APPLYING DOD-STD-2167A

James O'Day
Hughes Flight Simulation Operations
Herndon, VA

ABSTRACT

DOD-STD-2167A is rapidly becoming an international defacto standard for software development in the defense industry. This is largely due to the size of the Department of Defense market for software intensive applications and the lack of a readily available and more widely accepted standard for software development. At the same time DOD-STD-2167A is on its way to becoming one of the most widely used standards it is also one of the least understood software development standards in history. Misconceptions and confusion about the application and tailoring of DOD-STD-2167A are common and stem from a variety of factors. This paper will discuss some of these factors as well as issues, potential pitfalls, and approaches to applying DOD-STD-2167A.

BACKGROUND

DOD-STD-2167A, Defense System Software Development, was released in February 1988 and replaced DOD-STD-2167 as the single software development standard for the Department of Defense for mission critical computer resources. DOD-STD-2167A came about largely as a result of comments received from industry and the Ada community on DOD-STD-2167. Although DOD-STD-2167 was programming language independent, many of the issues involved incompatibilities between the standard and its use with Ada. Other problems included incompatibilities with new and evolving software engineering technologies. The Software Development Standards and Ada Working Group (SDSAWG) of the Association for Computing Machinery *Special Interest Group on Ada (SIGAda) headed by Don* Firesmith was instrumental in focusing attention on these as well as other issues during the revision of the standard. Major inadequacies identified in the June 1985 release of DOD-STD-2167 included: [2] [9]

- Incompatibility with Ada and Ada related issues
- Restrictions on development methodologies
- Lack of system engineering emphasis
- Confusion regarding tailoring
- Excessive "how to" imposed on contractor
- Overlap with DOD-STD-2168
- Confusion regarding software quality factors
- Confusion concerning application to firmware
- Confusion concerning application to non-deliverables
- A need to enhance readability
- Confusion regarding software development files
- A need to ensure adequate support planning and transition
- Excessive requirements for "informal" testing
- A need to encourage the use of reusable software
- A need to emphasize traceability of requirements
- A need to emphasize safety
- Need to establish engineering and test environments
- Confusion regarding qualification methods
- New requirements needed for testing
- Inadequate review of Data Item Descriptions (DIDs)

The revisions to DOD-STD-2167 that resulted in DOD-STD-2167A reduced the size of the standard by 40 pages and eliminated 8 DIDs. Even though the revisions to DOD-STD-2167 represented a significant improvement, there have been numerous problems and varying levels of confusion as projects first begin to apply DOD-STD-2167A.

MISCONCEPTIONS ABOUT DOD-STD-2167A

In an effort to communicate several of the more common misconceptions and clear up some of the confusion about the application of DOD-STD-2167A, the Joint Logistics Commanders presented a paper [10] at the Tri-Ada'89 Conference during October of 1989. Their presentation centered on 10 major issues. The following is a summary of their presentation.

1. DOD-STD-2167A does not impose the Waterfall Development Model. It does specify a set of activities that must take place during a software development project, but does not impose any requirements that these activities be performed sequentially or that one activity cannot begin before another is complete.

   Paragraph 4.1.1 of the standard indicates what these activities are and that they may overlap and be applied iteratively or recursively. Figures 1 and 2 of the standard while depicting a traditional sequential Waterfall approach to software development do not necessarily impose this method.

2. MIL-STD-1521B is in conflict with DOD-STD-2167A and is in the process of being revised. MIL-STD-1521B requires a sequential relationship between software development activities. For example it requires that activities such as coding not begin until Critical Design Review (CDR) has been completed. It was not the intent of DOD-STD-2167A to impose this rigid ordering on activities. DOD-STD-2167A also permits multiple reviews (PDRs and CDRs) as indicated in the standard on Figure 1. This permits flexibility to plan and schedule reviews in a manner that is appropriate to the project. Guidance on tailoring MIL-STD-1521B is contained in MIL-HDBK-287 [12].

