N89-16298 519-61 167043 8P.

SOFTWARE ENGINEERING AND ADA* IN DESIGN

Don O'Neill

IBM FSD
March, 1986
WADAS

*Ada is a registered trademark of the U.S. Government, Ada Joint Program Office

About the Author

Don O'Neill has been with IBM's Federal System Division (FSD) for the past twenty-six years. He is presently the Ada Technical Assistant to the FSD Vice President for Technology. As Manager of Software Engineering for FSD (1977-1979), Mr. O'Neill was responsible for the origination of FSD software strategies and the preparation of the FSD software Engineering Practices. He received an IBM Outstanding Achievement Award for his contribution to this effort. Mr. O'Neill has been applying modern software engineering on production software development projects. He has recently been leading the activity to prepare FSD for Ada use on projects.

Mr. O'Neill is a member of the Executive Board of the IEEE Technical Committee on Software Engineering. In addition, he has been a Distinguished Visitor of the IEEE Society. Mr. O'Neill also serves as a member of the AIAA Software Systems Technical Committee. He received his BS degree in mathematics from Dickinson College in Carlisle, Pennsylvania.

PREFACE

Modern software engineering promises significant reductions in software costs and improvements in software quality. The Ada language is the focus for these software methodology and tool improvements. The community may have underestimated the preparation for Ada, including compiler development and education. More must be done. On the other hand, the community may have underestimated the benefits of Ada productivity and quality. Perhaps expectations should be raised.

Software Engineering and Ada for Design overviews the IBM FSD Software Factory approach, including the software engineering practices that guide the systematic design and development of software products and the management of the software process. The revised Ada design language adaptation is revealed. This four level design methodology is detailed -- including the purpose of each level, the management strategy that integrates the software design activity with program milestones, and the technical strategy that maps the Ada constructs to each level of design. A complete description of each design level is provided along with specific design language recording guidelines for each level.

Finally, some testimony is offered on education, tools, architecture, and metrics resulting from project use of the four level Ada design language adaptation.

Section 1

INTRODUCTION

Software may be throttling the industrial development of the United States. As the information society takes hold, the demands for software are increasing. Furthermore, public expectation is increasing too; people want software that provides the right answers on time, everytime, and does so in a user-friendly manner. Software is intended to provide for the harmonious cooperation among people and machines. People possess an infinite variety and machines do only what is instructed, notwithstanding the promise of artificial intelligence. As a result, the burden on software is substantial indeed and is increasing.

Recently, software engineering has provided for the systematic design and development of software products and the management of the software process. The result should be quality software products obtained through design, sustained through development, and monitored through technical reviews. We have always known that good projects are ones with few errors at the end. We now know that good projects are also ones with few errors at the beginning. What may be needed now is a refinement of these methods, especially in the requirements and specification areas, their broad application, and preparation of adequate tools that re-enforce and enforce their use while assisting in productivity gains.

Section 2

SOFTWARE ENGINEERING FACTORY

Software engineering provides for the systematic design and development of software products and the management of the software process. Software

C-3

engineering may be viewed in the form of a state machine composed of inputs, transitions, outputs, and retained data (Figure 2-1).

The inputs to the process include the qualified people, labor saving tools, and practical technology needed to apply modern design, development, and management practices in the production of usable and reusable software products of sufficiently high quality to ensure life cycle benefits and confident customer ownership. People today are required to be highly qualified and equipped with specialized training in both software technology and applications. Testimony from early Ada users indicates that the training needs may be substantial. Harlan Mills observed that as we shape our tools, our tools may later shape us. Tools represent an institutionalized expert system, knowledge base of software methodology and style. Tool investments often lag behind their need. Practical technology requires the application of basic principles from advanced technologies repackaged into intuitive approaches and simplified for use by industry practitioners and acceptance by customers. Technology must employ an understandable conceptual model to assist the transition from user need to usable product.

