Planning for Object–Oriented Military Projects

Paper

Donald G. Firesmith
Advanced Software Technology Specialists (ASTS), Ossian, IN 46777–9406 USA
Object Technology, Software Consulting Specialists (SCS), Fort Wayne, IN 46898, USA

Abstract

The purpose of this paper is to provide managers with an overview of the information required to plan military projects using an object–oriented development (OOD) method. This paper begins by discussing the impact of OOD on project planning (regardless of whether the project is military or not). This discussion includes a general set of steps that should be taken on initial OOD projects. These steps in turn result in the development of a project–specific software development plan (SDP). This paper also discusses some of the actual and perceived inconsistencies between OOD and military standards and expectations. This paper then provides general guidance for planning object–oriented military projects. Lastly, this paper provides both an overview of the general planning requirements of DOD–STD2167A and detailed tailoring and interpretation guidance for developing the Software Development Plan (SDP) required by DOD–STD–2167A.

Keywords

Documentation, DOD–STD–2167A, Object, Object–Oriented Development, Planning, Software Development Plan

1. Introduction

Planning is critical to the success of any non–trivial software development process. **Those who fail to plan, plan to fail.** This is especially true whenever a technology transition occurs. Much of the common sense that management uses to plan a project is based on past experiences gained when working on traditional projects using older methods. OOD involves a new paradigm of software development that has major impacts on the way software developers analyze, design, code, test, and document their software. This, in turn, has many impacts on the way such projects should be planned and managed. This paper discusses these impacts on project planning that result from the transition from the traditional development paradigm (e.g., involving functional decomposition and the waterfall development cycle) to the object–oriented paradigm (e.g., involving objects, classes, and both iterative and recursive development cycles).

2. The Impact of OOD on Project Planning

There are numerous object–oriented software development methods including, but not limited to, [Booch 1991], [Coad and Yourdon 1989], [Colbert 1989], [Firesmith 1992], [Rumbaugh et al. 1991], and [Shlaer and Mellor 1988]. Because these different methods use slightly different models, graphics, and development cycles, the impact on project

planning may vary from method to method.

These methods have, however, a great many similarities that differentiate them from the older methods. First of all, object–oriented methods, as the name implies, are based upon the concept of an object, something that did not exist in older methods. An object is an abstraction or model of an application domain entity that localizes and encapsulates attributes (i.e., data), operations (i.e., functions), exceptions, and component objects. Objects are not data and are more than the sum of data and related operations. Similar or identical objects are instantiated from (i.e., created by) classes, whereby classes are templates that exist in inheritance hierarchies of superclasses and subclasses. Classes, inheritance, and the related concepts of polymorphism and dynamic binding are new to the majority in the military (and civilian industry). Object–oriented development methods use new and modified models (e.g., object, class, assembly, control) and graphics (e.g., semantic nets, interaction diagrams, classification diagrams).

Most object–oriented methods employ an incremental, iterative, and recursive[1] development process whose activities are different and exhibit massive overlap. Software engineers identify objects and classes and develop various object–oriented models. The concepts of preliminary and detailed design, if they exist at all, often have radically new meanings unrelated to the design of computer software components (CSCs) and computer software units (CSUs). Object–oriented methods often incrementally develop each assembly (e.g., computer software configuration item or CSCI), a subassembly (e.g., CSC) at a time, using an "Analyze a little, design a little, code a little, test a little" recursive development cycle. The concept of a single Preliminary Design Review (PDR) separating the Preliminary Design Phase from the Detailed Design Phase (a la MIL–STD–1521B) is utterly foreign to many OOD methods that instead rely heavily upon iteration and recursion and may instead employ In–Process Reviews (IPRs). In fact, many object–oriented methods use the same models and graphics for both software requirements analysis and design, and the distinction between the two is blurred, especially if significant iteration is employed. Thus deliverable documents, such as software requirements specifications (SRSs) and software design documents (SDDs), are often developed simultaneously during overlapping activities rather than during the corresponding disjoint phases of the traditional development cycles.

Figure 1 (Preparing and Planning for OOD) documents the most important planning tasks to be performed on initial OOD projects and the temporal dependency relationships between them. Because of the typical lack of experience and training in OOD, management's first task should usually be to obtain expert consulting and initial training in OOD and OOD management principles. Commitment from upper management is also critical in obtaining the necessary resources to make the transition to OOD work. Care should be taken in choosing the correct project(s) to use as a vehicle for transitioning to object technology so as to minimize risk and ensure adequate resources and visibility. A risk management plan should be developed dealing with the risks of such a technology transition. Because of the critical importance of training in transitioning staff to the new technology, a training plan should also be developed. The results of these in turn should be used to modify management's expectations (e.g., waterfall development cycle) and to obtain the necessary resources for 1) further training, 2) the evaluation. selection, and tailoring of models, methods. notation, languages, compilers. CASE tools, and objectbases[2], 3) the development of the associated standards and procedures, and 4) the development of the reuse

