SHOULD THE DOD MANDATE A STANDARD SOFTWARE DEVELOPMENT PROCESS?

Donald G. Firesmith
Software Methodologist

Magnavox Electronic Systems Company
Fort Wayne, Indiana

Abstract

This paper addresses the question of whether the DoD should mandate via "Defense System Software Development" (DOD-STD-2167) a standard software development process and life-cycle on private industry. It also questions the cost-effectiveness of establishing either required or default software development methods. It details both general problems with process standards for software development as well as specific problems relating to DOD-STD-2167. It concludes with the author's recommendations for solving these problems.

Keywords

DOD-STD-2167, Life-cycle, Process Standards, Software Development Methods

Acknowledgement

The author would like to acknowledge that this paper is heavily based upon the input provided by many members of the Ada (*) community to the SIGAda Software Development Standards and Ada Working Group (SDSAWG)[1] as part of the review process of DOD-STD-2167A. The author, however, takes sole responsibility for the opinions and recommendations contained herein. This paper does not, therefore, express the official position of the ACM, SIGAda, the SDSAWG, or Magnavox Electronic Systems Company.

Introduction

DOD-STD-2167 should be viewed from a historical perspective. The opinion seems to have been that many software contractors did not know how to properly develop software --otherwise, why would so many products of questionable quality be delivered overdue and overbudget? One of the DoD's answers to this software crisis was to teach industry how to develop software by mandating a "proven" process and life-cycle and by establishing "proven" default methods.

Like MIL-STD-1679 before it, DOD-STD-2167 is a process standard that "establishes a uniform software development process"[2] and mandates "requirements to be applied during the development ... of software in Mission-Critical Computer Resources"[3]. Unlike its predecessors, however, DOD-STD-2167 is a Tri-services standard which will be a requirement on a great many projects.

DOD-STD-2167 mandates the classic "waterfall" developmental life-cycle, and its basic process is strongly tied to that life-cycle. DOD-STD-2167 also either mandates, or establishes as a default, certain specific methods such as the use of a set, and standard formal reviews, the use of Program Design Languages (PDL's), and Software Development Files (SDF's), and decomposition methods that are functional and hierarchical. DOD-STD-2167 therefore restricts the contractor to those software development methods that are consistent with these requirements and defaults. This approach, as we shall see, is not without serious drawbacks.

Before looking at these problems, however, it is valuable to first consider the DoD's position with regard to DOD-STD-2167.

The DOD's Position

The Computer Software Management Subgroup of the Joint Logistical Commanders (JLC/CSM) is currently responsible for DOD-STD-2167. The position of the DoD, as stated by the JLC/CSM Subgroup, is that:

> "DOD-STD-2167 should be a process, as well as a product, standard. As currently defined, the software development process is driven by the DoD Acquisition process and must be an integral part of that process. In order for the government to maintain a common, single development process throughout a variety of software development projects, it is necessary for DOD-STD-2167 to define and direct the methodology for developing software. For those instances in which the contractor does not propose a development methodology in the Software Development Plan (SDP), a DEFAULT methodology is required. In these instances, the government must define the development methodology used in order to maintain control over software development. The government directed DEFAULT methodology is ... directed ... ONLY

(*) Ada is a registered trademark of the U.S. Government (AJPO).

when the contractor has not specified an alternative development methodology. A contractor always has the option of proposing an alternative methodology in the SDP. While the JLC/CSM Subgroup recognizes that a process standard inhibits innovation to some degree, it must be emphasized that DOD-STD-2167 does not prohibit the use of different development methodologies than those defined in the standard. If a different development methodology is used, it must be documented in the SDP and will be subject to government disapproval. The intent of DOD-STD-2167 is to allow the contractor to propose what he believes to be the best development methodology for a particular software project."[4]

General Problems

The question should not be whether the software development process, methods, and life-cycle mandated by DOD-STD-2167 are the best ones currently available; that is debatable. The question is whether any single process, method, or life-cycle, however general, should be mandated on the contractor. Even the best current process, methods, and life-cycle rapidly become obsolete as our industry evolves. By singling out a specific process, life-cycle model, and set of methods in DOD-STD-2167, the DoD is ensuring that the standard will become obsolete sooner than is necessary. And because the standard cannot be easily and rapidly updated, this will certainly have major negative consequences on the development of DoD software.

