for specifying package keywords. These keywords are chosen from a common dictionary and are intended to associate different packages that have properties in common. A keyword list can be used as a search list to identify packages that meet certain needs.

The WIS Ada PDL also provides a set of structured comments that can be used to collect software metric data. This data is primarily for use in applying Albrecht's Function Point Complexity Metric, but can be used for other complexity measures.

The need for structured comments that support verifying whether a design text conforms to a security model in a security sensitive application is identified. Some suggestions as to appropriate annotations with the capability for expressing first order logic are given, but no actual syntax rules are provided.

Additionally, the PDL provides a construct that allows grouping logically related definitions and associating a name with the group. The construct is implemented as a structured comment and can include additional types of structured comments which describe the properties common to the definitions enclosed by the construct.

A package of To-Be-Determined expressions and conditions is provided to allow expressing deferred decisions.

4.2.15.2 Automated Support Features of WIS Ada PDL

The WIS Ada PDL Standard describes a set of tools that will be developed or purchased to support the use of WIS Ada PDL in the development of WIS software. At present, these tools are not available.

4.2.15.3 Methodology Featurees of WIS Ada PDL

The WIS Ada PDL is independent of any particular design method. However, the similarity of the PDL to Ada is expected to result in the application of those software engineering concepts embodied in Ada.

The standard recommends using Ada-based PDL not only for the preliminary and detailed design phases in the software life cycle, but also for system design. It presents a generalized program design model consisting of three phases: preliminary, detailed, and algorithm/data design. Contractors developing and supporting WIS software are expected to map this design model to their own design method.

4.2.15.4 Documentation Features of WIS Ada PDL

The documentation for WIS Ada PDL consists of a single document that can serve both as a tutorial and a reference manual. The currently available document is intended to guide the use of the PDL on several software projects contributing to WIS. The individual projects may want to create two separate documents for internal use, one containing reference material for answering questions in every day use, and the other containing tutorial material for use during training. Although the present document contains some examples, more cases need to be treated to illustrate all of the points of the PDL usage.

A strong point of the WIS Ada PDL manual is its clear and well reasoned discussion of the purpose of the PDL and its relationship to design methodology. The manual describes the role PDL plays in the software life cycle as well as more detailed issues such as recommended PDL conventions for object-oriented programming.

4.2.16 XAda (GTE Sylvania Systems Group)

The philosophy of XAda is to "design a little -- code a little". The PDL requires execution of design text to provide dynamic design checking and assessment of expected resource loading. Other objectives for XAda are to support automated extraction of design commentary for documentation purposes, and to allow utilization of existing and planned commercially available tools.

4.2.16.1 Syntax/Semantic Features of XAda

XAda is an example of an Ada-based PDL that uses full Ada syntax without modification. In fact, XAda is the result of a survey of Ada-based PDLs that concluded "The Ada language could be used per se as a design language with no extensions or restrictions."

XAda extends Ada with mechanisms that allow capturing supplemental design information in a manner similar to Byron. Each extension consists of the special symbol (--|), a keyword, and English narrative.

4.2.16.2 Automated Support Features of XAda

A tool called SLICE, the Source Language Independent Comments Extractor, is under development for VAX hardware. The Byron tools (Section 4.2.4.2) are also being considered for use with XAda.

4.2.16.3 Methodology Features of XAda

The developers of XAda intend the name "XAda" to refer to more than just a design language. Instead, XAda is regarded as a method for using Ada as a design language.

The XAda approach defines three steps for the design process: high-level design, detailed design, and implementation. The particular design techniques and strategies used with the PDL are left to be chosen by the project team. The examples given of such techniques are data flow diagrams and structure charts. This flexibility is intended to allow an appropriate selection to be made for each project.

4.2.16.4 Documentation Features of XAda

The documentation presently consists of an article to be published in Ada Letters. The article describes the method of use of the PDL. The syntax of the PDL is identical with Ada. Therefore the Ada LRM can serve as a syntax reference manual.

5.0 COMPARISON OF Ada-BASED PDLS

When these Ada-based PDLs are reviewed together, one immediately receives the optimistic impression that each developer is participating in a movement within software engineering that is widely accepted as productive and promising. On a detailed level, there is considerable divergence in the form the PDLs take. In one sense this disagreement is natural since the field is addressing several research issues. On the other hand, the differences are largely caused by genuine problems that have not been solved in a universally satisfactory manner.

This section starts with some discussions of how Ada-based PDL developers are responding to some of the major problems in this field. The last part of the section provides a more detailed comparison of the PDLs following the four major areas of interest identified earlier.

5.1 Increasing Productivity

The earlier sections of this report have given a sense of the unifying aspects of Ada-based PDLs. At this point it is timely to discuss one of the major problems in software development and the varying responses of the PDL developers.

This first problem concerns productivity. Software seems to be one of the most expensive products in today's market place. Even so, the demand for software products is increasing every year and various projections indicate that there will soon be an acute shortage of software developers.

Consequently, one of the major concerns of the technical community is to find ways of developing software cheaper and faster, or in other words, increasing productivity. Of course, productivity is not simply a matter of increasing the speed with which software developers produce code. Other issues, such as quality, are inextricably intertwined. For example, saving several man-years of effort is of no consequence if the final product fails to satisfy reliability and security requirements. On the other hand, a high quality product means that there are fewer errors to fix and the product will be completed sooner.

The question now arises: How will the use of PDLs impact productivity in design activities? Only one developer, TRW, ventured an estimate of the possible gains. TRW states that they have found that PDL use improves their productivity by 15%. (This figure was given in reference to the earlier non-Ada PDL.) This amount is economically significant in a large project. Nevertheless, it remains small compared to such productivity factors as programmer/designer experience, the support environment, and clarity of the original requirements specification. In particular, the Ada-based PDL developers discuss the issues of training and translation of the design text to Ada code.

The reaction of one group of Ada-based PDL developers, which includes TRW, IBM, and MAYDA, has been to impose Ada syntax on the PDL which has served the organization well in the past. In this fashion, the organization's investment in training is preserved and the productivity return from the PDL is more certain.

A second group Ada-based PDLs, which includes Byron, GDADL, TI Ada PDL, and WIS Ada PDL, uses the Ada syntax as a starting point for developing a PDL. During the transition phase when Ada skills are still being developed, an organization using one of these PDLs may have difficulty because Ada is unfamiliar to the designers. However, Ada training will usually be required in any case for coding activities, so that the introduction of Ada at the design phase is arguably beneficial.

In the context of the productivity problem, one might also include in this second group those PDLs which follow Ada but allowed unrestricted use of English narrative and TBD expressions. Although these PDLs are not immediately compilable, they share the same attitude toward the training/productivity issue as PDLs in the second group. Members of this latter subgroup include ADAP, BTO, RN ADAP, and Sanders Ada PDL.

In a third group there are PDLs which introduce significant variations from Ada without appearing to base the syntax on a previous PDL with a large existing community of PDL users. Often, these extensions are stated with conviction, for example, that procedure and task calls must be discriminated in the code or that the designer needs non-Ada abstractions such as a loop on abstract sequence types. However, from the productivity point of view, there seems to be a burden of proof placed on this third group to show that there is a net productivity gain after training designers in the use of Ada extensions. Examples in this group include JPDL and LTH Ada PDL.

With the current lack of empirical data, it is difficult gain after training designers in the use of Ada extensions. Examples in this group include JPDL and LTH Ada PDL.

Additionally, PDLs which adhere to Ada syntax and semantics can facilitate the transition from design to Ada code, and this can increase productivity. However, when Ada is not the implementation language, these gains could be lessened or even reversed.

5.2 Increasing Reliability

To be reliable, a software system must be free from error and always perform as expected. Therefore, a PDL should help to prevent errors being made during design and coding activities, and help to ensure that the design not only meets requirements, but includes proper error handling to cope with any failures during operation. In its simplest terms, this can be reduced to validation and verification.

