# Experiences in Preparing a DoD-STD-2167A Software Design Document for an Ada Project

Charles A. Meyer, Sharon C. Lindholm, and Jack L. Jensen

Betac Corporation
Southwest Engineering Division
San Antonio, Texas

## ABSTRACT

This paper describes our experiences in using OOD and Ada in developing a Software Design Document (SDD) in accordance with DoD-STD-2167A. The paper first provides an overview of the SDD requirements stated in 2167A. It next describes the initial objectives and assumptions under which we began the project, then discusses the problems we encountered while trying to achieve our objectives and satisfy 2167A requirements. Three different categories of problems are described: those dealing with precise definitions of terms used in 2167A; those dealing with 2167A document format requirements; and, those dealing with satisfying customer expectations. The paper then describes the specific lessons we learned during this project, and finishes with some overall conclusions.

## 1. INTRODUCTION: SDD REQUIREMENTS

The requirements for the content, format, and organization of the SDD are stated in 2167A's Data Item Description (DID) DI-MCCR-80012A [80012A]. It specifies that a separate SDD is to be prepared for each Computer Software Configuration Item (CSCI) in the system. A brief summary of the major sections of the SDD and the tailoring to the SDD specified by our contract follows.

Sections 1 and 2 of the SDD are straightforward, requiring a system overview and project references, respectively.

Section 3 contains the preliminary design of the system. This includes an overview of the CSCI, with the stated purpose of each external interface. Section 3 also requires a description of system states and modes, and a description of memory and processing time allocation.

For the preliminary design of each Computer Software Component (CSC) and sub-level CSC, the following information is required:

- Requirements allocated to the CSC/sub-level CSCs
- Description of CSC/sub-level CSC preliminary design in terms of execution control and data flow, along with an identification of relationships between sub-level CSCs and CSCI internal interfaces.
- Derived design requirements for each CSC/sub-level CSC, and any design constraints imposed on or by the CSC/sub-level CSC.

Section 4 contains the detailed design for each CSC. This includes:

- A description of each of the Computer Software Units (CSUs) of a CSC, including a description of the relationships between CSUs in terms of execution control and data flow.
- The purpose of each CSU, its design specification/constraints, and the design of the CSU. This detailed design information includes identifying/describing:
  - input/output data elements
  - local data elements and data structures
  - interrupts and signals
  - algorithms
  - error handling
  - data conversion operations
  - other elements used by the CSU
  - logic flow, including description of the conditions under which the CSU is initiated
  - limitations or unusual features that restrict the performance of the CSU

Sections 5 and 6 describe the data requirements for the system. Section 7 calls out the requirements traceability between system-related documents, while Section 8 is reserved for miscellaneous notes.

For our project, the DID was tailored (by the contract) in several significant regards:

- Several major sections of the SDD were tailored out. This was because the information required for these sections was to be provided in alternate formats.
  - Section 5, *CSCI Data*, was delivered as a DoD-STD-1703 Data Dictionary [1703].
  - Section 6, *CSCI Data Files*, was delivered as a MIL-STD-7935 Data Base Specification [7935].
  - Section 7, *Requirements Traceability*, was formatted in accordance with a requirements traceability matrix specified by DI-MCCR-80025 from DoD-STD-2167 [80025].
- The design was only to be taken to "intermediate" design (but to avoid confusion, the remainder of this paper will refer to detailed design when discussing our activities). The means of defining the level of detail desired was established by specifying specific SDD paragraphs that were to be either excluded, or provided at contractor discretion. These paragraphs were primarily from Section 4, *Detailed Design*.

## 2. ASSUMPTIONS AND OBJECTIVES

We began this project with several explicit and implicit assumptions and objectives. Ultimately, it was these objectives and assumptions that drove the overall format and content of our SDD. We believe that some of these assumptions are still valid while others have proved to be remiss and lacking in foundation.

### 2.1 Assumptions

Many of our decisions concerning the organization and content of our SDD were based on the following assumptions.

a. During preliminary design, functional requirements from higher-order documents should be mapped to physical programming constructs.

b. Ada physical constructs can be mapped directly to 2167A logical concepts.

c. The DID requirements of 2167A are meant to be a set of guidelines that may be interpreted with respect to the project being addressed. 2167A DID requirements are intentionally broad in scope, and meant to be interpreted liberally as opposed to literally.

Also, we considered the goals of preliminary and detailed design to be as follows. For preliminary design:

- Mapping functional requirements to software concepts
- Defining an overall system architecture
- Defining the interactions between top-level components
- Refining the characteristics of the external interface's design.