The following general problems have been raised with regard to the process standard nature of DOD-STD-2167:

"How to" Constraints. Although the "intent of (DOD-STD-2167) is to permit any systematic, well-documented, proven software development methodology"[2], process standards must, by definition, contain "how-to" restrictions. By mandating a standard life-cycle and activities based upon the phases of this life-cycle, DOD-STD-2167 is no exception. DOD-STD-2167, therefore, permits only those software development methods that are consistent with its mandated process. Other methods are permitted only if one either proposes the inconsistent methodology in the SDP or tailors the requirements out, both approaches that are difficult and not without significant economic risk to the contractor.

Although it may be reasonable for the government to adopt a process standard to dictate how the government is to PROCURE software, it is something else entirely to mandate via a process standard how contractors are to DEVELOP software. This is an improper "how to" constraint on the contractor. Therefore, DOD-STD-2167 is in direct violation with the government's own Acquisition Streamlining Directive[5] which states: "As a first priority, this Directive establishes policy for streamlining ... contract requirements by: (a) Specifying contract requirements in terms of the results desired, rather than 'how-to-design' and 'how-to-manage'...". This directive mandates further that "a contractor's management systems, internal procedures, methods, processes, and data products shall be used instead of specifying other approaches unless the acquisition activity determines that the contractor's approaches cannot satisfy the program needs." Thus the government should mandate WHAT products it wants, not HOW the contractor is to develop these products. It is clearly the contractor's, and not the government's, responsibility to define the process, methods, and life-cycle model to be used to produce software.

One counter argument is that an important objective of the Software Standardization Program of the JLC was to establish a well-defined and easily understood software development process. DOD-STD-2167 was thus intended to be a process standard, was meant to dictate "how to" constraints, and has fulfilled its intention. For years, while software has become an ever larger part of any given system, the government has operated without a standard process for procuring software. This lack of a standard has resulted in many failures and additional expense.

However, there is a great difference between a process standard for procuring software and one for developing it. And as far as failures and additional expense are concerned, one can easily argue that the "how to" constraints have not, and cannot, solve the software crisis but rather have added greatly to the cost of defense software.

Another counter argument is that the contractor is free to propose an alternative software development process, method, or life-cycle so long as it is specified in the Software Development Plan (SDP) and is not disapproved by the contracting agency. If the contracting agency is truly interested in innovative approaches, the requirements and defaults of DOD-STD-2167 do not necessarily prejudice them.

However, although defaults may not unduly influence all contracting agency personnel, many developers are convinced that this is a real problem with numerous such personnel. Thus, the ability to propose an alternative (i.e., to ignore all or part of DOD-STD-2167) is an insufficient loophole since many contractors will certainly feel pressured to comply with the standard to win contracts. (Note: I will have more to say about this later.)

Another counter argument is that default approaches are needed for those instances where the contractor has not defined a software development process.

However, this is clearly specious because paragraph 5.1.1.3.c.1 of DOD-STD-2167 already requires the contractor's proposed software development methods and techniques to be documented in paragraph 10.2.5.1 of the Software Standards and Procedures Manual. Although not mentioned in the text of DOD-STD-2167, the exact same require-

ment is also REDUNDANTLY stated in paragraph 10.2.7.1.1 of the DID for the Software Development Plan. Besides, when a contractor does not propose a process or methods, the SSMP and SDP should be rejected as noncompliant rather than mandating a single, standard, default approach that may well not be appropriate.

Innovation Inhibition. Process standards inhibit innovation. Due to the perceived political and economic risks of proposing anything different than what the contracting agency expects, there is a very natural tendency for contractors to hesitate proposing software development processes, methods, and life-cycles that significantly deviate from those of DOD-STD-2167, regardless of their technical merit. The management of several companies have already mandated compliance with DOD-STD-2167 on their technical staffs and use this policy as part of their marketing strategy. Yet innovation is necessary and must be promoted in a rapidly evolving industry in which major improvements are necessary to solve the software crisis. Software engineering advances happen almost daily, and our foreign competition excels over us in technology insertion. It is not enough that some Requests for Proposals (RFP's) state that an innovative methodology is favorably considered in the contractor selection criteria. DOD-STD-2167 must also encourage innovation.