3. The Software Design Document (SDD) DID (DI-MCCR-80012A) does not preclude the use of iterative/recursive design methodologies. DI-MCCR-80012A does describe what sections of the SDD are to be presented at PDR and CDR but does not impose a limit on the number of PDRs or CDRs.

4. Timing and sizing information required by the Software Design Document DID (DI-MCCR-80012A) at PDR is appropriate at that stage of development. It needs to be recognized that this information is preliminary and is merely a refinement of estimates that have been made previously when sizing and costing a system. The purpose of this information is to help determine the proper amount of resources have been allocated and to uncover potential problems as early as possible in the development process.

5. DOD-STD-2167A does not impose top-down functional decomposition of requirements. Paragraph 4.2.1 of the standard requires that the contractor use systematic and well documented software development methods that support the formal reviews and audits required by the contract. Figure 3 of the standard represents an example of an organizational structure and standard set of terms for communicating and documenting a software design not the actual software architecture or calling structure of a system.

6. DOD-STD-2167A does not impose bottom-up testing. Although the standard requires CSU, CSC, and CSCI testing no order of activities is imposed. Testing methodologies may be mixed as appropriate for the project.

7. DOD-STD-2167A is compatible with projects organized into "builds". Projects that are organized into builds are required to make multiple passes through the software development process. Each pass adds increased functionality to the software product being produced and since each build is intended for operational use it represents a deliverable product that must be supported as the product evolves.

   Builds may be treated as separate contracts and DOD-STD-2167A can be tailored to each build. Builds may also be treated as one contract by tailoring the Statement of Work to reflect DOD-STD-2167A tailoring for each build. Tailoring would eliminate or update specific data items for each build as appropriate for that build.

8. DOD-STD-2167A is compatible with prototyping. Prototyping may be used for a variety of applications. Two major categories of prototypes are deliverable and non deliverable. Deliverable prototypes are usually associated with contracts involving concept exploration-demonstration-validation. Examples of possible non deliverable prototypes would be in applications such as requirements definition and risk abatement.

   Since DOD-STD-2167A imposes no required order on activities, there is no conflict in designing, coding, and testing software in support of prototyping. If the prototype is to be delivered and eventually evolve into an operational system then DOD-STD-2167A should be tailored to ensure the software is supportable over its lifecycle. "Throw-away" prototypes are not intended to be supported or delivered and DOD-STD-2167A should be tailored to minimize costs and reflect the limited objectives of these types of programs.

9. DOD-STD-2167A is compatible with Object Oriented Design. As stated previously paragraph 4.2.1 of the standard only requires that the contractor use systematic and well documented development methods. This does not preclude the use of new methodologies such as Object Oriented Design or Object Oriented Requirements Analysis, and only requires that they be well documented.

10. DOD-STD-2167A is compatible with Ada. The standard has been written to be language independent. CSCIs, CSCs, and CSUs are the elements provided by the standard that are to be used to communicate the software architecture of a project. The constructs and capabilities of the language being used need to be mapped to the definitions provided in the standard.

APPLICATION PROBLEMS

Even though DOD-STD-2167A has been in use for over two years, there are several factors which are hampering its application to software projects. These include:

- □ Vagueness in the standard and its DIDs intended to permit its use on a wide range of DOD software applications makes it difficult to apply [6].

- The need for the standard to allow for advances in software technology limits the amount of guidance that can be provided.

- A lack of historical basis and experience to aid practitioners in the decision making and application process makes it difficult to apply.

- A general lack of communication between contractor and contracting agency leads to different interpretations of the standard and its DIDs.

- A lack of an educational effort and resources similar to what accompanied the introduction of Ada has made it difficult to educate contractors and contracting agencies in its use.

- The late arrival MIL-HDBK-287 and tailoring guidelines meant that several programs began with minimal or no tailoring being done to the standard or its DIDs.