For detailed design:

- Decomposing higher-level design components to implementable units
- Defining detailed characteristics of data structures
- Ensuring the design reflects the intended final structure of the code.

We assumed that meeting these goals was the critical issue. We believed the SDD was nothing other than a vehicle for expressing the design. As such, we did not treat the SDD as a "bible," but rather as a guide.

### 2.2 Objectives

The following were the primary management objectives for performing design activities:

a. To design a relatively large and highly complex interactive software project, integrating the Ada design into SDD guidelines.

b. To design the project in two distinct stages - preliminary design and detailed design

c. To adhere to tradition waterfall life cycle design concepts by developing a stable preliminary design that would serve as the baseline for detailed design.

## 3. INTERPRETING AND APPLYING THE SDD DID

### 3.1 Selecting Working Definitions for Key 2167A Terms

Several terms became of utmost importance in documenting our design. These start with the very basic terminology introduced by 2167A, namely CSCs and CSUs. The following are the definitions as presented in 2167A [2167A].

- *Computer Software Component (CSC):* "A distinct part of a computer software configuration item. CSCs may be further decomposed into other CSCs and CSUs."
- *Computer Software Unit (CSU):* "An element specified in the design of a CSC that is separately testable."

We established working definitions for these terms based on viewing CSCs as Ada library units (the visible packages and subprograms in an Ada implementation) and CSUs as subprograms and tasks. The following paragraphs summarize the rationale involved in deciding upon these implementations.

#### 3.1.1 Interpreting the Term "CSC"

Initially, we believed there were several possible working definitions of a CSC to choose from. They could be

- logical entities that perform specific functions unique to the application (as in [ELL89])
- logical entities that provide system-wide services (e.g., User Interface CSC, Data Base CSC, Utilities CSC, Executive Control CSC, etc.)
- Ada library units (as in [MEA89]).

Note that these definitions are not mutually exclusive.

The definition of CSCs proved to be the most difficult to address. Acting in accordance with assumptions (a) and (b) above, we tried valiantly to tie CSCs to a physical Ada implementation. The following paragraphs describe the decision-making process which we used to develop working definitions for CSCs.

##### 3.1.1.1 Interpreting CSCs as Library Units

The concept of a CSC fits neatly with the concept of an Ada library unit. This fulfills the criteria that a CSC is composed of "other things". In other words, a CSC is not a fundamental building block, but rather a collection of related building blocks. By definition, a CSC is composed of CSUs or other CSCs. By definition, a package is composed of subprograms, tasks, or other packages; likewise, subprogram library units can be further decomposed. These definitions parallel one another so closely the association is natural: Ada packages should implement CSCs.

However, this definition introduces complications when scrutinized against the SDD DID. Per the DID's requirements, CSCs must be described in terms of **execution control** and **data flow**. We never did figure out a good way to describe this concept for Ada packages, since execution control and data flow occurs between subprograms/tasks, not between packages. We could show "withing" dependencies, but in our minds, this didn't truly address execution control and data flow. Moreover, since we were in the preliminary design phase, this detailed design information was not available. Yet this data was required in the Preliminary Design section of the SDD. This directly contradicted our objective to create a baselined preliminary design document and to keep the design phases distinct. As a result, we found ourselves developing preliminary and detailed design information in parallel.

##### 3.1.1.2 Interpreting CSCs as Ada Subprograms

The rationale for implementing CSCs as Ada subprograms is as follows. Upon initial investigation, it would appear that 2167A makes a case for implementing CSCs as subprograms. Both the definitional requirements and the Section 3 requirements can be satisfied easily and intuitively by defining CSCs as subprograms. They can be composed of nested subprograms (CSUs) or they can be depend on other external subprograms (CSCs), thus satisfying the 2167A CSC definition. The requirements to show execution control and data flow now quite appropriately apply to executable subprograms.

The problem with defining a CSC as subprograms lies in the requirements called out under section 4 (Detailed Design). As a procedure, CSCs would have specific logic flow and algorithmic considerations which must be documented. According to the DID, these topics are addressed in Section 4 of the SDD during detailed design, not during preliminary design. Moreover, these considerations are associated with CSUs, not CSCs. We felt that not addressing these topics for each procedure would create an incomplete and perhaps faulty design document. This definition lead us to contradictory (or at the very least, inconsistent) documentation requirements. We now

believe that 2167A does not consider a CSC to be a procedure, but rather something at a higher conceptual level.