To quote General George S. Patton, Jr.: "Never tell anyone how to do something. Tell them what needs to be done. They will surprise you with their ingenuity."

The counter argument has been raised that innovation is not for large systems acquired with taxpayer dollars, that the software development process, methods, and life-cycle used should be well understood, generally accepted, and have stood the test of time. Thus, DOD-STD-2167 "incorporates practices which have been demonstrated to be cost-effective from a life-cycle perspective."[2] New methods and life-cycles should first be proven on research and prototype projects.

However, systems grow larger and more complex as time goes by, and software development processes, methods, and life-cycles do not usually scale up. Are we to believe that projects, such as the Strategic Defense Initiative, are best accomplished using a process little changed since the early 1970's?

Standard Obsolescence. By its very nature, most of the DoD (certain advanced R&D efforts excluded) will always be technically several years behind industry, and even further behind the research community. Thus, it is vital that the contractor be encouraged to apply the methods it believes are most appropriate for producing the product and associated documentation that the DoD needs at the least possible cost, while still providing the government adequate oversight into the contractor's development effort.

The current DOD-STD-2167 has been developed using a process that ensures through numerous government and industry reviews that the standard is acceptable to the majority of those who must use it. While very laudable in principle, this consensus nature of DoD standards development is not without its disadvantages when applied to a process standard. Some companies take conservative approaches to software development, while others use methods that are more advanced. Any specific process acceptable to the majority of industry and government must therefore lag significantly behind the state-of-the-art. Thus, the objective of the JLC Software Standardization Program to integrate modern methods of developing software into DOD-STD-2167 is probably both inappropriate and impossible.

Acquisition Process Drivers. The basic process mandated by DOD-STD-2167 and the remaining standards is partially based upon the DoD acquisition process which was historically developed to support hardware and systems acquisition rather than software acquisition. As a consequence, the basic life-cycle and review process is not necessarily consistent with the most modern ways of developing software. This becomes especially important in light of the DoD's recent realization of the prime importance of software development to systems development.

Process Inappropriateness. Because the best software development process, method, and life-cycle are clearly application and implementation language specific, it is surely counterproductive for the DoD to choose any specific ones as either requirements or defaults (and therefore preferred).

The following arguments have been raised in favor of keeping DOD-STD-2167 as a process standard:

1. The lack of a standard software development method complicates the training of government personnel and leads to problems when personnel frequently transfer and are unable to apply their knowledge from the old project to the new. A single, standard process allows the government to train its managers and technicians so that they may work effectively with a contractor on a project. The best contractor-proposed method will not result in better software if the government does not understand it.

These typical advantages of standardization are probably the strongest arguments for keeping DOD-STD-2167 as a process standard. However, any advantages that the government would gain from maintaining "a common, single development process throughout a variety of software development projects"[4] would be outweighed by the inhibitation of innovation that would result.

2. Some people feel that it is not clear that there is yet sufficient justification for allowing the contractor to alter such fundamental concepts such as the DoD acquisition process and

the DoD established software development process, methods, and life-cycle model.

However, the DoD did not introduce the Acquisition Streamlining directives without valid reasons. Innovation is necessary for the continued growth of the defense software industry. Besides, a small number of contractors do this via tailoring now.

3. The contractor may not have the expertise to propose and implement an alternative process, method, or life-cycle.

However, if the contractor does not have such expertise, then the contractor should not be developing software.

4. The government is probably not qualified to evaluate contractor-proposed processes, methods, or life-cycles. Even when contracting agency personnel are so qualified, the evaluation of alternatives would be subject to argument.

However, just as contractor personnel must always keep up with a rapidly evolving technology if they are to remain competitive, government personnel must do so also. One hardly expects government personnel familiar only with vacuum tube technology to manage and maintain modern computer systems, and what applies to hardware applies equally to software. If the government is not qualified to evaluate contractor-proposed processes, methods, or life-cycles, then the government should hire an independent expert. After all, this is one of the standard IV&V contractor's jobs.

5. Because only the classic software development process, methods, and life-cycle mandated by DOD-STD-2167 have been proven to work, any alternate contractor proposed approach involves an unacceptable risk. The government needs to feel comfortable that the project will succeed prior to spending large amounts of money.