The transitions of the software engineering state machine are governed by the software engineering practices for design, development, and management, applying across the full life cycle. Software design includes methods for producing and verifying modular designs and structured programs. Designs are recorded using a design language based on Ada, including both procedural designs and data designs. Advanced design ensures semantic correspondence of specifications through data dictionaries. Systematic design provides for functional allocation and decomposition of procedures and data. Systematic programming includes the elaboration of program designs using stepwise refinement, program design language, and correctness techniques. Taxonomy includes a proper program with a single entry and single exit, a prime program composed of zero or one predicate constructs, inner syntax of data refinement and operations and tests. The Ada based design language used to record designs assists the reasoning of the designer and his communication with others in getting the design right, knowing it, and convincing others. Software development includes the methodology for the early implementation and integration of detailed designs into product increments represented as source code libraries, configuration controlled through library hierarchies. In addition to incremental releases, the concepts of rapid prototyping, software first, and component reuse are being refined for routine use on projects in the future. Software management assures the effective application of qualified people within a predictable process to originate a quality product that satisfies performance requirements on schedule within cost. The use of Software Development Plans and technical reviews ensure an accurate view of status.

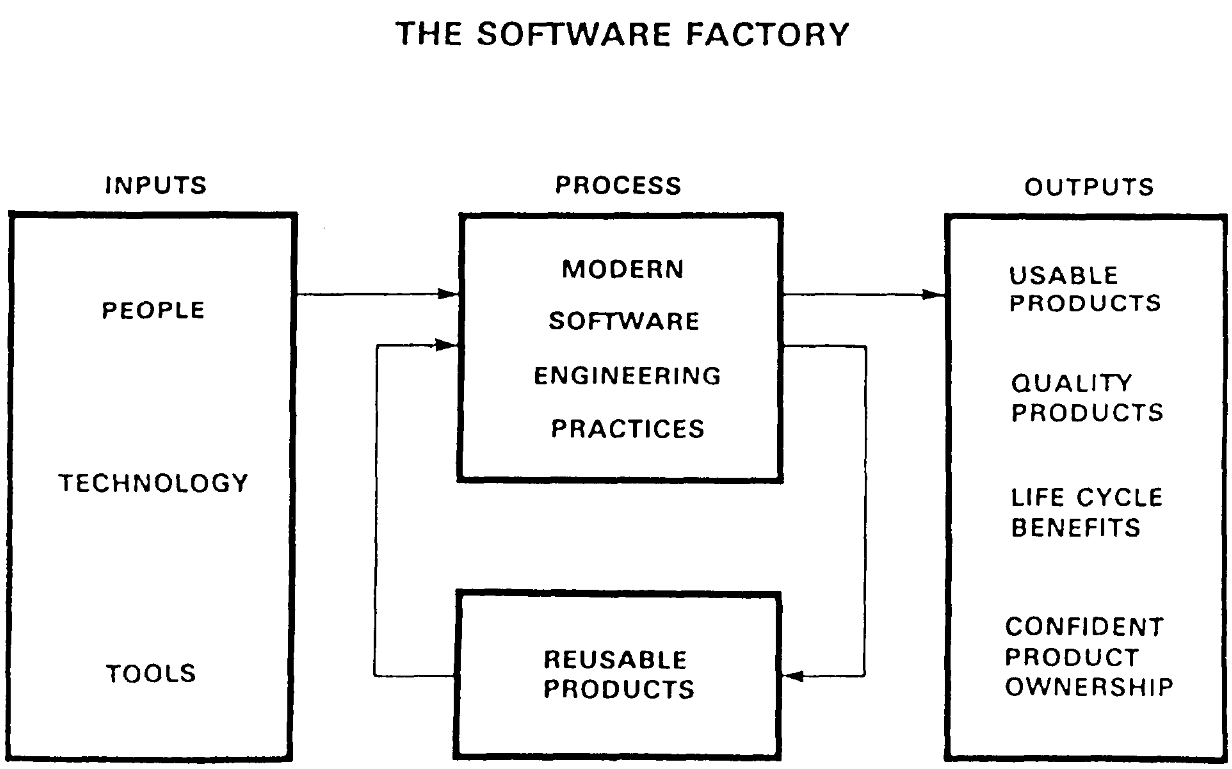

Figure 2-1. Software Factory

The outputs of the process include usable products of high quality that may be reusable capable of assuring confident customer ownership. Usable products are those that operate harmoniously within the user organization. They are adaptable to new requirements and feature userfriendly interfaces. The reward for this may be friendly users. Quality products are those that have few errors at the end. These are the same ones that have few errors at the beginning. A reusable product is one that continues to meet changing requirements through product enhancements. Futhermore, reusable products are transportable to other systems for similar uses. The Ada language promises to provide for software reusability. Using artificial intelligence, a components library of specifications can be interrogated for software components needed for new applications. As the industry becomes skillful and expert at matching existing products with new needs, the software factory may become a reality.