#### 3.1.1.3 Our Interpretation

We refined our concept of CSCs many times during the course of becoming more familiar with our design methodology. We always came back to the original idea that CSCs most closely modelled Ada library units. We eventually settled on this definition.

However, we determined we also needed a higher-level design unit than a CSC (as we defined it) around which to organize our design. We designated these higher-level units top-level computer software components (TLCSCs)[1]. Their purpose was to identify the software equivalents of the system's major functions. Thus, TLCSCs are logical in nature and don't directly map to any physical Ada equivalent. In summary,

a. TLCSCs represent major system processes/functions.

b. They are implemented by a collection of Ada library units.

c. Each TLCSC had a principle entry point - a procedure library unit.

We believe the TLCSCs served to make the system more modular and understandable.

In order to satisfy the requirement to show execution control and data flow, we developed a set of diagrams which were called object-relationship diagrams (ORDs). The primary purpose of these diagrams was to visually depict the dependencies between library units.

### 3.1.2 Interpreting the Term "CSU"

We conducted a survey of technical literature to identify possible definitions of a CSU. We found the following definitions were used:

- Any executable subprogram, possibly containing or calling other subprograms
- Ada packages/library units
- Terminal subprograms on the calling hierarchy. This is intended to satisfy the "separately testable" criterion, where the assumption is made that "separately testable" implies that the subprogram makes no calls to other subprograms.

#### 3.1.2.1 Interpreting CSUs as Library Units

We believe the type of information required for CSUs by Section 4 of the SDD (Detailed Design) clearly implies that the CSUs must be executable entities. For example, CSU design descriptions must include logic flow, detailed algorithms, and input/output elements for each CSU. Packages are not executable entities; hence, they cannot be designated as CSUs.

#### 3.1.2.2 Interpreting CSUs as Procedures

The possible choices for the definition of a CSU is directly impacted and limited by the definition of a CSC. Since CSCs are composed of CSUs, and packages are composed of procedures, the parallel was easy to draw. Thus, CSUs should be procedures. We found very few problems in implementing this concept. The design methodology chosen for the system was oriented around a functional decomposition of requirements, in parallel with an object-oriented viewpoint. Structure charts were used to diagram the design. This design tool readily leant itself to developing well-defined procedures and functions. Each module on the structure chart mapped neatly to an Ada procedure/function, which had well-defined execution and algorithmic requirements.

The only problem we encountered in this area came from the original 2167A definition of a CSU with respect to the words **"separately testable"**. Our customer interpreted this wording such that a CSU was a primitive construct and could not issue calls to any other executables. Using this criteria, only the terminal nodes on the structure charts could be considered CSUs. We adopted a liberal interpretation whereby high level procedures were considered elements by virtue of the fact that they performed a single, logical function. This did not, however, preclude external calls to other procedures. We also took a liberal interpretation of the requirement for a CSU to be separately testable. Again, we believe that a module can be separately tested if it is visible (in the Ada sense).

### 3.2 2167A Format Requirements

### 3.2.1 CSC Subordination: Logical vs. Physical

To provide design material in Section 3 of the SDD, a decision must be made as to how the concept of "sub-level" CSCs will be interpreted and applied. Sub-level relationships between CSCs can be defined along two different perspectives: a logical view and a physical view. The logical view would emphasize relationships based on similarities between CSCs. The physical view would emphasize the nature or structure of the code itself (e.g. nesting of procedures).

### 3.2.2 Logical

Sub-level relationships between CSCs using a logical perspective could be defined as groupings based on:

a. Layers or levels of decomposition (e.g. CSCs at the system-wide control level, the data transform level, external interface level, etc.)

b. Families of functionally similar CSCs (e.g. user interface CSCs, data base CSCs, etc.)

c. "Strings" of related CSCs that perform a distinct function.

In general, these groupings of CSCs into higher-level CSCs fall into two categories: horizontal subordination and vertical subordination (See Figure 3-1). "Horizontal" implies an organization of CSCs around layers that provide services to higher layers and utilize the resource provided by lower-level layers, as in [SHU88] and [SEI86]. "Vertical" implies an organization of CSCs around specific, (relatively) independent system functions. The advantages and disadvantages of each of these views needs to be identified in the context of the project under development.

### 3.2.3 Physical

Defining sub-level relationships between CSCs from a physical perspective could be defined as groupings based on:

a. Ada programming unit nesting (such as packages within packages, or lexically included subprograms)

c. Withing dependencies among Ada library units.

### 3.2.4 Our Approach

Our approach to developing an SDD follows. We concentrate here only on Sections 3 and 4, "Preliminary Design" and "Detailed Design," respectively.