However, as has been previously mentioned, it is not at all clear that the classic approach is the only one that has been proven to work. In fact, one can argue that it has often failed to ensure the goals of the DoD. No approach is without risk, and one must wonder how often the risk of trying something new is unacceptable for technical or economical reasons, and how often the reasons are psychological and social.

6. Although the government is comfortable with the software development process mandated in DOD-STD-2167, it will not restrict other methods if the contractor definitely shows during proposal evaluation how the government will gain in cost effectiveness and/or lower risk. All other factors being equal, the contractor who proposes to follow DOD-STD-2167 should lose out to any contractor who proposes a clearly better method.

However nice this is in theory, I am afraid that things are not always the way they should be in practice. Many contracting agency managers are not aware of current trends in software engineering and are unwilling to take what they perceive as unnecessary risks. Even when they look favorably on innovative solutions to their problems, industry managers may well not be willing to propose something other than what DOD-STD-2167 requires or has as a default.

7. It is the contractor's responsibility to ensure that the process used complies with government standards for software development.

One can argue, however, that the contractor has a higher duty to propose the best approach possible. Giving the contracting agency what it expects is not always in the DoD's best interest. The contractor should feel free to use its expertise to propose the best solution to the government's problems.

8. The government does not usually pay industry to develop new methods unless the new methods produce lowered cost, less risk, increased quality, etc.

However, the government should encourage industry to use new methods already developed and not inhibit industry from the development of new ones.

9. The government must define phase boundaries and milestones to be able to manage the software development process.

Although the government must have some phase boundries and milestones if it is to properly manage the software ACQUISITION process, it does not follow that any single specific set of phase boundries and milestones is optimal for all projects. Nor does it follow that the government is best able to define them. It is government's responsibility to manage the software acquisition process and industry's responsibility to manage the software development process.

10. The situation is no different than it was with MIL-STD-483, MIL-STD-490, and MIL-STD-1679.

Though true, this is hardly a justification for keeping a less than perfect status quo.

Specific Problems with DOD-STD-2167 Process

One of the many causes of the software crisis is that the classic software development process mandated by DOD-STD-2167 has not always been successful on large projects. The "proven" process, method, and life-cycle have rarely worked as well as promised and have often stood in the way of innovation.

The following specific problems have been raised with regard to the DOD-STD-2167 process:

Life-Cycle Constraint. Many new developmental life-cycle models have been introduced during the last few years, and others will continually be created as software engineering evolves. Several are fundamentally different from the

classic waterfall life-cycle and can not be mapped into it. Notable examples of methods having life-cycles prohibited by DOD-STD-2167 include certain rapid prototyping methods, AI methods, and recursive development methods such as Object-Oriented Development[6]. As Judah Mogilensky has put it: "The classical 'waterfall' life-cycle is to modern software management as global common data structures is to Ada software design." Thus, requiring conformity to a single standard life-cycle model is an improper "how to" restriction placed on the contractor.

The counter argument that all life-cycle models are minor variations of the classic waterfall life-cycle and are thus permitted within DOD-STD-2167 is simply not true.

Other counter arguments (e.g., that DOD-STD-2167 recognizes that the phase boundries of the classic waterfall life-cycle are not distinct, that considerable overlap of phases is permitted, and that iterative software development is permitted) are true, but do not really address this issue because of the DOD-STD-2167 and MIL-STD-1521 requirements that specific products be developed and reviewed during specific life-cycle phases. While more life-cycle models are consistent with DOD-STD-2167 than with MIL-STD-1679, there are still other models which are prohibited.

Functional Decomposition Constraint. The functional emphasis of the Software Requirements Specification, the functional aspect of certain design entities (e.g., unit) of the static software hierarchy of DOD-STD-2167, and the way this hierarchy is tied to the software development process tends to force the developer into using a functional, hierarchical-decomposition software development method.

Thus, the static software hierarchy is tied too closely with the software development process, prohibiting one from first developing the proper Ada structure and only then decomposing it into a static hierarchy for purposes of Software Configuration Management. The static software hierarchy of DOD-STD-2167 also does not map well into the network structure of well-designed Ada software, and impacts the order and scope of integration and testing. This, however, should be method, language, and software architecture dependent. This problem is another example of improper "how to" constraints on the contractor.