Validation of a PDL design can be supported by identifying the requirements met by individual design units. Almost half the PDLs provide mechanisms that can do this. Information that facilitates verification of the design text is less often captured. harris ada pdl and lth ada pdl allow the specification of testing requirements; and adap, jpdl, and rn adap all provide mechanisms for stating preconditions and postconditions for design units, jpdl also allows specifying task communication protocols and identifying modules that are used concurrently, which is also useful for verification.

Information about timing and sizing constraints, hardware interfaces, and software environments can support both validation and verification. ADAP, Byron, Harris Ada PDL, JPDL, and RN ADAP all express timing and sizing constraints. While only ADAP and RN ADAP provide mechanisms for stating hardware interfaces. BTO Ada PDL and JPDL capture information about the software environment in which the eventual code will execute.

The error handling required by a design module can be highlighted by identifying the exceptions raised or handled by a design unit, and the error messages it produces. Information for the first of these, exceptions, is captured by several PDLs. Whereas only Byron provides a facility for specifying the error messages issued.

Putting aside these detailed considerations of supplemental information, the current set of Ada-based PDLs are a far step away from supporting full validation and verification of designs. Indeed, when appropriate information is captured, such as the requirements met by a design unit, there is no attempt to fully use this by automating the bracking of requirements. In general, these PDLs could be extended by adding notations that would permit some types of design analysis, for example, checking for freedom from deadlock. Additionally, verification languages like ANNA could be included in any PDL that adheres to Ada syntax and semantics.

Similarly, use of a compilable Ada-based PDL introduces the ability to execute a design and gather information for assessing the expected operation of software that will be produced. This topic was only discussed for XAda, and even then the full potential of an executable design was not exploited.

Admittedly, these are relatively new areas in software engineering, but even so, the opportunities offered by Ada-based PDLs for increasing software reliability seem to be largely ignored.

5.3 Promoting Reusability of Software Designs

The current approach to developing software is to build each new system from scratch. The software developer is one of the few professionals who does this. Alternatively, if a software product is largely constructed from existing parts, not only will productivity be increased, but the use of proven software will improve quality also. This is also a new area in software emgineering and is the focus of many research efforts. While it is still uncertain what impact this consideration will have on the design process as a whole, there are some ways in which a PDL can increase the ease of reusing software designs.

To some extent, Ada itself addresses this issue and those PDLs that encourage use of the Ada package and generic features will benefit accordingly. The Ada-based PDLs discussed in this report all support packages, even those which do not follow Ada closely in other respects. However, the TI Ada PDL developers recommend deferring the use of generics until the code and unit testing phase.

The provision of a library manager for building and maintaining a library of reusable designs also facilitates reusability. Byron and TRW Ada PDL both provide library managers which can be used to maintain a library of reusable designs. In the case of TRW, the library manager is specifically intended for this purpose.

Reusability of PDL designs is emphasized by SAIC Ada PDL, TRW Ada PDL, and WIS Ada PDL. However, WIS Ada PDL is the only one that explicitly includes features to promote reusability. This is done in two ways. Initially, the PDL provides a scheme for categorizing package design units and provides templates to indicate the kinds of supplemental design information that should be given with each category of package. At the moment, only five categories resulting from an object-oriented design approach are specified, but the intention is to expand this in the future. Secondly, WIS Ada PDL provides an annotation for recording keywords that identify the essential properties of packages. This allows a data base of package keywords to be built which can be searched to identify packages that perform a particular function or relate to a certain topic.

5.4 Minimizing Support Costs

Another major problem facing the software community is post-deployment support. If overall software costs are examined, it can be argued that the development productivity issue is secondary in importance to the support cost issue. In this case, the main thrust of Ada-based PDLs should be to ease the support burden. There are several ways in which a PDL can do this. It should: 1) identify the requirement(s) that are met by each design unit; 2) explicitly record the dependencies between design units; 3) structure the design units to hide unnecessary information; 4) document the design to allow rapid and accurate modification; and 5) ensure that inconsistencies do not develop between the design and code.

The above issues are addressed individually in the following subsections. However, they are not presented as the only ways in which a PDL can facilitate software support. For eample, the ability to include a change log, overview descriptions, technical references, or preconditions and postconditions are always helpful.

5.4.1 Identification of Requirements Met

Explicit specification of the requirements met by each design unit can support carrying out software modifications in two ways. It helps to identify which design units must be changed to correspond to a change in the requirements. It also helps in validating a design module after it has been modified to ensure that it still satisfies the appropriate requirements.

Several of the Ada-based PDLs provide mechanisms for capturing information about the requirements met by each design unit. These are ADAP, Harris Ada PDL, RN ADAP, SAIC Ada PDL, TI Ada PDL, TRW PDL Ada, WIS Ada PDL, and XAda. The general form of the information recorded is a reference to the relevant chapter and paragraph in the requirements specification document.

It is also desirable that the PDLs provide a facility for recording any timing and sizing constraints that are imposed by the requirements and that impact individual design units. This is useful information for both development end support activities. From the support viewpoint, it can help to explain certain aspects of a design and identify constraints that must still be satisfied by the modified design.

5.4.2 Recording Dependencies Between Units

All too frequently, when a change is made to one design unit, several others also require updating. Consequently, it is desirable that the PDLs annotate each design unit with a list of all external data entities and program units that are referenced by the design unit. ADAP, Byron, Harris Ada PDL, RN ADAP, TI Ade PDL, WIS Ada PDL, and XAda all provide this capability, at least partially.

Cross-reference tools can also be used to provide this information and of all the tools that are currently available for use with the Ada-based PDLs, this tool is the most common. The PDLs that are supported by a cross-referencer are ADA/SDP, Byron, GDADL, Harris Ada PDL, SAIC Ada PDL, Sanders Ada PDL, and TRW Ada PDL. A cross-referencer is also under development for TI Ada PDL.

5.4.3 Hiding Unnecessary Details

To a large extent, the hiding of unnecessary details is the result of good use of the software engineering principle of information hiding in conjuncion with the principles of localization and modularity. Therefore, the design method used in conjunction with the PDL is important. Of all current design methods, the one most attuned to this issue is the Software Cost Reduction Method developed by Dave Parnas. Although none of the Ada-based PDLs addressed in this survey mentioned this method, there is no reason why any of them cannot be used in conjunction with this, or any other method.

Information hiding is another benefit of adherence to Ada. This is because the specification parts of Ada program units were designed to enforce hiding of the implementation of data types and operations on those types.

The provision of design abstractions can facilitate making modifications to a sofware design by postponing design decisions. Not only does this help to limit the amount of information the software developer must review in order to understand a design, but it also increases the likelihood of a change being limited to the code and not affecting the design itself. Most of the

Ada-based PDLs provide mechanisms for postponing the specification of the implementation of types, The only ones that do not are BTO Ada PDL, Harris Ada PDL, and XAda. Similarly, the PDLs generally provide mechanism for postponing the implementation of operations. In this case, the PDLs that do not are BTO Ada PDL, Byron, LTH Ada PDL, TI Ada PDL, and XAda.

In those cases where no special features are provided for postponing decisions, the Ada comment can be used. However, this is likely to result in a host of error messages if an attempt is made to use the compiler to check the syntax of the design text. Alternatively, software developers could provide their own package of TBD definitions.

5.4.4 Documenting the Design

Good documentation of the design helps the software developer to understand the purpose of the current design and determine how it needs to be changed. Automated document generators are useful for developing accurate and cost-effective documentation. They also help to update the documentation once changes have been made. The only PDL that has such a tool is Byron, although the developers of TI Ada PDL, TRW Ada PDL, WIS Ada PDL, and XAda plan to make these tools available in the near future.

Another tool that is useful in this context is a design annotation summarizer. This tool extracts supplemental design information from PDL text to produce a report. Byron and GDADL currently have this tool, and it is among those recommended for WIS Ada PDL.

5.4.5 Preventing Inconsistencies

Modification is likely to occur during all phases of the software life cycle and can easily lead to inconsistencies between PDL design text and code. The type of consistency problem that arises will depend upon whether the project keeps separate PDL and Ada code units. If so, changes must be made to both. Ideally, methods should exist to validate the consistency of the different units. Alternatively, the design unit can be progressively refined until the Ada code is produced with the PDL still embedded as commentary.