- Limited knowledge of the rationale behind the document makes it difficult to apply.

- Confusion on how to map software architecture into DOD-STD-2167A terms has the potential to create serious cost and schedule impacts.

DOD-STD-2167A was designed to be used in a variety of DOD applications. Because these applications span many problem domains, each with its own unique set of requirements, the standard and its DIDs were written to be flexible and capable of being interpreted and tailored in a variety of ways to accommodate varying application domains. Software engineering technology is also rapidly changing and evolving as the challenges and complexities of modern weapons system software development increase. This in turn has required that the standard be flexible enough to take advantage of advances in new technologies. The need to provide this flexibility has meant that the standard and its DIDs tend to be vaguely worded documents open to varying interpretations with varying cost and schedule impacts. Vagueness and the lack of rationale in the standard and its DIDs have also make it difficult to defend or dispute decisions made when applying and interpreting the standard.

Communication is clearly the most important element in tailoring DOD-STD-2167A to a project. Unless the contracting agency and contractor are both familiar with the standard and its tailoring and are willing to communicate with each other the costs of applying the standard can easily consume a disproportionate share of project resources adversely impacting the project. Estimates of complying with DOD-STD-2167A requirements vary widely and can range from less than 10% to as high as 40% of total software development costs. This is clearly dependent on the number of deliverables required and how DOD-STD-2167A is interpreted and applied.

The issue of communication also involves understanding what products and levels of detail are necessary to give the contracting agency visibility into the software development process and what products are necessary to maintain the system over its lifecycle. Most of the costs associated with complying with DOD-STD-2167A are in producing and maintaining the documentation associated with the DIDs. Since methods such as technical interchange meetings and multiple reviews can be used to provide insight into the software development process rather than volumes of intermediate documentation, the contracting agency and contractor should consider the cost impacts and options available to them as well as the potential benefits to the project when tailoring DOD-STD-2167A.

The relatively recent arrival of DOD-STD-2167A has meant that the experiences and results of the first few programs using it are just now slowly starting to become public. This provides a relatively small historical basis of experience to draw from making it more difficult for the average software developer to apply the standard in his/her work for the first time. The general lack of organized industry feedback on the standard and its application has hampered the communication of lessons learned making it more likely the same lessons will be learned more than once as the defense industry transitions to the standard.

The lack of resources associated with educating both government and industry in the application of the standard has also resulted in many of the same types of lessons being learned more than once. There exists no communications mechanism such as a computer bulletin board or officially sanctioned periodic newsletter on DOD-STD-2167A that can publicize guidance or case study materials that would aid in the application of the standard. This is unfortunate considering the potential cost and schedule impacts that varying interpretations of the standard can have on a program across its lifecycle. The introduction of Ada has involved a vast amount of resources and energy but it all will have been wasted if the reduced lifecycle costs it was intended to deliver are consumed by an over interpretation of DOD-STD-2167A requirements.

MIL-HDBK-287 provides guidance on how to tailor DOD-STD-2167A. Unfortunately it was late in arriving and does not provide a great deal of rationale on what is trying to be accomplished on a paragraph by paragraph basis in the standard or its DIDs. Rationale would make it easier to comply with the intent of the standard and make it simpler to determine when sections of the standard and DIDs should be tailored out of a contract.

Another difficulty in applying DOD-STD-2167A is in the area of mapping the software architecture into the terms provided in the standard. This is an area where cost and schedule impacts caused by documentation are directly proportional to how the terms Computer Software

Configuration Item (CSCI), Computer Software Component (CSC), and Computer Software Unit (CSU) are interpreted. This is further complicated by the fact that some software tools that could help generate major portions of required documentation are constrained by assumptions made by their designers on how these terms would be interpreted.