Section 3

SYSTEMATIC USE OF ADA AS A DESIGN LANGUAGE

Flirting with Ada? Careful. She is more than a programming language but less than a compiler for many. In Megatrends, John Naisbitt points out that trends are like horses. If you want to ride them, it pays to go in the same direction the horse is already traveling. He also points out that fads originate at the top, tend to peak, and then fade out. On the other hand, trends are bottom up, possess broader support, and persist.

The use of Ada as a programming language may correspond to Naisbitt's characterization of a fad, top down, perhaps explaining its sluggish beginning. For Ada the programming language, this is the awkward period between promise and delivery. On other hand, the use of Ada as a design language may be a trend, arising from the bottom as a popular choice. It is happening today.

A design language may be used for a number of reasons. It provides the facility to record design decisions. Once recorded, these design decisions can be shared with others forming the communication baseline among system engineers, software engineers, and integration and test engineers. It provides the basis for the designer to be more convincing in the defense of his design. It provides others with a clear reference point to focus their criticisms. The result is a better design. The use of Ada as a design language encourages good software engineering while at the same time permitting the design to obtain rigor in syntax and semantics through the use of Ada compiler product tools. Ada as a design language provides a platform for systematically accomplishing rapid prototyping through use of the emerging software design and product itself. In ways yet to unfold, Ada design language may also be a useful basis for assisting the access of reusable components. To be able to support these various uses systematically, Ada as a design language needs to be integrated into a software engineering methodology.

3.1 Four Level Design

An Ada based software design methodology has been adapted from the software engineering practices discussed in advanced design, systematic design, and systematic programming. This adaptation features four levels of design supported by a management strategy and a technical strategy. The management strategy maps the first two levels of design to the specification process and its review and the last two levels of design to the detailed design process and its review. The technical strategy purposefully and rigorously utilizes the expressive power of Ada at each level of design by mapping particular Ada constructs for use at each level.

The purpose of each level of design (Figure 3-1) considers the expected audience hierarchy within a project, ranging from readers to writers and including programmers, engineers, and managers. Early design levels must be intuitively understandable by all members of the audience and cannot depend on everyone being fully Ada literate. To support this need, Level 1 design is intended as the user contract. The user should be thought of as other software products that might utilize or interface with the software being designed as opposed to the end user of the system. Level 2 design portrays the design parts and their relationships both data interfacing and tasking. Level 3 design elaborates a detailed functional design that is independent of the target operating system and instruction set architecture. Finally, Level 4 designs are detailed designs that are fully targeted to the operating system and instruction set architecture, ready for implementation considering, efficiency and capacity constraints.

FOUR LEVEL DESIGN

METHODOLOGY THAT RIGOROUSLY UTILIZES THE EXPRESSIVE POWER OF ADA PDL AT EACH LEVEL

LEVEL 1 USER CONTRACT

LEVEL 2 DESIGN PARTS AND RELATIONSHIP

LEVEL 3 DETAILED FUNCTIONAL DESIGNS INDEPENDENT OF TARGET
- OPERATING SYSTEM
- INSTRUCTION SET ARCHITECTURE

LEVEL 4 DETAILED DESIGNS FULLY TARGETED READY FOR IMPLEMENTATION

Figure 3-1. Four Level Design

3.2 Management Strategy

The management strategy for the four level design approach (Figure 3-2) maps levels 1 and 2 to the specification review milestone and levels 3 and 4 to the design review milestone. The specification review milestone equates to the Preliminary Design Review (PDR), the design review milestone equates to the Critical Design Review (CDR). In the MILSTD 2167 process level 1 and 2 designs are included in the Software Top Level Design Document; level 3 and 4 designs are included in the Software Detailed Design Document.