Most PDLs can be used with a single unit approach. For example, PDL/Ada recommends that each PDL line should be retained as a comment if it is to be replaced by code. If coding requires several statements for each PDL line, PDL/Ada even describes how the Ada lines should be grouped to show clearly the association with PDL statements.

However, when this single unit approach is used, a new problem arises; namely, how shall new staff become familiar with the design of the system after the PDL has been encrusted with implementation code? This can be resolved by extracting a design summary for use in explaining the system to new staff. The Ada-based PDLs that define an extractable type of commentary offer a significant advantage in this case. Such PDLs include Byron, GDADL, TI Ada PDL, and WIS Ada PDL.

5.5 Comparison of Detailed Features

This subsection compares the Ada-based PDLs on the basis of the characteristics recorded in the tables in Section 4.1. The following discussions also serve to summarize the current state-of-the-art in this field.

5.5.1 Comparing Syntax/Semanntic Features

In Section 5.1, Ada PDLs were compared with respect to their underlying syntax. In terms of the larger scope of design language features, there is remarkable agreement among the PDLs that Ada program units (packages, procedures, functions, and tasks) are suitable as units of design. There are some minor innovations in the design units but the mapping to Ada program units is usually clear. Whereas design units are used to structure the abstract elements of a system, partition work in a team development effort, and hopefully promote reusability, program units primarily address compilability issues. However, the commonality between design units and program units is not surprising. Instead, it reflects the concern of the Ada-based PDL developers to extend the advantages of Ada back into the design activiy and facilitate the transition to Ada code.

The trend in Ada language coverage seems to be to use full Ada syntax and defer the use of implementation oriented or lowlevel language features. Both PDL/Ada and SAIC Ada/PDL identify the specific Ada features that are suitable for use at different design levels. The remaining PDLs generally give guidelines as to the type of features that should be deferred.

In the few cases where a PDL uses a restricted Ada syntax, the language features omitted are the USE clause and the pragmas INLINE and SUPPRESS.

The majority of PDLs provide mechanisms for postponing specification of the implementation of data types, although mechanisms for postponing the implementation of operations are less frequent. The mechanisms range from the provision of packages of different TBD types, to use of a simple reserved word "TBD." Roughly half are compatible with Ada.

Among all the Ada-based PDLs, only GDADL, PDL/Ada, Sanders Ada PDL, and TRW Ada PDL do not provide annotations for capturing supplemental information. However, since most annotations are written in the form of Ada comments, there is no reason why these PDLs cannot adopt annotations defined in the other PDLs.

Most of the mechanisms for expressing supplemental design information are associated with templates used at the beginning of each design unit. The templates contain slots named by keywords where the designer inserts various attribute values. In semantic terms, each template is one long sentence with many clauses. For ease of use, the templates generally resemble fill-in-the-blank forms. Again, PDLs that do not define their own templates could borrow from those PDLs that do.

In those cases where the information captured by the templates is structured, tools could be developed to scan the design templates to analyze and summarize the named attributes. As yet, no PDLs provide a tool specifically intended for this purpose. However, some of the report generators could be used to produce a summary of the design annotations. Of those PDLs that do process annotations, only XAda allows the user to define additional keywords specific to his needs.

5.5.2 Comparing Automated Support Features

At the moment, the most popular tool appears to be the cross-reference generator. The listings from this tool are intended to locate where entities are defined and used. The activity of tracking down the location of a definition occurs so often during design that two of the pretty printers, those for ADA/SDP and GDADL, actually place the references on the same page as the PDL design text.

The least popular tool appears to be the syntax directed editor. Only ADAP and WIS Ada PDL plan to provide this tool. There are no plans to develop tools that can check the syntax of design text entered without the benefit of these special editors. Of course, it should be noted that the Ada compiler can be used to check syntax for those Ada PDLs which are directly compilable.

In addition to formatting design text, pretty printers recognize the basic block structure of a design and can be used to identify major structuring errors. The PDLs that have pretty printers are ADA/SDP, Byron, Sanders Ada/PDL, and TRW Ada PDL. While SAIC Ada PDL, TI Ada PDL, and WIS Ada PDL plan to provide this tool in the near future.

Several PDLs provide dictionary generators. The primary purposes of this tool are to facilitate completeness and consistency checking and support documenting the design. However, a dictionary generator can also help software developers and reviewers locate postponed decisions. Many Ada-based PDLs distinguish postponed decisions with reserved words. Some also annotate deferred definitions with some sort of narrative. The cross-reference facility can be used to find the position of the reserved words and narrative in the design text, and summarize this information in a data dictionary. Dictionary generators are now available for ADA/SDP, Byron, and TRW Ada PDL. They are also planned for TI Ada PDL and WIS Ada PDL.

One observation concerning dictionary generators is that many PDLs will be used in conjunction with a methodology that encourages the immediate development of a data dictionary. In these cases, the dictionary is usually partially developed before the PDL is introduced to the project. Automated support for checking the consistency of the early project dictionary and the PDL dictionary would seem desirable. Hopefully, tools to accomplish this will be provided in the near future.

Only Byron is currently supported by a flexible document generator. This is a very useful tool that can greatly ease the task of producing documentation, and keeping the documentation up to date. WIS Ada PDL and XAda also plan to provide document generators, and the developers of TI Ada PDL are developing a document generator that can produce parts of a

XAda also plan to provide document generators, and the developers of TI Ada PDL are developing a document generator that can produce parts of a MIL-STD-490 C5 Level Specification.

A surprising absence in the present offering of tools is the lack of code generators for those PDLs that are not compilable. Although this situation may reflect a principled decision to keep design and coding separate, at least a complementary effort to develop verification tools that could check the consistency of Ada code and PDL would seem to be in order.

5.5.3 Methodology Features

PDLs are largely method independent and so may be used with a variety of design methods. However, a PDL is essentially a tool for expressing a design and it is important that it is used in conjunction with some method that can lead the software developers through the design process.

GDADL, Harris Ada PDL, and PDL/Ada were all specifically developed for use with an in-house software development approach. The documentation for these PDLs includes materials discussing how they should be used with the relevant method.

Byron, JPDL, LTH Ada PDL, and TI Ada PDL all offer some guidance for using the PDL with a particular method. Whereas ADAP, RN ADAP, and Sanders Ada/PDL recommend the use of Structured Design, but give no suggestions on how the PDL is used in this context.

The Ada-based PDLs are similar in terms of life cycle coverage. In this report, it is a condition of their classification as Ada-based that they support the detailed design phase in the software life cycle. With the exception of BTO Ada PDL and SAIC Ada PDL, these PDLs are also intended for use during preliminary design.

Harris Ada PDL is additionally used for expressing software requirements. This is unusual because requirements specification languages are intended to portray a different type of information and, consequently, are often structurally different to design languages. However, the Harris Corporation uses their PDL in conjunction with a fairly sophisticated development methodology and this encourages correct use of the PDL.

The developers of JPDL state that this PDL can be used to express some of the contents of a Mil-STD-490 B-Level specification, but say this should be done cautiously and provide some guidelines for the process.

Finally, the WIS standard encourages use of the PDL for both system design and software design activities. Though its use in software design is discussed in depth, no guidelines are given for its use at the system level.

5.5.4 Documentation Features

Ideally, documentation should be helpful and appropriate to each person involved in a design project, no matter what the title, job description, or experience level of that person may be. In practical terms, there should be several documents keyed to different interests and needs. At present, all of the developers have a reference manual or are able to point to the Ada LRM as a reference. Thus, all developers have defined their Ada-based PDL and this is a good start. However, the successful introduction of Ada-based PDL on projects will depend on the development of additional kinds of documentation.

At the moment, the developers provide the most complete documentation for those aspect of PDL that are of most immediate concern to them and have deferred other writing tasks. Thus for Byron, the manual emphasizes the machine processible aspect of PDL. The Harris, IBM, and TI manuals emphasize the application of the PDLs to their respective methodologies. While RN ADAP and JPDL devote space to the translation of PDL design text to non-Ada lanaguages, and the LTH and WIS manuals explain the relation of PDL to government contract procurement procedures. Ideally, when the developers finish their job, each PDL will have documentation that suffices for all needs. Currently, none of the available documentation is sufficient to explain all issues relating to the use of Ada-based PDLs.