MAPPING ADA TO DOD-STD-2167A

Mapping the software architecture of a system into CSCs and CSUs is proving to be one of the more difficult tasks in applying DOD-STD-2167A. This is because of a variety of reasons including:

- ◻ Any mapping will have a major influence on the cost and schedule of producing such documents as the SDD. There are many mappings possible each with advantages disadvantages and varying cost and schedule impacts.
- ◻ Many of the principles and constructs embodied in modern programming languages such as Ada do not map well to the definitions in the standard for a CSC or a CSU.
- ◻ It is not clear in a mapping process how many of the elements should represent physical entities and how many of them should represent logical groupings or processes.
- ◻ The best software engineering design may make the documentation process too expensive in a competitive biding situation and the software engineering process can become document driven to the point that it ends up impacting other software design activities.

As mentioned previously, mapping the software architecture into the logical structure and terms defined in DOD-STD-2167A can be a major cost and schedule driver. This is mainly due to the high level of detail required in deliverables such as the Software Design Document. This is easily illustrated when you consider the mapping of Ada constructs to CSCs and CSUs. If Ada packages are used to represent CSUs rather than Ada subprograms the number of units to be documented will be significantly less than for CSUs based upon a subprogram mapping. On larger projects this difference can easily amount to thousands of pages of documentation and thousands of dollars. This is why the documentation process and software architecture mapping process are becoming increasing important cost issues in programs using DOD-STD-2167A. Since the standard provides the flexibility to accept either mapping; the major unknown in costing the documentation for a system becomes the contracting agency's acceptance of the contractor's mapping strategy and the contracting agency's willingness to tailor the levels of detail required by the DIDs. Since this is usually first documented in the Software Development Plan, which is not approved until after contract award, it is often too late to correct a bid that was based upon the wrong assumptions.

Because there are so many more capabilities and constructs in modern programming languages such as Ada, there are many more ways to map a software architecture into DOD-STD-2167A terms. Unfortunately it is difficult to find one that will consistently address all the requirements of the SDD DID without putting limits on or impacting the software design. For example, the logical concept of a CSC is more closely related to one or more Ada packages rather than any other Ada construct, yet because of the wording of the SDD DID it must have data and control flow associated with it. Since a package is a mechanism for grouping logically related entities it has no data or control flow in and of itself. For this mapping to take place, the CSC becomes a logical entity in and of itself that is partially composed of physical entities such as Ada packages. This results in documentation that maps to both logical and physical terms making the job of understanding the software architecture and doing software maintenance more difficult. The mappings tend to be much cleaner for languages such as Fortran which have less capability in hiding and localizing visibility of data. Other issues in this category include constructs and capabilities that should be documented but do not clearly fall under the CSC or CSU organizational structure. This includes such things as Ada packages composed entirely of types and constants, tasks, generics, nested packages, and nested subprograms.

Another issue that falls out of the need to map the software architecture into DOD-STD-2167A terms is the issue of a document driven software development approach. DOD-STD-2167A was intended only to give the contracting agency visibility into the software development process and to produce documents that could be used in maintaining the software that was developed throughout its lifecycle, not to influence or determine the software design. While a software design that might make heavy use of Ada private types to protect and control operations on variables might be good for maintaining a system in the long term, in the short term it would add thousands of subprograms that, under a CSU to subprogram mapping strategy, would each require several pages of documentation in the SDD. This would increase the initial development costs in what may be a competitive fixed price procurement. Other spinoffs from strict interpretations of documenting every small subprogram will be a reluctance on the part of the software developer to limit functionality when designing subprograms. This will mean that software developers will try to make subprograms do more than they should to avoid documenting additional subprograms. In the long term this can adversely impact the maintainability of the code. It sometimes appears that the drive to produce more maintainable and readable source code and the process of software documentation are working against each other rather than complementing each other.

Specific problems and mapping strategies for DOD-STD-2167 and DOD-STD-2167A have been discussed in several recently published papers and presentations [3] [4] [6]. These papers point out the difficulties in developing a mapping strategy and contain more details on pitfalls in making these types of decisions. The great flexibility available in developing strategies for mapping software architectures to DOD-STD-2167A points out more than ever before the need to reexamine our traditional approach to documentation in light of what needs to be accomplished to provide insight into the software development process and what is necessary for continuing software maintenance.