#### 3.2.4.1 Section 3: Preliminary Design

In the course of mapping the requirements of the SDD DID to our design effort, we instituted significant modifications to the organization and content of Section 3. These modifications accommodated two goals: 1) To adequately address the relationships between CSCs based on the working definitions we assigned, and 2) To adequately depict each TLCSC's relationships with external entities.

#### 3.2.4.1.1 CSC Organization

As discussed in paragraph 3.1.1.3, we defined the CSCs that implemented top-level system functions as TLCSCs. Thus, our TLCSCs enforced a "vertical" approach. We further refined the concept of a TLCSC and a CSC to establish a relationship between them, the design methodology, and Ada library units:

a. TLCSCs are implemented by a collection of Ada library units

b. Each TLCSC has a principle entry point: a procedure library unit.

---

1 TLCSCs were originally defined in DoD-STD-2167.

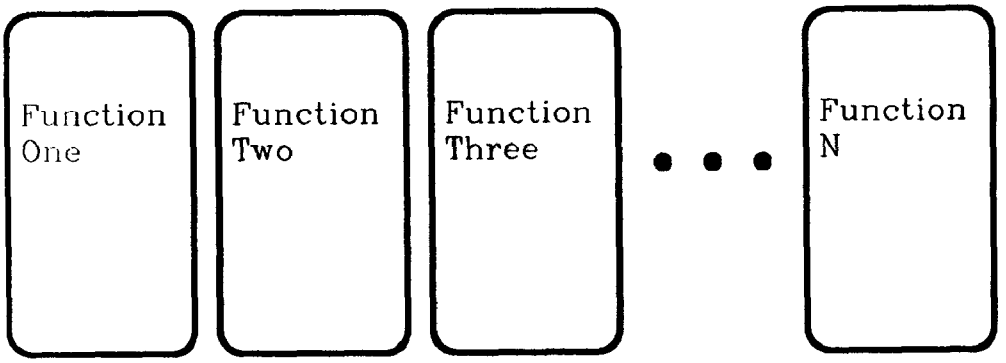


**Figure 3-1a Vertical Organization of CSCs**

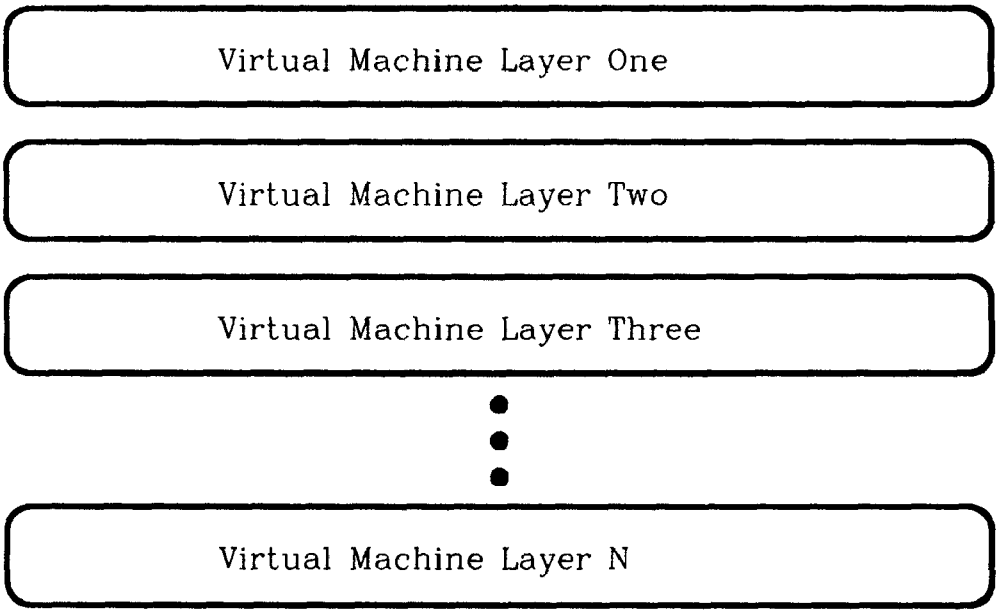


**Figure 3-1b Horizontal Organization of CSCs**

c. Physical (sub-level) CSCs were defined as either object-oriented packages or as virtual machine-oriented packages[2] or subprograms.