Beginning with Level 1, the specification is input to the software design process. A Level 1 design is produced and recorded in the form of an Ada Package Specification. The Level 1 design and

SOFTWARE ENGINEERING AND ADA IN DESIGN

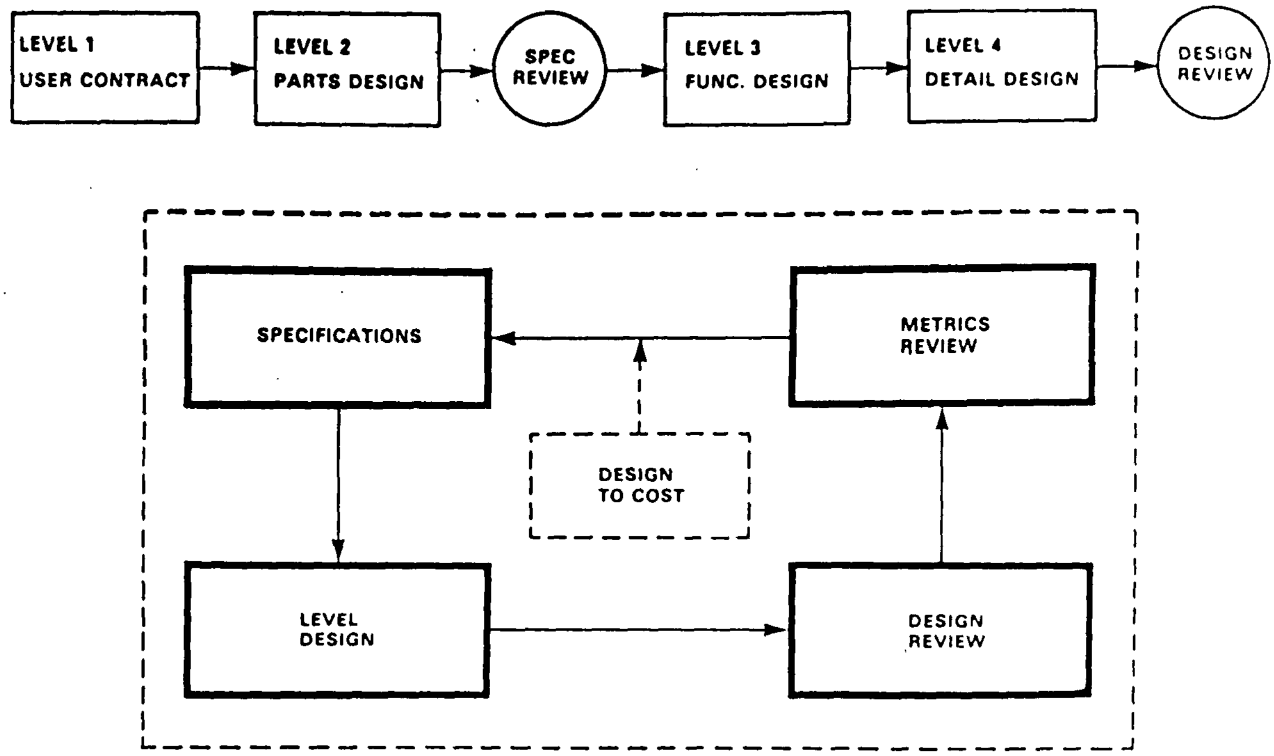

Figure 3-2. Management Strategy

any reuse candidates are subjected to a design review. The design review may be conducted electronically, or it may be conducted through a meeting of team members. Participants are highly trained experts committed to reviewing the design for completeness, correctness, usability, performance, and overall user satisfaction. Each reviewer must be personally satisfied with every aspect before the design review is concluded. The application of modern software engineering practices and their enforcement through a unanimous concensus of these highly trained experts is expected to provide a powerful impetus to dramatically improved product quality.

Once the design is satisfactory, the metrics associated with the software engineering process and the software product are reviewed and expectations revised. For the software engineering process the metrics include productivity and quality expectations. For the software product these metrics include complexity measures, reliability, and computer resource loading. These metrics are reviewed for compliance with budgets, perhaps necessitating adjustments in the design in an effort to achieve compliance. The specification itself may need to be reassessed and partitioned into essential requirements and desirable features. Certain desirable features may need to be eliminated or reduced in order to comply with management budgets.