6.0 OTHER DESIGN LANGUAGES

Some of the languages identified in the survey are not described in the previous sections. These languages are omitted due to lack of documentation, or because they are not considered Ada-based PDLs. These languages are identified and described briefly in this section.

6.1 PDLs in Early Stages of Development

Two of the organizations contacted in the course of the survey responded that they are developing an Ada-based PDL, but as yet no information is publicly available. These were Caine, Farber, & Gordon, Inc. and IDL Tech. The second of these, IDL Tech, expects to have information available in the first quarter of next year.

Five other organizations currently have no formal documentation on their PDLs, but provided some brief notes. The remainder of this subsection describes these PDLs. All these PDLs are still under development.

6.1.1 ACK (Edward F. Hoover, III)

Ack is a highly generalized PDL that uses neutral and inflected tokens. These tokens are related to the concept of viewpoints of duty in a cycle of action as causes, actions, or results. Each of these may comprise a view of auxiliary or tangible referents. This leads to an abstract syntax closely

related to simple English grammar and homologous to class and subclass constructs, nested graphic objects, textual outlines, and relational tables.

A Socratic dialogue is used to address the role of the software developer in the dynamic context of his job, orient his conceptual approach, and elicit the necessary observations for analysis. These observations are recorded using graphic structures that support generically reducing the observations or extending them into increasingly detailed designs. A verbal verification procedure is provided.

The development of automated tools to support this approach and facilitate the transition to Ada code is being considered.

6.1.2 ATS Ada PDL (Advanced Technology Systems)

ATS is developing an Ada-based PDL that will be used and evaluated as an alternative to an existing PDL in an operation where the target language is COBOL. In this case, the Ada-based PDL will serve as a training path so that programmers who are already familiar with PDL can develop an understanding of Ada.

6.1.3 RCA Ada PDL (RCA)

RCA is developing an Ada-based PDL for internal use only. The PDL will support the full Ada syntax and provide additional structured comments. A variety of tools are planned that cover most of the items listed in Table 4-6 of this report. RCA is also studying the feasibility of a tool that will apply a tasking model to analyze the real-time behavior of a system described by a PDL model.

6.1.4 SDC Ada PDL (SDC, A Burroughs Co.)

The Ada-based PDL being developed by SDC supports MIL-STD-1815A Ada, and provides additional mechanisms for expressing supplemental design information embedded in Ada comments. This extra information addresses issues such as: completeness, requirements bracing, invariants, assertions, modifications, exception identification, and enforcement of a documentation discipline.

A suite of tools is being developed to support use of the PDL. The primary tool is a PDL processor that performs syntactic and semantic analysis of a PDL design. This is done by use of an ordered attribute grammar and tree-walk evaluation of a Diana tree that represents the Ada source language and annotations. The result of running the PDL processor on a PDL design is the detection of static semantic errors, the production of diagnostic messages, and the creation of attribute trees that can be used as input to other tools in the suite. The processor also performs standards adherence checking.

The other tools in the suite will include a pretty printer, cross-referencer, and module dependency and structure chart generator. Documentation generators and metric reports are also being considered, along with tools to support configuration management, managerial procedures, and other life cycle activities.

6.1.5 TAP (Reifer Consultants, Inc.)

The TAP Ada-based PDL and tool set is expected to be available in the near future. A VAX version of the tool set is currently undergoing testing and the target date for its release is March 1985. An IBM XT/AT version is also under development and is intended for release in May 1985. The tool set is being developed as a product. Documentation and training will be provided, and licensing agreements will include full support.

TAP supports a subset of the Ada language features, but provides several extensions including an innovative keyword feature that allows users to define their own addititonal constructs. Directives are used to ease potential man-machine interface problems and provide the user with options for both processing and format control.

The outputs provided by the tool set include: a cross-reference of all non-keywords, a module invocation tree, and multiple cross-references of marked symbols that cross package boundaries. The tools also provide pretty printing capabilities and enforcement of design standards. A novel feature is a box concept that can be used to highlight critical areas of a design.

6.2 Other Design Languages

A number of techniques are discussed here that do not fall under the classification of Ada-based. These include a general design support environment, a program verification language, a graphical design language, and a training research program.

6.2.1 AIM/SEM (ISDOS, Inc.)

The ISDOS project was begun at the University of Michigan in the early 1970's. Over the past fourteen years it has produced an extensive set of powerful tools aimed at designing, building, documenting, and maintaining large software systems. Unlike Ada-based PDLs, the ISDOS project does not focus on design activities, but intends to provide total life cycle support. It has already gained widespread recognition for the Problem Statement Language/Problem Statement Analyzer (PSL/PSA) tool that supports the specification of software requirements.

The ISDOS project is based on the premise that all knowledge pertinent to a system development project can be represented by entity relationship (ER) modelling. The ER model forms the schema of a very general data base for capturing information. Existing tools are able to extract data for the model by analysis of structured text or program code itself. Once data has been captured, a variety of analyses and reports can be generated. This general

model can be specialized to work with particular conventions and standards at each stage of the life cycle. The system data base is maintained for the entire life cycle and becomes the repository of all system information.

Recent advances have added capabilities designed to support the development of Ada software. In order to illustrate how ISDOS currently relates to Ada, it is best to trace the history of AIM/SEM, and thereby to explain its name. During the ISDOS project, a general methodology support tool was developed: the System Encyclopedia Manager (SEM). This general tool can be specialized for a particular method using the Information System Language Definition Manager or ISLDM. One such specialization has already been performed in the case of the Ada Integrated Methodology (AIM) developed by General Dynamics under the sponsorship of the U. S. Army. The resulting support environment for Ada development is known as AIM/SEM.

The PDL used in AIM/SEM is the GDADL Ada-based PDL developed by General Dynamics. This PDL is described in Section 4.2.5.

The ISDOS support environment has much to recommend it to the Ada world. It emphasizes automated support, provides tools for a wide slice of the life cycle, and has an extensive track record. It is logical to expect that ISDOS will provide full support for a particular PDL in the near future. Meanwhile, any project large enough to justify the effort needed to set up and maintain a comprehensive project data base could consider adapting the ISDOS tools and data base to run with the Ada-based PDL selected for the project.

6.2.2 ANNA (Stanford University)

ANNA extends the Ada language with facilities that permit formal specification of the intended behavior of Ada programs. It provides a means for augmenting Ada with precise machine processible extensions that allow the application of well established formal methods of specification and documentation techniques to Ada. As such, its primary use lies in the verification of Ada programs.

ANNA provides two kinds of formal comment: virual Ada text and annotations. Both take the form of legal Ada comments. The first is preceded by the virtual comment indicator (--:), and the second is preceded by the symbol --|).

The virtual Ada text is used to define concepts and compute values that are used in the annotations. It is stated using the syntax and semantics of Ada and must be legal in the context of the underlying Ada program.

One of the examples given in the ANNA documentaion is as follows:

```
-- ITEM and MAX are previously declared in the Ada text
package STACK is
--: function LENGTH return NATURAL;
procedure PUSH(X : in ITEM);
     --| where in STACK.LENGTH < MAX,
     --|     out (STACK.LENGTH = in STACK.LENGTH+1) ;
procedure POP (X : out ITEM) :
end STACK;

package body STACK is
    type TABLE is array (POSITIVE RANGE <>) of ITEM;
    SPACE : TABLE(1 .. MAX) ;
    INDEX : NATURAL range 0 .. MAX := 0 ;

--:  function LENGTH return NATURAL;
    --| where return INDEX;
--:  is separate;
          ...
end STACK;
```

Here the function used in the annotations is defined in virtual ada text.

Annotations may contain two types of variables: logical variables and program variables. they are used to express constraints on the values of program variables over their scope, where the scope of an annotation depends on its position in the underlying ada text and is determined according to Ada scope rules.