DATA ITEM DESCRIPTIONS

One of the most frequently complained about areas in applying DOD-STD-2167A is in the area of the Data Item Descriptions. The two DIDs that are most frequently mentioned are those for the Software Requirements Specification (SRS) and Software Design Document (SDD). The most common complaints center around excessive levels of detail.

While DOD-STD-2167A describes a preliminary and final version of the SRS, neither it or the SRS DID provide any guidance on the what level of content is appropriate for a preliminary versus a final submittal. Other problems areas mentioned with the SRS include the excessive amount of detail required for data elements. Requirements analysis can easily slip into a design effort rather than a requirements effort. For example if the contractor is modeling a real world system he must analyze how the real world system operates and then make decisions about how his model will operate. Along the way he is likely to make some assumptions and simplifications about what data elements will be required from the real world system to model the system. In doing so he has changed the way data elements are generated from the real world and created new interfaces that have no counterpart in the real world; in the process of doing so he has drifted into design and his design now begins to influence the requirements instead of the other way round.

This problem is not unique to SRS associated with DOD-STD-2167A but is part of the classic problem of how to separate requirements from design or the "what" from the "how". Unfortunately requirements analysis is different in each application domain and excessive detail tends to encourage the "how" rather than the "what" in an attempt to fill in all the data on internal interfaces.

In the SDD the problem of excessive detail is clearly a design issue. In this case the data elements are clearly part of the design. In any major software development effort there are likely to be thousands or hundreds of thousands of data elements that will have to be documented according to instructions in the DID. This information tends to be very volatile during software development, expensive to update, and takes the software developer away from other important activities. The end product is usually more paper than there are resources to read it and so much detail it is difficult for a maintenance programmer to grasp the system design because it has been obscured by the detail that implements it.

Unfortunately the approach that has been taken in the SDD DID ignores some of the very features that Ada was intended to address. These include more readable code that becomes its own documentation when supplemented with good system and subsystem overviews in design documentation. Most maintenance programmers do not trust the documentation of a system, especially once they find even the smallest error or inconsistency, and usually go directly to the code once they have a sufficient understanding of the system. Many of the paragraphs of the SDD DID deal with providing very specific detail while at the same time place few requirements on the very important area of a system overview of how the system functions as a whole and interacts with itself. While a maintenance programmer can easily extract detail from source code it is much more difficult for him to build an understanding of how the system functions as a whole and understand its interactions from detailed code listings. The maintenance programmer is also faced with another major undertaking which is to maintain the SDD that he has inherited. The larger and more detailed the document the harder and more expensive the task will be to maintain it over the system's lifecycle.

The concept of automating documentation so that much of the required material may be extracted from source code and made into documents such as the SDD only solves half the problem. The half that is solved is the production part, the part that is missing is a way to read and understand volumes of information that are produced. This approach is in fact addressing the wrong problem. Producing large volumes of documentation is not the problem - producing good high quality documentation that provides the right amount of information is the problem.

TAILORING DOD-STD-2167A

There are several common misconceptions regarding the tailoring of DOD-STD-2167A. The most common of these is that tailoring can only be done once and this is at the beginning of the program. Paragraph 4.3.3 and Figure 4 of MIL-HDBK-287 show that tailoring is an ongoing process that begins before the first draft Request for Proposal (RFP) is issued and continues after contract award until the project has been completed. This points out that tailoring can take place anytime during a program that it makes sense. This is especially important contractors and contracting agencies who are applying DOD-STD-2167A and its DIDs for the first time.