Finally, we recognized that many CSCs would be used as resources by several TLCSCs. These were designated as "shared" CSCs. To avoid complications in document production (such as extensive cross-referencing and the ripple-effect caused when a shared CSC is added to or deleted from the design), we put a shared CSCs in one of two sections: a "Shared Objects" section and a "Shared Virtual Machine" section. Thus, these CSCs were categorized around a horizontal organization. Our resulting CSC organization was a combination of the vertical and horizontal organizations (See Figure 3-2).

#### 3.2.4.1.2 External Interfaces

To accommodate the design of our user interface, and to specify the interaction of TLCSCs with the data base, we added a numbered subparagraphs to every TLCSC to document this information. User interfaces were documented in their own paragraph. Data base interfaces were depicten in a paragraph titled "Context Diagram." (Note: the concept of context diagrams was borrowed from Tom DeMarco [DEM79]. Their purpose is to show a system's interfaces with entities external to the system under study. We found this to be a useful means for identifying the interaction of a TLCSC with data base files, other TLCSCs, and commercial off-the-shelf (COTS) software.)

#### 3.2.4.1.3 Our Organization of Section 3.

The organization of Section 3 used for this project is shown in Table 3-1. Note that the SDD DID does not identify any explicitly numbered subparagraphs subordinate to paragraph 3.2.X; nor does it specify our paragraphs 3.3 and 3.4.

---

[2] The design methodology, LVM/OOD [SHU88], emphasizes the existence of both virtual machines and objects in a design. Virtual machines are functionally decomposed, while objects are an encapsulation of operations performed on abstract data types. The paper "Experiences in Applying the Layered Virtual Machine/Object-Oriented Design Methodology to an Ada Design Effort" details our use of LVM/OOD on this project.