ANNA defines a formal syntax and semantics for expressing annotations that is similar to Ada. One of the major differences results from the provision of two additional quantifiers FOR ALL and EXIST. Several different kinds of annotations are provided that roughly correspond to the Ada language constructs. There are annotations of objects, subtypes and types, statements, and subprograms. The earlier example demonstrates the use of a subprogram annotation. Additionally, ANNA provides axiomatic annotations of packages, propagation annotations of exceptions, and context annotations.

Each kind of annotation describes a set of properties that must be satisfied by computations of the Ada text. If the computations do satisfy all the properties, the Ada text is deemed consistent with the annotations.

Automated verification of Ada programs will be based on a definition of ANNA that permits a series of transformations to be performed on the annotations. Following this approach, each annotation will be successively transformed into simpler annotations until they have all been reduced to assertions. The assertions can then be translated into Ada text that checks whether the assertion is satisfied by a program state. Tools are currently being developed to perform these functions.

In its present form ANNA is not a PDL, though its developers are investigating how it can be modified for use as a PDL. Meanwhile, ANNA is a valuable tool that can be employed in conjunction with any of the Ada-based PDLs discussed in this report.

6.2.3 CAEDE (Carleton University)

The Carleton Embedded System Design Environment (CAEDE) is an implementation of the design notation used in a design methodology discussed in "System Design with Ada" [8]. It is a graphical program design language intended to support production of Ada code. While it is an Ada-based PDL in name and fact, the use of graphs and symbols sets this PDL apart from the others in the survey. The motivation for CAEDE is also novel within the current circle of PDL offerings.

Ada was developed to support the development and maintenance of embedded systems, that is, systems that contain cooperating processes communicating asynchronously. Yet it is possible to express a design in an Ada-based PDL that shows poor performance during execution. Proper design of an embedded system requires an understanding of the behavior of the network of queues and processes that generally only becomes available as a result of compiling the Ada-based program.

CAEDE provides a high level graphical technique for designing the network. It uses a novel paradigm for developing software. Systems are designed using an interactive graphical editor. Properly annotated system designs can then be transformed into an engineering data base that consists of Prolog facts. The facts can be analyzed to write correct Ada language statements for packages, tasks, and their interfaces. Moreover, the facts can be given to a theorem prover to test whether real-time behavior faults such as deadlock are possible.

The developers hope that this PDL will mitigate the shortage of engineers who are trained to analyze and design such embedded systems.

6.2.4 Hazeltine Ada PDL (Hazeltine Corp.)

Information about this PDL was taken from the fifth chapter in a recent monograph by R. S. Freedman. It uses the full Ada syntax and it is directed at the same phase of the life cycle as Ada-based PDLs. However, it does not go beyond Ada in either of the two areas mentioned in Section 3.1. It does not establish design rules and guidelines or provide mechanisms for capturing supplemental design information. Therefore, this PDL is not classified as Ada-based and there is no fair basis for directly comparing this effort with the others.

On the other hand, the book describing this PDL could be adapted to work as tutorial or training material for any of the Ada-based PDLs. The material in Chapter 5 includes discussion of a data flow methodology for system development and presents a sizable case study illustrating the use of Ada as a PDL. The lack of any unique or proprietary PDL features allow the book's use to supplement the sketchy preliminary documentation that characterizes many Ada-based PDLs.

The bulk of the book provides instruction in the use of Ada to develop code. In comparison to other texts, this book is best suited to time-constrained training programs where an in-depth presentation of Ada is not possible. An instructor could teach the material in this book as the classroom portion of the course and assign one of the more comprehensive textbooks as additional reading.

6.2.5 York Ada PDL (University of York)

The University of York is conducting a detailed assessment of the suitability of Ada for use both as a PDL and as a language for specifying requirements, with a particular focus on the development of information systems. To date, there is no evidence that they are developing a particular Ada-based PDL.

The work has involved a re-examination of the roles of specification languages and program design languages. It gives several examples of how Ada can be used for these purposes, discusses useful extensions to Ada, and suggests desirable support aids.

One of the conclusions reached is that formal models can be used to record functional requirements during the development of various kinds of information systems. These different types of information systems are: transition-effecting systems, facility providing systems, and relation-maintaining systems. Ada subprograms, packages, and tasks correspond to these.

A further conclusion is that Ada provides adequate notations for the major PDL issues, particularly structural ones. The main deficiency of Ada is that the specification of interfaces does not identify effects. However, the provision of a notation for declarative assertions could eliminate this problem.

7.0 Conclusions

Much progress has been made since the previous survey two years ago. Many more Ada-based PDLs are available and these PDLs reflect a more consistent approach to the use of Ada for design purposes. While the PDLs differ in the precise language features they provide, the need for mechanisms for expressing postponed design decisions and information that supplements the PDL design statements is generally recognized. Additionally, these PDLs are starting to be fairly well supported by automated tools.

However, the work in this field is still continuing. Not only do new Ada-based PDLs continue to be developed, but many of the existing PDLs are expected to evolve. This evolution will occur in response to the experience gained through the practical use of the PDLs, and also as a result of further study into design issues.

Indeed, several trends that justify close observance are already evident. One is the potential use of Ada-based PDLs to promote reusability of software designs. Another is the increased ability to track requirements through to designs, and designs through to Ada code. An example in the second case is ANNA. Although ANNA is not itself a PDL, it can be used in conjunction with any PDL that follows Ada syntax and semantics. Use of ANNA facilitates some forms of formal verification of designs and also supports verifying that a piece of code does indeed implement a design. New paradigms for the interaction of people with the design document are also starting to appear. The best example of this is CAEDE.

The automated support aspect of Ada-based PDLs will change significantly over the next few years. Not only will there be more tools available, but the nature of these tools is likely to change. One advance, demonstrated by AIM/SEM, in automated support is the integration of PDL tools with automated support functions covering earlier life cycle phases.

This atmosphere of continual change presents difficulties for those trying to decide which PDL to adopt. The current set of Ada-based PDLs provides a wide spectrum of choice. While no one PDL is demonstrably superior to the others, there are some guidelines that a potential user can use to narrow the selection.

If an organization has a large investment in an existing non-Ada PDL, the evolutionary development of an Ada-based PDL is beneficial in the short run. If this is not the case, or if the organization is willing to expand the community of PDL users by an extensive training program, a PDL that uses the full Ada syntax with no variations is recommended. There seems to be no reason to doubt the acceptability of Ada syntax for a design language, although it might be wise to defer the use of certain Ada language features. The advantages of a unified syntax are significant. Moreover, design and coding experience become transferabale.

No such firm recommendation can be given in the case of the compilability of the PDL. Many developers argue that the use of a compiler forces an inappropriate attention to detail. If there are no tools available other than the compiler, then it is clearly desirable to have a compilable PDL. However, this is not the situation for all PDLs. Many Ada-based PDLs have pretty printers, cross-reference generators, and data dictionary tools that provide much of the documentation a compiler produces. The choice between a compilable PDL and a noncompilable but automated one cannot be decided on principle alone This difference should be the subject of future field tests.

Consideration of the types of mechanisms desired for postponing design decisions and expressing supplemental information also influence a PDL selection. As noted earlier in this report, the provision of these additional

PDL features is the area of most divergence among the PDLs. A few of the factors that influence these issues include the software application type, and the need to meet existing organization standards.

An organization may need an Ada-based PDL to develop software that is implemented in languages other than Ada. In this case, it is best to choose a PDL that provides guidance for translating the design text into these other languages.

Sometimes, an organization wishes to specify a PDL that will be used by contractors to develop software that must subsequently be supported by people other than the developers. Or the organization may need to purchase a software system developed using an Ada-based PDL. In these cases, the priority of selection factors may change. The most important consideration is now likely to concern the documentation aspects of the PDL, since documentation quality affects design reviews and the eventual modification and maintenence of the software. The documentation is a function of 1) the design unit templates; 2) the methodology and style guidelines; and 3) the effectiveness of automatic documentation tools. These PDL aspects are described elsewhere in the report. However, it is important to note that the documentation features described in Byron and WIS Ada PDL, to name just two, are independent of the PDL selection and can be added to any other PDL.