At Level 2, the design for each component part identified in level 1 is recorded as an Ada package specification and its body. The Level 2 Ada package specification and body are evaluated for reuse candidates, continuing the systematic exploitation of reusability. The design review is conducted, as in Level 1. Metrics data is acquired and analyzed for Level 2 with the design to cost procedure followed if necessary. The Software Top Level Design Document is then subjected to the Preliminary Design Review (PDR). Throughout this process, systems engineers and software engineers work in a dependable relationship in shaping and fitting designs to meet user needs.

At levels 3 and 4, the design for each identified subunit is recorded as an Ada procedure with its accompanying intended function commentary. Procedure CALL semantics and intended function commentary are evaluated for reuse candidates, again continuing the systematic exploitation of reusability. Design reviews are conducted. Metrics data is analyzed. The design to cost procedure continues to operate but with diminished flexibility since the specification has been baselined at PDR.

3.3 Technical Strategy

The Technical Strategy (Figure 3-3) governs the mapping of Ada constructs to each level. This mapping is intended to follow the architectural line of the language. Futhermore, the construct mapping by level provides a natural partitioning suitable for educating both readers and writers a little at a time.

Ada constructs are mapped to each level for outer and inner syntax. Outer syntax includes organizing units and control structures, both sequential and asynchronous. The inner syntax provides the format for expressing data and the operations and tests on the data. The constructs are assigned to each level with the objective of satisfying the purpose of that level. Once assigned to a level, a construct is permitted to be used in subsequent levels.

The Ada product form for Level 1 is the package specification used to express the user contract. This calls for an organizing outer syntax along with inner syntax commentary. Level 1 is limited to a few self evident constructs needed to accomplish its purpose. Those constructs are listed in Figure 3-4. They can be conveniently organized into a package specification template used to govern the style of the design recording. Other design recording guidelines may be set forth, including naming convention, commentary for intended functions, and key words in structured commentary useful in encouraging the use of the state machine model.

The product form for Level 2 is the package specification and package body used to express the design parts and their relationship. This calls for an outer syntax of structuring and tasking constructs. The inner syntax may be expressed as Ada abstractions, including abstract data types. Procedural elaborations are not carried out in Level 2 but instead are permitted to appear as procedure calls. The additional Level 2 constructs are shown in Figure 3-4. Those too can be conveniently organized into package specification and package body templates used to govern the style of the design recording. Design recording guidelines of Level 1 may be expanded to include the abstract data types permissible.

The product form for Level 3 is the procedure elaboration used to express a detailed functional design. This calls for a full complement of outer syntax function expressions and inner syntax data refinement, including predefined data types and Ada primitives. The additional Level 3 constructs are shown in Figure 3-4. Here too, procedures and task templates are used to guide the style of the design recording. Additional recording guidelines may be stated. Furthermore, to control the quantity of the Ada PDL being produced, Level 3 may be limited to the elaboration of only those procedures present in Level 2 as procedure calls.

The product form for Level 4 is the procedure elaboration, as well as function elaborations used to express a fully targeted, detailed design. Full MIL-STD 1815A is available at Level 4 (see Figure 3-4).

Section 4

CONCLUSION

Software Engineering and Ada in Design is but an early milestone report on the systematic use of Ada as a design language. From this experience, it is clear that the preparation for the use of Ada has been underestimated in several areas, including Ada compiler acquisition, tool integration, and education.