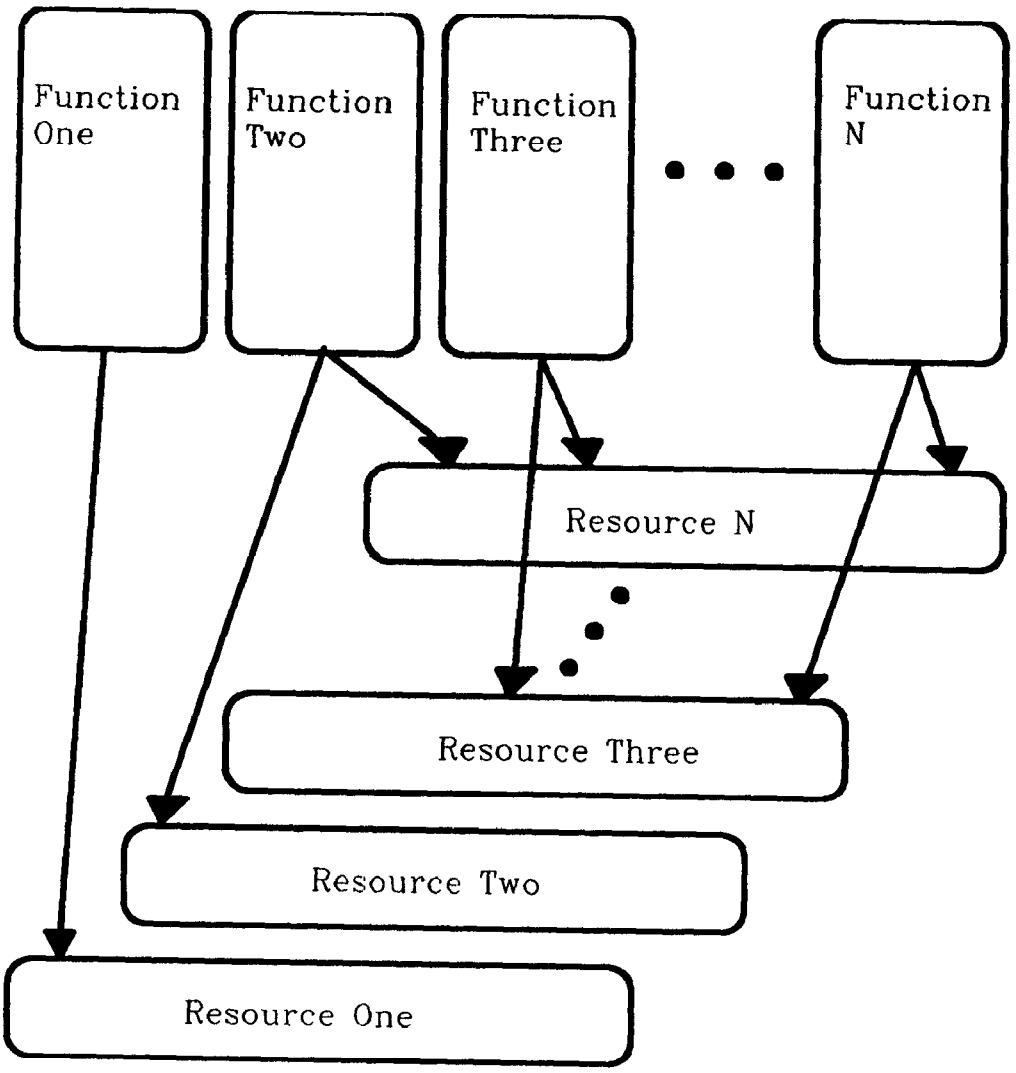


**Figure 3-2 Project's Organization of CSCs: A Combination of Horizontal and Vertical**

**TABLE 3-1 ORGANIZATION OF SECTION 3**

| PARAGRAPH | CONTENTS |
|---|---|
| 3.2.X | TLCSC X |
| 3.2.X.1 | Requirements |
| 3.2.X.2 | User Interface |
| 3.2.X.3 | Context Diagram |
| 3.2.X.4 | Design Considerations |
| 3.2.X.5 | Design |
| 3.2.X.6 | Subordinate Object-Oriented CSCs |
| 3.2.X.7 | Subordinate Virtual Machine CSCs |
| 3.3 | Shared Object-Oriented CSCs |
| 3.4 | Shared Virtual Machine CSCs |

#### 3.2.4.2 Section 4: Detailed Design.

In the course of mapping the requirements of the SDD DID to our design effort, we instituted significant modifications to the organization and content of Section 4. Some modifications were contractually required (specific portions of Section 4 were tailored out of the DID by the contract). We introduced additional modifications to accommodate two goals: 1) To distinguish between object-oriented and virtual machine CSCs (based on our application of the design methodology, as mentioned earlier in this paper), and 2) to separate the design information for file-oriented CSCs from other CSCs. The latter was desirable since no data base management system interface was specified by the contract. As such, these CSCs were likely to change during follow-on design efforts, so grouping them together would ease their modification at a later date.

The organization of Section 4 used for this project is shown in Table 3-2. Note that the SDD DID does not identify any explicitly numbered subparagraphs subordinate to paragraph 4.1.X.Y.2; nor does it specify the organization of Section 4 into 4.1 (Object-Oriented CSCs), 4.2 (Virtual Machine CSCs), or 4.3 (File-Oriented CSCs).

**TABLE 3-2 ORGANIZATION OF THE TAILORED SECTION 4**

| PARAGRAPH | CONTENT |
|---|---|
| 4.1 | Object-Oriented CSCs |
| 4.1.X | CSC X Description |
| 4.1.X.Y | CSU Y Description |
| 4.1.X.Y.1 | CSU Y Design Constraints |
| 4.1.X.Y.2 | CSU Y Design |
| 4.1.X.Y.2.1 | CSU Y Input/Output Elements |
| 4.2 | Virtual Machine CSCs (contains the same subparagraphs as paragraph 4.1) |
| 4.3 | File-Oriented CSCs |

#### 3.2.5 Difficulties Encountered

As a result of adopting this approach to applying the SDD DID to out design effort, we experienced difficulties with the inter-relationships between Sections 3 and 4, and with the volume of resulting documentation.

#### 3.2.5.1 Relationship Between Sections 3 and 4

We encountered two problems with our selected approach. First, significant portions of our design activity required parallel development of preliminary and detailed design documentation. Second, our definitions of CSCs introduced unwanted redundancies between Sections 3 & 4.

Much of our preliminary and detailed design work was done in parallel; distinctions between them were blurred. For example:

a. Structure charts are, by definition, CSU oriented; top-level design requires material to be organized that isn't appropriate (according to the SDD DID) until detailed-design.

b. The development of package specifications requires the detailed definition of data types/structures. Package inter-dependencies often result from this design activity. Thus, relationships between CSCs (packages) cannot always be established until detailed design.

Also, the discovery of new packages and package inter-dependencies during detailed design required an update of Section 3. We would have preferred a more static preliminary design.

Significant portions of our preliminary and detailed design documentation were redundant. For example, the purpose of each CSC was described in Section 3. In Section 4, we repeated this documentation to introduce the CSC.

#### 3.2.5.2 Volume of Documentation Produced

Each CSU requires about one page of documentation for simple CSUs; if limited private/private types are packaged, the CSUs that manipulate objects of these types tend to be primitive/trivial. Documenting them is tedious. For Section 4 material, expect ratios of pages of design documentation to pages of (uncommented, finished) code to be between 5-to-1 to 10-to-1. When other sections of the SDD are added into this ratio, it falls between 15-to-1 to 20-to-1.