It is also wise to consider the future support that will be provided for a given PDL. Those that are part of a long term program, such as the WIS Ada PDL, are likely around for many years. This means that new tools will probably be developed and, where these are publicly available, this is a considerable advantage. The long term availability of the PDL is also relevant in the context of developing libraries of reusable designs.

However, if there are no PDLs that meet the requirements, an organization should consider tailoring an existing PDL or selecting different features from several PDLs to build a composite. One note of caution: if this approach is followed, it is important that the composite pdl is well documented. both the software developers and design reviewers need a clear and concise description of the syntax and semantics that constitute the PDL.

The recent progress in Ada-based PDLs is encouraging. On the whole, the design of these PDLs reflects contemporary software engineering practices, and the PDLs themselves are generally better documented than earlier efforts. Additionally, the common Ada ancestry has resulted in a high degree of similarity among the PDLs.

Although the degree of similarity among the PDLs is sufficient to allow the mixing and matching of PDL features, there are advantages to be gained by more standardization. In many respects, standardization of Ada-based PDLs would offer the same benefits as standardization of programming languages. Although several DoD agencies are in the process of developing standards and guidelines, these are independent efforts. This is unfortunate. The establishment of a community of software developers that use a common design language and the increased availability of automated support tools that will result from standardization would support attaining many of the benefits sought by the technical community.

In terms of future advances to the state-of-the-art in Ada-based PDLs, these will be the result of integrating PDLs with design techniques. This is necessary if there is to be full validation and verification of the information captured by a PDL, or full use of the PDL design text to guide the generation and testing of Ada code.

REFERENCES

[1] "Ada Programming Design Language Survey FINAL REPORT," Naval Avionics Center, Indianapolis, Indiana, October 1982.

[2] "A Guide," IEEE Ada as a PDL Working Group, (Draft), July 1984.

[3] "Ada DL Developers," Ada Letters, Volume IV, Issue 3, November/December 1984.

[4] "The Development of ADA Program Design Languages," Department of the Army, Headquarters United States Army, Communications and Electronics Command, Fort Monmouth, New Jersey, January 1984.

[5] "Military Standard - Ada Programming Language," ANSI/MIL-STD-1815A, United States Department of Defense, January 1983.

[6] Grady Booch, "Software Engineering with Ada," The Benjamin/Cummings Publishing Company, Inc., Menlo Park, California, 1983.

[7] Joint Logistics Commanders, "Defense System Software Development," Computer Software Management Subgroup, (Draft), August 1983.

[8] R. J. A. Buhr, "System Design with Ada," Prentice Hall, Englewood Cliffs, New Jersey, 1984.

APPENDIX A: Addresses of PDL Developers

This appendix provides the names and addresses of the organizations responsible for developing the PDLs discussed in this report. In those cases where the PDL is sponsored by a different organization to the one that developed it, the name of the sponsoring organization is given. The appendix also identifies an appropriate contact at each of the organizations given.

The information is presented in two different formats. The first part of the appendix lists the details for each PDL in the order dictated by an alphabetical sorting on PDL abbreviations. Whereas Part 2 presents the same details ordered on the name of the organization.

PART 1

PDL Identification: **ACK**

Organization: (Independent consultant)

Contact: Edward F. Hoover, III
505 Cypress Point, #171
Mountain View, CA 94303
Tel: 415-961-8396

PDL Identification: **ADAP**

Organization: SofTech, Inc.
3100 Presidential Drive
Fairborn, OH 45324

Contact: Glen Kersnick
Tel: 513-429-2771

PDL Identification: **ADA/SDP**

Organization: Mayda Software Engineering, IL Ltd.
PO Box 1389
Rehovot
Israel 76113

Contact: Zui Yami
Tel: (08) 473-480

PDL Identification: **AIM/SEM**

Organization: ISDOS Inc.
325 Eisenhower Parkway, Suite 103

PO Box 4179
Ann Arbor, MI 48106-4179

Contact: Jeff Baron
Tel: 313-663-6027

PDL Identification: **ANNA**

Organization: Stanford University
Computer Systems Laboratory
Department of Electrical Engineering & Computer Science
Stanford, CA 94305-2192

Contact: David C. Luckham
Tel: 415-497-2300

PDL Identification: **ATS Ada PDL**

Organization: Advanced Technology Systems
8027 Leesburg Pike
Vienna, VA 22180

Contact: William Morsch
Tel: 703-827-9519

PDL Identification: **BTO Ada PDL**

Organization: Bell Technical Operations
6365 East Tanque Verde
Suite 200
Tucson, AZ 85717

Contact. Mary Jane Stoughton
Tel: 602-721-0500

PDL Identification: **Byron**

Organization: Intermetrics, Inc.
733 Concord Avenue
Cambridge, MA 02138

Contact: Barbara Guyette
Tel: 617-661-1840

PDL Identification: **CAEDE**

Organization: Carleton University
Department of Computer and System Engineering
Ottawa
Ontario, K1S 5B6

Canada

Contact: Prof. Ray Buhr
Tel: 613-231-2645

PDL Identification: **CFG Ada PDL**

Organization: Caine, Farber, & Gordon, Inc.
750 E. Green Street
Pasadena, CA 91101

Contact: Kent Gordon
Tel: 818-449-3070

PDL Identification: **GDADL**

Organization: General Dynamics
Pomona Division
PO Box 2507
Pomona, CA 91769

Contact: Thomas S. Radi
Tel: 714-620-7511

PDL Identification: **Harris Ada PDL**

Organization: Harris Corporation
Government Information Systems Division
PO Box 98000
Melbourne, FL 32902

Contact: Cameron Donaldson
Harris GISD Software Operations MS 1/1545
505 John Rodes Blvd.
Mel ourne, FL 32902
Tel: 305-242-5181

PDL Identification: **Hazeltine Ada PDL**

Organization: Hazeltine Corporation
Cuba Hill Road
Greenlawn, NY 11740

Contact: Roy S. Freedman
Tel: 516-261-7000 Ext. 2509

PDL Identification: **IDL Ada PDL**

Organization: IDL Tech

18590 Centura Blvd., #202
Tarzana, CA 91356

Contact: Dr. Guy DeBalbine
Tel: 818-343-2083

PDL Identification: **JPDL**

Organization: U.S. Air Force Systems Command
Electronic Systems Division
TCS
Hanscom AFB, MA 01731

Contact: Robert G. Howe
Mitre Corporation
PO Box 208
M/S X010
Bedford, MA 01730
Tel: 617-271-2000

PDL Identification: **LTH Ada PDL**

Organization: LTH Systems, Inc.
776 Shrewsbury Avenue
Tinton Falls, NJ 07701

Contact: Bob Weissensee
Tel: 201-530-0990

PDL Identification: **PDL/Ada**

Organization: IBM Federal Systems Division
15 Firstfield Drive
Gaithersburg, MD 20875

Contact: Don O'Neill
Tel: 301-921-8913

PDL Identification: **RCA Ada PDL**

Organization: RCA
Government Systems Division
Moorestown, NJ 08057

Contact: R. M. Blasewitz
Tel: 609-541-7000

PDL Identification: **RN ADAP**

Organization: SofTech, Inc.