Top-Down Constraint. Paragraph 4.8 of DOD-STD-2167 states: "The contractor shall use a top-down approach to design, code, integrate, and test all CSCI's unless specific alternate methodologies have been proposed ... and received contracting agency approval." This choice of "top-down" as the single default development approach implies that it is the preferred approach for all software development activities. Yet the appropriateness of "top-down," "bottom-up," "outside-in," "inside-out," or "holistic" approaches is language, application, method, and life-cycle dependent. Examples of situations where other approaches may well be preferable include:

(1) Extensive reuse often implies a "bottom-up" approach to design and test.

(2) The compilation order restrictions of Ada encourages a "bottom-up" approach to CSC testing.

(3) The development of critical software implies "bottom-up" design and testing.

(4) The development of test suites requires at least a partial "bottom-up" approach.

This requirement is therefore an improper "how to" restriction on the contractor.

The counter argument that "top-down" is the currently preferred approach of many is irrelevant because a consensus rarely produces state-of-the-art approaches and because any default is subject to obsolescence.

The counter argument that all design methods are top-down is just incorrect. Not only do many other approaches exist, it can be argued that every major project should use a combination of methods.

The counter arguments that most of the bottom-up examples are subject to debate (e.g., there are strong arguments for "top-down" testing) miss the point that other methods exist and the contractor should be free to use the best method for his specific application.

Program Design Language Constraint. Paragraph 5.2.1.4 of DOD-STD-2167 makes the use of a PDL the default method for the top-level design of each CSCI and paragraph 5.3.1.5 mandates the use of a PDL in the development of the detailed design.

But the use of a PDL as a top-level design description method is probably inappropriate. Graphics, such as those of Booch[7] and Buhr[8] are clearly superior in terms of understandability when it comes to presenting top-level architectural designs in terms of software units and their relationships.

The use of a PDL as a detailed design description method is also becoming inappropriate. Due to the high modularity and low complexity of well-designed Ada units, the lack of distinction between Ada PDL and Ada code, and the design aspects of the Ada specification, the nature and purpose of PDL is changing in the Ada community. PDL is not needed to document the logic of the body of many units because of their trivial size and complexity.

The PDL requirement and default probably resulted from findings that the use of PDL's

increased productivity and reduced life-cycle costs compared to the use of flow-charts. They certainly are useful for specifying the internal logic of programs written in low-level languages or resulting from software development methods that produce relatively large and complex unit bodies. PDL, especially when viewed as a program documentation language, may also prove useful in the automatic generation of Software Detailed Design Documents. However, it is inappropriate to imply that the use of PDL's, or any single detailed design method or tool, is to be preferred under all circumstances. This inhibits innovation and is an improper "how to" constraint on the contractor.

One counter argument often heard in government circles is that defaults do not necessarily prejudice the contracting agency. If the customer is truly interested in "graphic methods", then the PDL default should not adversely affect this. However, although defaults may not unduly influence all contracting agency personnel, many developers are convinced that this is a real problem.

Another counter argument is that specifying the use of a PDL does not necessarily specify how it is to be used. Although this is true, it would nevertheless be hard to argue that a graphic method is a PDL.

"Informal" Test Constraints. DOD-STD-2167 currently contains many requirements and defaults regarding the performance and documentation of informal (i.e., contractor-internal) testing. These improper "how to" constraints cover such areas as the documentation of unit-level test requirements, responsibilities, schedules, test cases, procedures, and results in the Software Development Files; the default for individual testing and configuration management; the implication that units are integrated individually into Computer Software Components (CSC's); and the required documentation of considerable information concerning contracter-internal CSC integration and testing in the Software Test Plan. Many developers view these process and documentation requirements as not being cost-effective in many cases and as an unnecessary and unwanted micro-management by the government that formalizes "informal" testing.

Review Process Constraints. The size and complexity of today's systems overwhelm the formal review process of DOD-STD-2167 and MIL-STD-1521. It is not humanly possible to properly perform technical reviews on manually-produced "gothic novel" sized specifications. There is often insufficient time for a proper in-depth analysis and the correction of errors found. The reviews therefore tend to concentrate on superficial formatting problems while important technical issues become buried. The forest gets lost for the trees.