The Ada compiler acquisition difficulties in industry are well known. The need for Ada products during the design activity has received less attention. It is nice to have an Ada front-end product for semantic and syntax analysis during levels 1

TECHNICAL STRATEGY

	PURPOSE	OUTER SYNTAX	INNER SYNTAX FUNCTION	INNER SYNTAX DATA
LEVEL 1	USER CONTRACT	ORGANIZING	COMMENTARY	COMMENTARY
LEVEL 2	DESIGN PARTS AND RELATIONSHIPS	STRUCTURING, TASKING	PROCEDURE CALLS	ABSTRACT DATA TYPES
LEVEL 3	DETAILED FUNCTIONAL DESIGNS INDEPENDENT OF TARGET	—	EXPRESSIONS	PREDEFINED DATA TYPES, ADA PRIMITIVES
LEVEL 4	DETAILED DESIGNS FULLY TARGETED	—	REFINEMENT	REFINEMENT

Figure 3-3. Technical Strategy

	PURPOSE	OUTER SYNTAX	INNER-SYNTAX: FUNCTION	INNER-SYNTAX: DATA
LEVEL 1	USER CONTRACT	PACKAGE SPECIFICATION PROCEDURE SPEC TASK SPEC WITH, USE	COMMENTARY	COMMENTARY ABSTRACT DATA TYPES

LEVEL 2	DESIGN PARTS AND RELATIONSHIPS	PACKAGE BODY
		IS SEPARATE
		BEGIN
		IF THEN
		CASE
		LOOP (WHILE FOR, EXIT WHEN)
		FUNCTION
		ACCEPT, DO
		SELECT
			PROCEDURAL CALLS
			TASK ENTRY CALLS
				GENERIC INSTANTIATIONS OF ABSTRACT DATA STRUCTURES
				PRIVATE DATA TYPES
				DERIVED DATA TYPES
				TASK TYPES

LEVEL 3	DETAILED FUNCTIONAL DESIGNS INDEPENDENT OF TARGET OPERATING SYSTEM AND INSTRUCTION SET ARCHITECTURE	ELSIF WHEN	:=,+ ,-,*,**,/ REM, MOD OR, AND, XOR, NOT RANGE, ABS =, , =, , =, /= TERMINATE DELAY EXCEPTION, RAISE	ADA DATA TYPES RECORD ARRAY RANGE ACCESS TYPE CONSTANT SUBTYPE
LEVEL 4	DETAILED DESIGNS FULLY TARGETED READY FOR IMPLEMENTATION		PRAGMA ABORT	DELTA, DIGITS FOR, USE, AT

ADA CONSTRUCTS

Figure 3-4. Ada Constructs

and 2. It is a necessity to have this tool available and ready for use during levels 3 and 4. Without it, the reenforcement of Ada education through the design activity is lost. Furthermore, the error discovery opportunity is postponed to downstream. The design inspection accompanying each design level needs the output of the Ada front-end. Where rapid prototyping is intended, the Ada compiler itself is needed to permit code generation and execution.

For early Ada projects, the education of the project team may need to be integrated with the design activity. One approach to this is to train people in one design level at a time, followed by the performance of the design activity and its review. In this way, the training schedule can be distributed throughout the performance period, the training for each level can be refined based on the results of the preceding design review, and project people new to Ada can progress through the experience sharing problems and obtaining assistance within the team. In the four level design approach, Level 1 represents only four constructs, all contained in a template. As a result, there is an early success for the new Ada PDL designer. Level 2 adds more constructs and is again guided by templates, assisting success. Level 3, however, represents the first time the Ada PDL designer must operate substantially on his own with a large number of Ada constructs. At Level 3, design reviews may result in a substantial reworking of the design. By Level 4, the Ada experience begins to pay off, and Ada PDL designers are completing their designs with confidence.

In formulating architectures for Ada software designs, new thinking may be needed. Important benefits are possible in Ada through modern software engineering. To obtain these benefits, software designs must make the transition from designs that simulate data flow to designs that encapsulate data in ways natural to the application providing only as much visibility as necessary and as much information hiding as possible. Furthermore, to obtain these benefits, the Ada tasking model needs to be exploited at appropriate levels in the design. The interface with commercial software products needs to be accommodated in a way that retains the cost benefits of these products, but does not dominate the software architecture. More work is needed in uniform design morphologies for Ada to provide useful Ada architecture models for early users, as well as the framework for exploiting reusable components by all users.

Very little is known about Ada metrics. As a result, there are many questions about the size of Ada programs and designs, Ada productivity, Ada quality, and Ada performance. The early experience with Ada PDL seems to show that a low ratio may exist between Ada source lines and Ada design lines. It may be 2:1 or 3:1. Where Ada is both the target language and the design language, the Ada PDL is part of the product. In this case, insight about the ratio may assist the allocation of effort and schedule between the design and code activities. The recent experience showed that the combined Level 1 and 2 ratio was about 25:1, Level 1-3 about 10:1, and Level 1-4 less than 5:1. Not enough is known to use these results as management budgets.