460 Totten Pond Road
Waltham, MA 02254

Contact: Chris Braun
Tel: 617-890-6900

PDL Identification: **SAIC Ada PDL**

Organization: SAIC Comsystems Division
2801 Camino Del Rio South
San Diego, CA 92138

Contact: Timothy Porter
Tel: 619-293-7500

PDL Identification: **Sanders Ada/PDL**

Organization: Sanders Associates, Inc.
Federal Systems Group
95 Canal Street
Nashua, NH 03061

Contact: Daryl R. Winters
Tel: 603-885-9225

PDL Identification: **SDC Ada PDL**

Organization: SDC, A Burroughs Co.
Research & Development
PO Box 517
Paoli, PN 19301

Contact: Terry Paton
Tel: 215-648-7200 Ext. 7268

PDL Identification: **TAP**

Organization: Reifer Consultants, Inc.
25550 Hawthorne Blvd.
Torrance, CA 90505

Contact: Don Reifer
Tel: 213-373-8728

PDL Identification: **TI Ada PDL**

Organization: Texas Instruments, Inc.
PO Box 801
Mail Station 8007
McKinney, TX 75069

Contact: Richard Conn
Tel: 214-952-2139

PDL Identification: **TRW Ada PDL**

Organization: TRW
MS R2/1134
One Space Park
Redondo Beach, CA 90278

Contact: Hal Hart
Tel: 213-535-5776

PDL Identification: **WIS Ada PDL**

Organization: WWMCCS Information System
U.S.A.F. Systems Command (Electronic Systems Div.)
Captain Percy Saunders
ESD/SCW-2E
Hanscom AFB, MA 01731

Contact: Al LaMontagne (Developer)
GTE Communication Systems
1777 North Kent Street
Arlington, VA 22209
Tel: 703-247-9282

PDL Identification: **XAda**

Organization: GTE Sylvania Systems Group
Western Division
100 Ferguson Drive
PO Box 7188
Mountain View, CA 94039

Contact: Dr. Sam Harbaugh (Developer)
Integrated Software Inc.
620 Iris Avenue
Sunnyvale, CA 94086
Tel: 408-773-1621

PDL Identification: **York Ada PDL**

Organization: University of York
Heslington
York, YO1 5DD
England

Contact: Professor I. C. Pyle
Tel: 44 + 0904 59861

PART 2

Organization: **Advanced Technology Systems**
8027 Leesburg Pike
Vienna, VA 22180

PDL Identification: ATS Ada PDL

Contact: William Morsch
Tel: 703-827-9519

Organization: **BTO Technical Operations**
6365 East Tanque Verde
Suite 200
Tucson, AZ 85717

PDL Identification: Bell Ada PDL

Contact: Mary Jane Stoughton
Tel: 602-721-0500

Organization: **Caine, Farber, & Gordon, Inc.**
750 E. Green Street
Pasadena, CA 91101

PDL Identification: CFG Ada PDL

Contact: Kent Gordon
Tel: 818-449-3070

Organization: **Carleton University**
Department of Computer and System Engineering
Ottawa
Ontario, K1S 5B6
Canada

PDL Identification: CAEDE

Contact: Ray Buhr
Tel: 613-231-2645

Organization: **General Dynamics**
Pomona Division
PO Box 2507
Pomona, CA 91769

PDL Identification: GDADL

Contact: Dr. Thomas S. Radi
Tel: 714-620-7511

Organization: **GTE Sylvania Systems Group**
Western Division
100 Ferguson Drive
PO Box 7188
Mountain View, CA 94039

PDL Identification: XAda

Contact: Dr. Sam Harbaugh (Developer)
Integrated Software Inc.
620 Iris Avenue
Sunnyvale, CA 94086
Tel: 408-773-1621

Oganization: **Harris Corporation**
Government Information Systems Division
PO Box 98000
Melbourne, FL 32902

PDL Identification: Harris Ada PDL

Contact: Cameron Donaldson
Harris GISD Software Operations
MS 1/1545
505 John Rodes Blvd.
Melbourne, FL 32902
Tel: 305-242-5181

Organization: **Hazeltine Corporation**
Cuba Hill Road
Greenlawn, NY 11740

PDL Identification: Hazeltine Ada PDL

Contact: Roy S. Freedman
Tel: 516-261-7000 Ext. 2509

Organization: **IBM Federal Systems Division**
15 Firstfield Drive
Gaithersburg, MD 20875

PDL Identification: PDL/Ada

Contact: Don O'Neill
Tel: 301-921-8913

Organization: **IDL Tech**
18590 Centura Blvd., #202
Tarzana, CA 91356

PDL Identification: IDL Ada PDL

Contact: Dr. Guy DeBalbine
Tel: 818-343-2083

Organization: **Intermetrics, Inc.**
733 Concord Avenue
Cambridge, MA 02138

PDL Identification: Byron

Contact: Barbara Guyette
Tel: 617-661-1840

Organization: **ISDOS, Inc.**
325 Eisenhower Parkway, Suite 103
PO Box 4179
Ann Arbor, MI 48106-4179

PDL Identification: AIM/SEM

Contact: Jeff Baron
Tel: 313-663-6027

Organization: **LTH Systems, Inc.**
727 Eastern Lane
Brickton, NJ 08725

PDL Identification: LTH PDL

Contact: Bob Weissensee
Tel: 201-530-0990

Organization: **Mayda Software Engineering, IL Ltd.**
PO Box 1389
Rehovot
Israel 76113

PDL Identification: ADA/SDP

Contact: Zui Yami
Tel: (08) 473-480

Organization: **Reifer Consultants, Inc.**
25550 Hawthorne Blvd.
Torrance, CA 90505

PDL Identification: TAP

Contact: Don Reifer
Tel: 213-373-8728

Organization: **RCA**
Government Systems Division
Moorestown, NJ 08057

PDL Identification: RCA Ada PDL

Contact: R. M. Blasewitz
Tel: 609-541-7000

Organization: **Sanders Associates, Inc.**
Federal Systems Group
95 Canal Street
Nashua, NH 03061

PDL Identification: Sanders Ada/PDL

Contact: Daryl R. Winters
Tel: 603-885-9225

Organization: **SAIC Comsystems Division**
2801 Camino Del Rio South
San Diego, CA 92138

PDL Identification: SAIC Ada PDL

Contact: Timothy Porter
Tel: 619-293-7500

Organization: **SDC, A Burroughs Co.**
Research & Development
PO Box 517
Paoli, PN 19301

PDL Identification: SDC Ada PDL

Contact: Terry Paton
Tel: 215-648-7200 Ext. 7268

Organization: **SofTech, Inc.**
3100 Presidential Drive
Fairborn, OH 45324

PDL Identification: ADAP

Contact: Glen Kersnick
Tel: 513-429-2771

Organization: **SofTech, Inc.**
460 Totten Pond Road
Waltham, MA 02254

PDL Identification: RN ADAP

Contact: Chris Braun
Tel: 617-890-6900

Organization: **Stanford University**
Computer Systems Laboratory
Department of Electrical Engineering & Computer Science
Stanford, CA 94305-2192

PDL Identification: ANNA

Contact: David C. Luckham
Tel: 415-497-2300

Organization: **Texas Instruments, Inc.**
PO Box 801
Mail Station 8007
McKinney, TX 75069

PDL Identification: TI Ada PDL

Contact: Richard Conn
Tel: 214-952-2139

Organization: **TRW**
MS R2/1134
One Space Park
Redondo Beach, CA 90278

PDL Identification: TRW Ada PDL

Contact: Hal Hart
Tel: 213-535-5776

Organization: **U.S. Air Force Systems Command**
Electronic Systems Division
TCS
Hanscom AFB, MA 01731

PDL Identification: JPDL

Contact: Robert G. Howe
Mitre Corporation
PO Box 208
M/S X010
Bedford, MA 01730
Tel: 617-271-2000

Organization: **University of York**

Heslington
York, YO1 5DD
England

PDL Identification: York Ada PDL

Contact: Professor I. C. Pyle
Tel: 44 + 0904 59861

Organization: **WWMCCS Information System**
U.S.A.F. Systems Command (Electronic Systems Div.)
Captain Percy Saunders
ESD/SCW-2E
Hanscom AFB, MA 01731

PDL Identification: WIS Ada PDL

Contact: Al LaMontagne (Developer)
GTE Communication Systems
1777 North Kent Street
Arlington, VA 22209
Tel: 703-247-9282

APPENDIX B: Titles of Documentation Received

This appendix identifies the PDL documentation that was received during the course of the survey. The information is ordered alphabetically on the PDL abbreviations.

PDL Identification: **ADAP**

i) "The Ada Program Design Language (ADAP) User's Guide," October 15, 1983, SofTech.

ii) "SYNtax Directed Interactive Editor (SYNDIE), SofTech.