By allowing the contractor to greatly limit the scope of any single review, a better analysis would result for the following reasons:

(a) Smaller documents and partial documents are easier to review. There is less reviewer fatigue and the end of the documents will be reviewed with the same care as the beginning. Currently, the tail end of larger documents often "slide by" due to reviewer fatigue, lack of time, etc.

(b) Because smaller documents and partial documents can be prepared with less lead time, they will be more current when reviewed.

(c) Because smaller documents and partial documents take less time to produce and review and have a more narrow scope, a small percentage of the project's personnel grind to a shorter stop.

(d) Having a larger number of smaller reviews makes each single review less important. By becoming part of the (almost weekly) development activities, the developers are less impacted by "non-productive" work; the "dog and pony show" atmosphere is reduced.

(e) If any "show stoppers" are discovered, they will likely be limited in scope and result in holding up the development process for a shorter period (e.g., corrections can be processed in a recap session).

(f) Spreading out each review permits better contractor man-power leveling by overlapping the requirements analysis, design, and coding of separate elements. The same advantages offered in DOD-STD-2167 now for the separate review of different CSCI's and incremental reviews (e.g., for each build or release) would also result if applied to smaller, relatively independent "chunks" of software (e.g., those resulting from each recursion of the Object-Oriented Development process).

(g) Major process problems will show up earlier when they will be easier and less expensive to correct.

Although DOD-STD-2167 allows incremental reviews, the linear nature of the classical life-cycle with its formal reviews that act as bottlenecks between phases (4.1.2) prohibits the use of recursive "design a little, code a little, test a little" methods. Thus, although DOD-STD-2167 permits a small number of incremental PDR's per CSCI per build or release, it does not permit methods such as Object-Oriented Design in which small amounts of code (e.g., approximately 1KLOC) are recursively designed, coded, and tested during each pass through the method. On large projects (e.g., > 100KLOC), it is clearly impractical to hold several hundred traditional CDRs and PDR's. The bottleneck nature of the formal reviews also prohibits one from coding and testing

as one goes -- an important aspect of such methods that permits one to incrementally validate the evolving design.

Because all methods do not produce the same intermediate products in the same order, the scope of the current reviews is sometimes inappropriate.

The timing and the scope of the results of certain design activities are set by the timing and nature of the formal reviews. This is an improper "how to" constraint on the contractor.

Critical Design Review. The process of DOD-STD-2167 does not account for the evolving nature of the Critical Design Review (CDR).

Due to the high modularity and low complexity of well-designed Ada software, the lack of distinction between Ada PDL and Ada code, and the design aspects of the Ada specification, the classic purpose of the CDR (i.e., to review and approve unit-internal logic prior to coding) is no longer relevant. Coding the Ada specification is a design activity. PDL is not needed to document the logic of the body of many units because of their trivial size and complexity. One should go ahead and code the body once started since it involves little added work and allows one to use the compiler to partially check the results prior to any (semi)formal review. By performing the unit testing immediately, one can also validate the design as one goes.

With the use of the same language for both design and implementation (e.g., Ada), there exists a very real non-trivial problem of defining what is design and what is code. This has a very real impact on determining the scope of the CDR as currently defined.

By requiring at CDR and prior to coding and unit test, a formal review of the "detailed design," one is prohibiting the contractor from using RECURSIVE software development methods that result in the hierarchical top-down design, code, and test of very small amounts of software (e.g., approx. 1K SLOC). This is a very major and improper "how to" constraint on the contractor.

Delay Problems. On large projects, the DOD-STD-2167 process results in a long delay between requirements definition and their implementation. During this time, contracting agency personnel are likely to change, causing the project to be subject to different "hot buttons" and large changes in requirements. Contractor personnel turnover is also likely, resulting in the loss of the rationale for certain key decisions.

Lack of Intermediate Software. Useful software does not exist until the end of the development life-cycle. Thus, it is not until the end of a build, release, or project that one has a product that:

(a) Validates requirements and design
(b) Is testable
(c) Is subject to user scrutiny.

This is a major argument for prototyping and also one of the incentives that has tempted some contractors to rush into coding without a systematic software development method or adequate preparation.