The revised IBM FSD four level Ada PDL methodology has demonstrated some important benefits in recent use (Figure 4-1). Expanding the audience of

BENEFITS: FOUR LEVEL ADA PDL METHODOLOGY

- **AUDIENCE** — BOTH TECHNICAL AND NON-TECHNICAL
- **PRODUCTIVITY** — TEMPLATES AT LEVEL AND CONSTRUCT
- **QUALITY** — MINIMUM CYCLOMATIC COMPLEXITY
- **PERFORMANCE** — FOCUS ON TASKING AT LEVEL 2
- **PORTABILITY** — FULLY TARGET INDEPENDENT LEVEL 3
- **REUSABILITY** — LEVEL FORMAT PERMITS EFFECTIVE ACCESS FROM COMPONENTS LIBRARY
- **ADA TRAINING** — LEARNING AND USING ADA, A LITTLE AT A TIME, IS AN EFFECTIVE APPROACH TO ADA TRAINING, ALONG ARCHITECTURE LINE
- **MAINTAINABILITY** — FOUR LEVELS PROVIDE A STAGED, LAYERED INTRODUCTION TO DESIGN AND IMPLEMENTATION DETAILS
- **PREDICTABILITY** — MEETING COST AND SCHEDULE AS ASSISTED BY DESIGN TO COST FEATURE OF MANAGEMENT APPROACH

Figure 4-1. Benefits: Four Level Ada PDL Methodology

design reviewers from technical to non-technical permits useful and needed user input to the completion of the specification and to early design decisions. This is made possible by a training program, patterned after the four levels, that teaches Ada a little at a time along the architectural line of the language. Furthermore, the templates that govern the product style at each level provide a crutch for the early Ada user both reader and writer, a boost to productivity, and the assurance of uniformity in design style. Productivity may be given a more substantial boost when reuse of existing Ada components can be obtained. The Level 1 template format may assist this component reusability by providing the semantics needed to access a components library. Managing and meeting cost and schedule budgets is assisted by the systematic use of the design to cost feature embedded in each design level. Once completed, the four levels of Ada PDL provide the layered introduction to design details needed by the maintainer to learn design details as needed and to originate any required product adaptations with confidence. Designs produced with the four level Ada PDL methodology tend to be the simplified designs that result from modern software engineering. At the same time these designs consider performance requirements and meeting real time deadlines through the tasking focus at Level 2 and through the metrics at every level. Finally the methodology supports portability through the Level 3 target independence of operating system and instruction set architecture.

Although true that the community has underestimated the preparation for Ada, this preparation has been started and is underway. It may also be true that the community has underestimated the benefits of Ada which are substantial and are still being discovered.

BIBLIOGRAPHY

1. "Reference Manual for the Ada Programming Language (MIL-STD-1815A)," Department of Defense, 17 February 1983.
2. "Methodman II," Institute for Defense Analysis (IDA), Memorandum Report M-11, November 1984.
3. "Survey of Ada™ - Based PDL's," Naval Avionics Center Technical Publication, TP-598, January 1985.
4. Naisbitt, J., "Megatrends: Ten New Directions Transforming Our Lives," Warner Books, Inc., 1982.
5. O'Neill, D., "Software Engineering Program," IBM Systems Journal, December 1980, Vol. 19, No. 4.
6. O'Neill, D., "An Integration Engineering Perspective," The Journal of Systems and Software, 3, 77-83 (1983).
7. O'Neill, D., "At IBM - A Strategy for Software Management," Information Systems News, February 1981.
8. "Ada as a Design Language," IEEE Computer Society Working Group (P. 990), Draft 1985.
9. O'Neill, D., "An Overview of Gloval Positioning System Software Design," Software Engineering Exchange, October 1980, Vol. 3, No. 1.

Key Words

Software Factory

Four Level Design

Ada based design

Management Strategy

Technical Strategy

Ada Constructs