Our SDD contained about 470 CSCs, and was 1800 pages long. At our customer's request, all numbered paragraphs were included in the table of contents. One hundred and forty of the 1800 pages were for the table of contents alone. Additionally, the Data Dictionary and the Data Base Specification (normally Sections 5 and 6, respectively, in the SDD), were about 700 and 650 pages long, respectively. Had we included these two documents in the SDD, it would have been about 3,150 pages long!

As a point of interest, our contract required 15 copies of each document to be delivered for each review. We had three reviews. This came to 3 * 15 * 3,150 = 142,000 pages. This is a stack of paper 79 feet tall!

#### 3.2.5.3 Customer Response

We initially had difficulty interesting our customer in evaluating or critiquing our design and documentation approach. They had no Ada background, nor were they familiar with object-oriented design concepts. Also, they had only a passing familiarity with the DoD-STD-2167A SDD with regards to its application to Ada projects.

This was a detriment to our design effort. We ended up spending a fair amount of time addressing Ada/design methodology/SDD issues during walkthroughs and reviews. This was a trivial matter compared to the atmosphere during the final months of the project: The customer changed management of our project, and we found ourselves covering substantial past history. Unfortunately, the bulk of these interchanges with the customer were conducted within the context of defending our approaches as being compliant with the DID - hence, the contract.

## 4. LESSONS LEARNED

Based on our experiences with applying the DoD-STD-2167A SDD DID, we would implement the following alterations to our use of the SDD:

- *CSC definition:* A CSC will be treated as a purely logical concept that doesn't as a necessity map to any particular Ada construct. CSCs will be identified during preliminary design and decomposed to a point where they can be mapped to physical Ada constructs during detailed design. This will alleviate the constant need to update preliminary design material in Section 3 whenever a new library unit is identified during detailed design. Also, it will allow designers to defer implementation details (such as defining package specifications) until detailed design. We would propose performing limited prototyping activities during the tail end of preliminary design to aid in selecting implementation strategies for CSCs during the detailed design effort.

  Also, we believe that Ada packages have no equivalent in 2167A terminology based on a strict interpretation of the SDD DID. Thus, we will utilize Ada packages during detailed design as a method of grouping CSUs. This will be accomplished by an agreed upon tailoring of the numbering scheme of Section 4.

- *DID Tailoring and Interpretation:* Any deviations to the format (i.e. paragraph numbering scheme, titles, location of material), as well as the working definitions of CSCs and CSUs, should be established at the time of contract negotiation. If during the course of the design process a better method is discovered, this should be coordinated with - and formally approved by - the customer.

As a result of this project, some of our basic assumptions have changed:

a. We had initially assumed that during preliminary design, functional requirements should be mapped to physical Ada software components (in the form of CSCs). Since CSCs were essentially Ada library units, we found that the discovery of new library units during detailed design required substantial updates to Section 3 (Preliminary Design) of the SDD. Since our document was 1800 pages in length, these updates were not trivial. We now believe it is extremely desirable to establish definitions of CSCs and CSUs that mitigate this effect. Thus, in future projects, we intend to define CSCs as purely logical units that will be less likely to change as the result of detailed design activities.

b. We has also initially assumed that the SDD need not be treated as a "bible," but rather as a guide. However, depending on SQA/IV&V/customer expectations, this can be a dangerous assumption. Our customer adopted the attitude that our adherence to the letter of the DID was a contractual issue. In other words, if we didn't strictly adhere to the DID, we were in violation of the contract.

## 5. CONCLUSIONS

### 5.1 Usability of the SDD DID

We had originally wanted to use the 2167A SDD DID because of the combination of explicit guidance and flexibility it provided. However, we found its guidance restrictive in some cases, and vague in others.

#### 5.1.1 Flexibility

At first glance, the SDD DID appears to offer the promise of great flexibility. This apparent flexibility takes several forms: the tailoring instructions, the content and format instructions (paragraph 10.1), the loose definition of key terms (such as CSC and CSU), and finally, the comforting phrase "this standard is not intended to specify or discourage the use of any particular software development method. The contractor is responsible for selecting software development methods ... that best support the achievement of contract requirements." [2167A]. We felt this flexibility would enable us to produce a comfortable, if not superior, design document tailored to our design approach.