Prototype Inhibition. Although the use of prototypes has long been recognized as an important and productive technique in every engineering field, it is something for which DOD-STD-2167 does not adequately allow. The DOD-STD-2167 development life-cycle is also inconsistent with that of several prototyping models. Only by building prototypes, in addition to producing a "paper" design, can one verify the feasibility of the design. And this is perhaps one of the reasons why hardware engineering has advanced beyond software engineering.

Reuse Inhibition. The top-down development process mandated by DOD-STD-2167 seems based on the tacit assumption that all software will be built from scratch. This thwarts one of the major goals of Ada use, namely the production and use of libraries of reusable software. Isolated references to the importance of reusability in DOD-STD-2167 are insufficient if the general process inhibits it.

Automation Inhibition. To increase the efficiency of the software development process and to increase the quality of the resulting software and documentation by reducing human error, significant portions of the process need to be automated. This includes, but is certainly NOT limited to, the production of documentation. DOD-STD-2167 seems based on the tacit assumption that all software is to be built from scratch by performing all activities of the prescribed process in accordance with the standard life-cycle. Yet when significant portions of the life-cycle are automated, this will have a major effect on the description of these required activities and the associated reviews. It is not at all clear that this can be adequately or efficiently handled by merely deleting requirements from DOD-STD-2167, the only method of tailoring allowed.

A counter argument to part of the automation problem is that the government will only want to pay for automation if it can be proven to be cost-effective. Automation in and of itself is not necessarily good. In judging the cost-effectiveness of something, trade-offs should be performed. Does the readibility of the code or document suffer from the automation so that reviewing and maintenance takes longer and costs more? If such questions are ignored until too late, the project will suffer.

While the above counter argument raises valid questions, they must be weighed against the improved quality and productivity that are the goals of automation. The manual production of anything opens the doors to human error. Much of documentation consists of translating design information from one format (e.g., that of the working documentation or software structure) into another (i.e., the required format of the deliverable documentation). When this translation

is performed by hand, things tend to fall through the cracks and transcription errors occur resulting in an inconsistency between the software and its documentation. When documentation is manually produced, it is not always updated to incorporate changes in design due to the large amount of effort involved. Much the same can be said about the process of "translating" requirements into design and "translating" design into software. And as for making the reviews take longer and cost more, one can argue that because automation should entirely eliminate certain classes of errors, the reviews should be more productive since reviewers will not need to spend time finding such errors.

The following arguments for the current DOD-STD-2167 process have been raised:

1. One important feature of the DOD-STD-2167 process is that it ensures the production of intermediate products that can be used to restart the process should development be stopped mid-stream.

However, the DOD-STD-2167 process is not the only one that generates intermediate products. All contractor-proposed processes should.

2. The classic waterfall life-cycle of DOD-STD-2167, with its phase boundaries and milestones, brings a structured management process to software development.

However, the classic waterfall life-cycle is not the only life-cycle that is structured or that defines phase boundaries and the milestones needed to manage software development.

Recommended Solutions

Before looking at specific recommendations, however, it is valuable to first consider the needs of the DoD that formed the foundation of the DOD-STD-2167 process. The DoD needs to:

1. Understand the process, method, and life-cycle used to develop the deliverable software.

2. Be confident that the process, method, and life-cycle are cost-effective and will result in a product that is delivered on time and within budget.

3. Exercise proper oversight of the development process (e.g., via reviews of intermediate products) so that it can be reasonably certain that there will be no major surprises.

4. Be confident that the delivered software will work as expected and be maintainable.

To solve the preceding problems while ensuring that the needs of the DoD are met, I recommend that the DoD:

1. Rename DOD-STD-2167 from "Defense System Software Development" to "Defense System Software Acquisition."

2. While keeping the DID's, modify DOD-STD-2167 from a development process standard into an acquisition process standard.

3. Provide industry with financial incentives to do good work.

4. Modify the current acquisition process to account for the differing needs of software and hardware developers.

And if the DoD considers my recommendations to be too radical, I propose that DOD-STD-2167 at least be modified to make it clearly independent of any required or default software development process, method, or life-cycle. This could be accomplished by:

1. Replacing paragraph 4.7 of DOD-STD-2167 with the following:

"The contractor shall propose in the SDP and detail in the SSPM a systematic software development process, methods and tools that are appropriate for the application and implementation language. Once approved by the contracting agency, the contractor shall develop all CSCI's in accordance with these methods and tools."