There are several restrictions on the tailoring process that are enumerated in paragraph 4.3.5 of MIL-HDBK-287. These include such things as tailoring instructions for DOD-STD-2167A are to be specified in the Statement of Work (SOW); tailoring instructions for the

DIDS are to be specified in the Contract Data Requirements List (CDRL); requirements can not be added to DIDs; and requirements may not be added to DOD-STD-2167A but may be added to the contract using the SOW.

MIL-HDBK-287 also stresses that tailoring is a team effort involving contractors as well as contracting agencies. The primary means of communicating the contractor's recommendations for tailoring comes in the contractor's Software Development Plan. This is the place the contractor describes his DOD-STD-2167A compliant software development process and any tailoring that he feels is appropriate.

The handbook also describes the mechanics of the tailoring process and provides sample checklists of tailoring examples. Although the handbook does provide some insight into DOD-STD-2167A it does not provide enough of the rationale behind the paragraphs contained in DOD-STD-2167A and its DIDs to really understand and interpret those vaguely worded paragraphs that are likely to be the most difficult and costly to comply with. In the absence of this rationale, contracting agencies and contractors must develop a mutual understanding of the intent of each paragraph before effective tailoring can occur. This is difficult with limited experience with DOD-STD-2167A and causes ineffective tailoring to occur, resulting in higher than necessary software development costs.

ABOUT THE AUTHOR

James O'Day is a software engineering supervisor with the Flight Simulation Operations of Hughes Simulation Systems Inc., Training and Control Systems Division. He is responsible for the management of the Flight Simulation Operations Software Support Center and has over nine years experience in software development of flight simulation devices. His previous experience includes over 1500 hours of flight experience as an Air Force helicopter pilot in Special Operations and spacecraft recovery missions. He holds a Masters Degree in Systems Management from the University of Southern California and a BSEE from the U.S. Air Force Academy. Mr. O'Day has also completed the course work requirements towards a Masters Degree in Computer Science at George Washington University.

REFERENCES

[1] Firesmith, D., *Ada and DOD-STD-2167A*, presentation; National Institute for Software Quality & Productivity Case Technology Conference April 11-12, 1988.

[2] Firesmith, D. and Gilyeat, C., *Resolution of Ada - Related Concerns in DOD-STD-2167, Revision A*, Ada Letters, Vol VI Number 5 September,October 1986, pages 29 - 33.

[3] Gray, L., viewgraphs Washington D. C. ACM SIGAda Presentation, Sept 27, 1989.

[4] Grau, J. K., and Gilroy, K. A., *Compliant Mappings of Ada Programs to the DOD-STD-2167 Static Structure*, Ada Letters, Vol VII Number 2 March, April 1987, pages 73 - 84.

[5] Maibor, D. S., viewgraphs Washington D. C. ACM SIGAda Presentation, Sept 27, 1989.

[6] Meyer, C. A., Lindholm, S. C., and Jensen, J., *Experiences in Preparing a DOD-STD-2167A Software Design Document for an Ada Project*, Association of Computing Machinery, Inc. TRI-Ada'89 Proceedings; pages 118 - 124.

[7] Springman, Michael, *Software Design Documentation Approach*, Association for Computing Machinery, Inc. TRI-Ada'89 Proceedings; pages 93 - 103.

[8] Whitney, S., viewgraphs Washington D. C. ACM SIGAda Presentation, Sept 27, 1989.

[9] Whitney, S., viewgraphs; *Changes from DOD-STD-2167 to DOD-STD-2167A*, March 1, 1987.

[10] Joint Logistics Commanders, *Software Development Under DOD-STD-2167A: An Examination of Ten Key Issues*, presentation - Association for Computing Machinery, Inc. Tri-Ada'89 Conference, Oct 23-26, 1989, Pittsburgh, Pa.

[11] *DOD-STD-2167A Defense System Software Development*, Department of Defense, 29 February 1988.

[12] *MIL-HDBK-287 A Tailoring Guide for DOD-STD-2167A*, Defense System Software Development, 11 August 1989.