PDL Identification: **ADA/SDP**

i) "ADA/SDP User Manual," Mayda Software Engineering, December 1984.

ii) Nancy Linden Yavne, "A Simple Approach to a Related Syntax for an Ada PDL," Mayda Software Engineering.

iii) Nancy May Linden, "CASPOMAT - an Automated Teller System," and tool input and output listings, Mayda Software Engineering, 1984.

PDL Identification: **AIM/SEM**

i) David E. McConnell, "Productivity Initiatives for Effective Lifetime Support of the Navy's AEGIS Weapon System Computer Programs," Naval Surface Weapons Center, Dahlgren, VA 22448, ISDOS Ref.# M0543-0.

ii) Edward J. Dudash and David E. McConnell, Excerpt from: "Computer Program Design Document for the Automated Data Collection Systems (ADCS) Computer Program," June 1, 1983, Naval Surface Weapons Center, Dahlgren, VA 22448, ISDOS Ref.# M0544-0.

iii) Elliot J. Chikofsky, "Application of an Information Systems Analysis and Development Tool to the Maintenance Effort," May 1983, ISDOS Project, University of Michigan, ISDOS Ref.# 83PSA-0444-1.

iv) Daniel Teichroew, Kyo Chul Kang and Vaclav Rajlich, "AIM/SEM: An Ada Support Environment," May 1984, ISETT 1984 Conference, ISDOS Ref.# M0652-0.

v) "Problem Statement Language (PSL) Language Reference Summary, July 1983, ISDOS, Inc., ISDOS Ref.# M0174-8.

PDL Identification: **ANNA**

i) David C. Luckham, Friedrich W. von Henke, Bernd Krieg-Brueckner and Olaf Owe, "ANNA": A Language for Annotating Ada Programs," Technical Report No. 84-261, Program Analysis and Verification Group Report No. 24, July 1984, Stanford University.

ii) David Luckham and Friedrich W. von Henke, "An Overview of ANNA: A Specification Language for Ada," Technical Report No. 84-265, Program Analysis and Verification Group Report No. 26, September 1984, Stanford University.

PDL Identification: **BTO Ada PDL**

i) "PDL Specifications," Bell Technical Operations, Tucson, Arizona.

PDL Identification: **Byron**

i) "The Byron User's Manual," Version 1.1, 1984, Intermetrics, Inc.

ii) "Byron Reference Manual," August 1983, Intermetrics, Inc.

iii) "IR-MA-335 AIE-Ada Compiler: Technical Description," July 19, 1984, Intermetrics, Inc.

iv) Barbara Liskov, "Modular Program Construction Using Abstractions," Computation Structures Group Memo 184, September 1979, MIT Laboratory for Computer Science.

PDL Identification: **CAEDE**

i) R.J.A. Buhr and G.M. Karam, "An Informal Overview of CAEDE: A Design Environment for Ada," Report SCE 84-18, Carleton University, Ottawa Canada, October 1984.

ii) Carol Hayes, "CAEDE User's Guide," Report SCE-84-24, Carleton University, Ottawa Canada, November 1984.

iii) R.J.A. Buhr, "An Ada-Inspired Graphical Design Notation for Manual and Computer-Aided Design of Modular, Concurrent Systems (with Examples)," Report SCE-84-1, Carleton University, Ottawa Canada, January 1984.

PDL Identification: **GDADL**

i) Thomas S. Radi, "Disciplined Software Design Approach - Executive Summary," July 5, 1984, Pomona Division, General Dynamics.

ii) Thomas S. Radi, "Ada/PDL," September 5, 1984, Pomona Division, General Dynamics.

iii) Thomas S. Radi, "General Dynamics Ada-Based Design Language (GDADL) User's Guide," General Dynamics.

PDL Identification: **Harris Ada PDL**

i) "Ada Process Description Language Guide," Revision 2.0, Harris Corporation, Government Communications System Division, Software Methodology Development Group, 1984.

ii) "Ada Process Description Language Guide," Revision 1.0, Harris Corporation, Government Communications System Division, Software Methodology Development Group, March 1982.

iii) Brian Davis, "Partition Generator," Version 1.0, Harris Corporation, May 1983.

iv) "NESP Tool User's Guide (PDLAF, DGP, and DOCLST)," Harris Corporation, April 1983.

v) "Diagram Graphics Program (DGP) User's Guide," Harris Corporation, April 1983.

vi) "Document List Program (DOCLST) User's Guide," Harris Corporation, April 1983.

vii) "XREF User's Guide," Harris Corporation, June 1983.

viii)"Example of Progressive Elaboration of PDL by Adding More Procedures within an Encapsulation Package," Harris Corporation.

ix) "Appendix C - Compilable Harris Ada PDL Units," and examples of PDL, Harris Corporation.

PDL Identification: **Hazeltine Ada PDL**

i) R. S. Freedman, "Programming Concepts with the Ada Language," Petrocelli Books, Inc., Princeton, New Jersey, 1982.

PDL Identification: **JPDL**

i) "JAMPS PDL Guide," Document Ref. 3285-2-210/3, October 1984, U.S. Air Force Systems Command, Electronic Systems Division.

PDL Identification: **LTH Ada PDL**

i) "An Interim Guideline for Ada Based Development and Product Design Documentation," Final, October 31, 1984, LTH Systems, Inc.

PDL Identification: **PDL/Ada**

i) Don O'Neill, "Software Engineering," IBM Federal Systems Division.

ii) Don O'Neill, "GPS Adaption of Modern Software Design," IBM Federal Systems Division.

iii) "WIS CUS Design Guidelines," IBM Federal Systems Division.

iv) "Process Design Language/Ada - (PDL/Ada) Reference Manual," April 14, 1983, IBM Federal Systems Division, Ada Coordinating Group.

PDL Identification: **RN ADAP**

i) "Regency Net Ada Program Design Language User's Guide," Document Ref. 3295-3, May 1984, SofTech, Inc.

PDL Identification: **SAIC Ada PDL**

i) Tim Porter and Brian H. Creighton III, "Ada Program Design Language (PDL) Guidelines," Version 1.1, Document No. SAI 10000-025-0002, SAIC, Comsystems Division, November 1984.

ii) Donald A. Heimburger, Marcia A. Metcalfe and Henry Kleine, "SAI-SDDL User Instruction Manual," February 24, 1982, Science Applications, Inc.

iii) "A-SDDL Software Design and Documentation Language," Science Applications, Inc.

iv) "SAI-SDDL Design Example," Science Applications, Inc.

v) "SAI-SDDL Simscript Documentation Example," Science Applications, Inc.

vi) "SAI-SDDL Error Handling Design Example," Science Applications, Inc.

PDL Identification: **Sanders Ada/PDL**

i) "Reference Manual for Sanders Ada/PDL Program Design Language," Draft, December 1983, Software Systems Engineering Federal Systems Group, Sanders Associates.

PDL Identification: **TI Ada PDL**

i) "TI Ada PDL Preliminary," Texas Instruments, Inc.

PDL Identification: **TRW Ada PDL**

i) "Ada PDL User's Manual," TRW Defense Systems Group, Redondo Beach, California, September 1984.

ii) Hal Hart, "Ada PDL Demonstration," TRW Defense Systems Group, December 1983.

iii) Hal Hart, "The Role of Ada PDL in TRW's Ada Preparation Strategy," TRW Defense Systems Group, June 1983.

PDL Identification: **WIS Ada PDL**

i) "WIS Ada PDL Standard," CDRL Sequence No. 403, Contract No. F19628-84-C-0032, September 17, 1984, Electronic Systems Division, Air Force Systems Command, Hanscom AFB, MA 01731.

PDL Identification: **XAda**

i) Sam Harbaugh, "XAda: An Executable Ada Design Language Methodology," in Ada Letters November 1984.

ii) "SLICE Requirements Specification," GTE Sylvania Systems Group.

PDL Identification: **York Ada PDL**

i) I. C. Pyle, "Using Ada for Specification of Requirements and Design," York Computer Science Report No. 63, York University, Great Britain.

END

FILMED

7-85

DTIC