2. Remove all mention of method-dependent terms such as PDL and top-down.

3. Deleting paragraphs 4.1.1 and 4.1.2 of DOD-STD-2167 and replacing paragraph 4.1 with the following:

"The contractor shall propose in the SDP a specific, structured, life-cycle model with well-defined phases that provides adequate intermediate products and milestones that is appropriate for the application, implementation language, and proposed software development process, methods, and tools. Once approved by the contracting agency, the contractor shall develop all CSCIs in accordance with this life-cycle."

4. Remove all requirements governing contractor internal processes (e.g., informal testing, controls and visibility requirements concerning non-deliverable items, the use of software development libraries, development configurations).

5. Decoupling DOD-STD-2167 from the reviews mandated by MIL-STD-1521, and adding the following paragraph to DOD-STD-2167:

"The contractor shall propose in the SDP formal and informal reviews based on the phases of the software development life-cycle model. Once approved by the contracting agency, all CSCI's shall be reviewed in accordance with these proposed reviews."

6. Decoupling the static software hierarchy from the software development and review process so that the contractor may first develop the proper software structure and then decompose it into a static hierarchy for purposes of SCM.

7. Adding criteria to DOD-HDBK-287 for evaluating the contractor's proposed software development process, methods, life-cycle, and reviews.

Conclusion

DOD-STD-2167 is a Tri-services military standard that establishes and mandates a uniform software development process for the Mission-Critical Computer Resource software. It will therefore be applied to a great many projects and have a major impact on the way software is developed in the United States.

The draft of DOD-STD-2167A contains several improvements that begin to answer some of the problems mentioned above. As part of the review cycle for this draft revision, members of the Ada community identified numerous problems concerning the compatibility of the DOD-STD-2167 with the proper use of Ada and modern software development methods. Some of these problems are specific to the DOD-STD-2167 process, methods, and life-cycle, while others are problems with software process standards in general.

It is the professional opinion of this author that the goals of the DoD would be best served if all process, method, and life-cycle requirements and defaults were deleted from DOD-STD-2167. In accordance with its own policy, the DoD should specify WHAT it needs and leave the contractor free to propose the best application and implementation language-specific way it should be developed. Only by promoting innovation and rewarding achievement will the DoD ensure the use of the best process, methods, and life-cycle for the application and implementation language.

References

1 SDSAWG, "Issues and Sub-issues Report," October 86

2 DOD-STD-2167, "Defense System Software Development," Military Standard, 06/04/85

3 DOD-STD-2167A, "Defense System Software Development," Military Standard, 08/15/86.

4 Firesmith, D.G. and Capt. Gilyeat, C., "Resolution of Ada-Related Concerns in DOD-STD-2167, Revision A," Ada Letters, July-August, 1986

5 DODD 5000.43, "Acquisition Streamlining," DoD Directive, 1985

6 Firesmith, D. G., "Object-Oriented Development," presented at the SIGAda Conference (2/25/86), the National Conference on Software Methodologies (3/11/86), and the Ada and the Space Station Conference (6/3/86).

7 Booch, Grady, SOFTWARE ENGINEERING WITH ADA, U.S. Air Force Academy, Benjamin/Cummings, 1983

8 Buhr, R.J.A., SYSTEM DESIGN WITH ADA, Englewood Cliffs, NJ, Prentice-Hall, 1984

About the Author

As Software Methodologist for the Tactical Systems Division of Magnavox Electronic Systems Company, Mr. Firesmith supports the Advanced Field Artilery Tactical Data System (AFATDS) Project, the first major DoD program (ca. 770K SLOC) to use Object-Oriented Development (OOD) and Ada as implementation language. He is responsible for the development and maintenance of Magnavox's OOD methodology. He is Chairman of the ACM SIGAda Software Development Standards and Ada Working Group (SDSAWG) and is a member of the CODSIA Software Development Standards Task Group. Mr. Firesmith has worked in software development, quality assurance, and configuration management. He may be reached at Magnavox Electronic Systems Company, Dept. 566, 1313 Production Road, Fort Wayne, IN 46808, or by telephone at (219) 429-4327.