Unfortunately, we encountered several difficulties with taking advantage of this perceived flexibility while producing a strictly DID-compliant document. For example, our definition of a CSC as an Ada library unit resulted in our inability to provide an accounting of the execution control and data flow between CSCs as required by Section 3 of the DID. Specifically, packages do not strictly pass data among themselves, nor is there any execution control between them. Rather, this is accomplished at the CSU level. Thus, our diagrams that showed library unit dependencies were considered by our customer to be inadequate.

Another case in point is the document preparation instruction stating "All paragraphs and subparagraphs starting with the phrase 'This (sub)paragraph shall...' may be written as multiple subparagraphs to enhance readability. These subparagraphs shall be numbered sequentially." We felt this allowed us to add subordinate subparagraphs where we felt they were needed. At the end of our contract, our customer informed us that this phrase also required *every* paragraph in the document to be numbered.

Finally, we found that 2167A in general was lacking in its documentation requirements for interactive systems. For example, there is no clean place to document the appearance and behavior of the user interface. This is in sharp contrast with MIL-STD-7935, which explicitly requires the documentation of input screen formats and system report formats [7935].

#### 5.1.2 Understandability

Much of the difficulty we had with understanding how to satisfy the documentation requirements of the SDD DID originated in the vagueness of many of the terms used in the DID. For example, design limitations, design constraints, design specifications, design requirements, and derived design requirements must be provided. Unfortunately, neither a definition nor a statement of the purpose of these terms are provided in the standard.

Naturally, this allows the unsavory condition to exist where the customer can easily take issue with the working definitions established by the contractor. For example, the standard states that CSUs must be "separately testable". Separate from what? In the final months of our contract, we were strongly encouraged by our customer to redefine a CSU to be any subprogram that didn't perform calls to other subprograms. To them, this was the essence of "separately testable."

### 5.2 Summary

After our experiences in applying the 2167A SDD DID to an Ada project, we feel we can safely make the following observations:

a. The consistent trust and goodwill of the customer towards your design effort will be the single most important factor in your ability to complete your project on time and within budget. For example, Lease says they were "fortunate in having a customer who did not demand a literal interpretation of DoD-STD-2167, but rather was supportive of ... attempts to tailor DoD-STD-2167 for the proper use of Ada." [LEA88]

b. The selection of working definitions for CSCs and CSUs has the largest impact on the manageability of the document and the ability to satisfy the literal requirements of the DID. Working definitions of CSCs and CSUs should be selected with an eye towards minimizing the degree of change required when additions of CSCs and/or CSUs are required as a natural part of the design process.

c. The SDD DID provides very little solid ground to stand on when a difference of opinion over interpretation arises - either internally, among design team members, or externally, with the customer or IV&V personnel.

d. The project's design and development approach, as well as the interpretation of the requirements of each contractually required design document, should be incorporated into the contract as a valid and binding document. This document could be in the form of a Technical Proposal or a Software Development Plan.

## 6. REFERENCES


[1703] *NSA/CSS Software Products Standard Manual*, April 15, 1987.

[2167A] DoD-STD-2167A, *Military Standard Defense System Software Development*, February 29, 1988.

[80012A] DI-MCCR-80012A, *Software Design Document*, February 29, 1988.

[DEM79] DeMarco, T., *Structured Analysis and System Specification*, Yourdon Press, Englewood Cliffs, NJ, 1979.

[ELL89] Ellison, K. S., and Goulet, W. J., "A Practical Approach to Methodologies, Ada and DoD-STD-2167A," *Proceedings of the Seventh Annual National Conference on Ada Technology*, Atlantic City, NJ, March 14-16, 1989, pp. 51-57.

[LEA88] Lease, D., "A Marriage of Convenience: Developing a Practical APSE for Use with Ada and DoD-STD-2167", *Proceedings of the Sixth National Conference on Ada Technology*, Arlington, VA, March 14-17, 1988, pp. 57-64.

[MEA89] Mead, N. R., and Lockhart, R. J., "Using a Multi-Level Design Method Under DoD-STD-2167A," *Proceedings of the Sixth Washington Ada Symposium*, Washington, D.C., June 26-29, 1989, pp. 21-38.

[SEI86] Seidewitz, E., and Stark, M., *General Object-Oriented Software Development*, National Aeronautics and Space Administration, Goddard Space Flight Center, Greenbelt, MD, 1986.

[SHU88] Shumate, K., "Layered Virtual Machine/Object-Oriented Design," *Proceedings of the Fifth Washington Ada Symposium*, Washington, D.C., 1988, pp. 177-190.