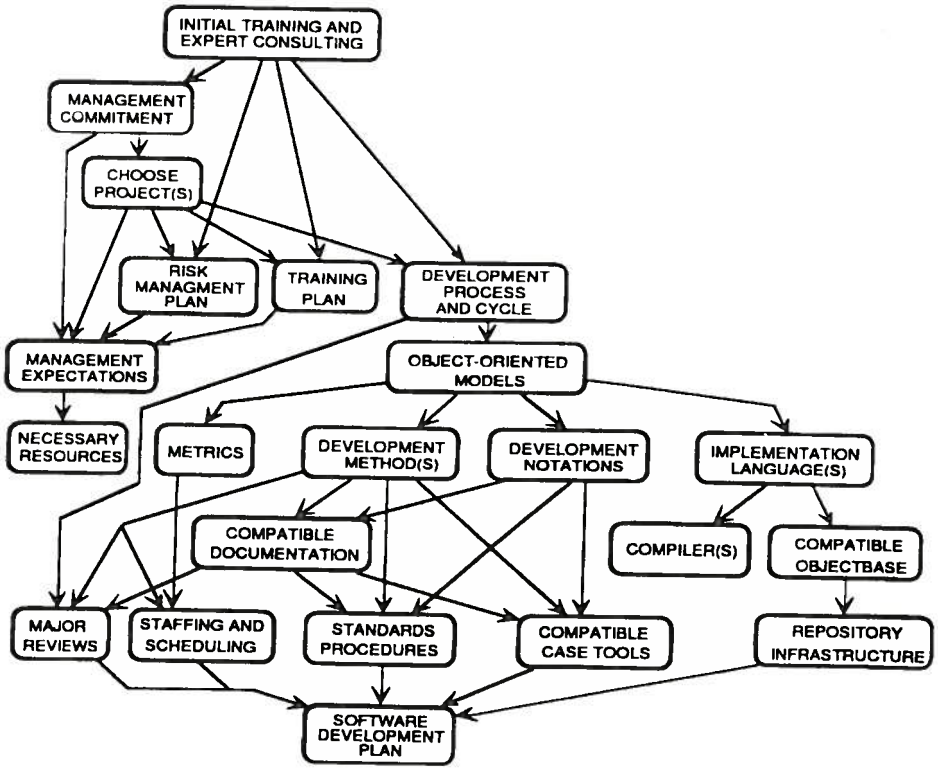

Figure 1: Preparing and Planning for OOD

repository infrastructure required to benefit from the increased reuse delivered by OOD. Planning should ensure that the development process and cycle, models, methods, notations, languages. and CASE tools are consistent and appropriate for OOD.

3. Inconsistencies Between OOD and the Military Standards and Expectations

Numerous inconsistencies, both real and perceived, exist between OOD and current military standards and expectations. The military and object–oriented paradigms are based on different concepts, models, hierarchies, documentation formats, development cycles, and formal reviews. Some of these inconsistencies have critical contractual implications and require early tailoring. Other inconsistencies are merely perceived (e.g., that DOD–STD 2167A mandates the waterfall development cycle[3]), represent misunderstandings, or result from human inertia. These inconsistencies are best treated with early and adequate training.

The military considers software to consist of CSCIs that are decomposed into CSCs which are further decomposed into lower–level CSCs[4] and CSUs[5]. The military also requires software requirements to be decomposed into capabilities and designs to be decomposed into data (e.g., input/output data elements, local data elements, data structures, local data files or database) and operations (e.g., algorithms, error handling, data conversion operations, logic flow). The Software Design Document (SDD) is organized by CSCs (which are designed during the Preliminary Design activity) and CSUs (which are designed during the Detailed Design activity). Object–oriented methods consider software to consist of subassemblies (i.e., clusters, subjects, "subsystems") of objects and classes which primarily encapsulate attributes and operations. A successful mapping between the two sets of concepts is necessary in order to docu-

ment the object–oriented entities according to the military standards.

The military is primarily accustomed to older, functional–decomposition versions of methods such as Structured Analysis [DeMarco 1978], Structured Design [Yourdon and Constantine 1979], and Structured Development for Real–Time Systems [Ward and Mellor 1985]. Many military reviewers thus expect to see software engineering models based upon such graphics as Data Flow Diagrams (DFDs), State Transition Diagrams (STDs), and Structure Charts. Object–oriented methods use other models. Object Models consist of Semantic Nets, Interaction Diagrams, Composition Diagrams, and the specification and body of all objects. Class models consist of Classification Diagrams and the specification and body of all classes. State Models are similar, although the State Transition Diagrams are restricted to the state of individual objects or classes. Control Models use object–oriented Control Flow Diagrams (CFDs) and operation bodies rather than functional decomposition Data Flow Diagrams. Timing Models consist of Timing Diagrams and/or STDs and CFDs annotated with temporal information.

The military software organization is an aggregation hierarchy based upon the hierarchical decomposition of CSCIs into CSCs into CSUs. Object–oriented methods use aggregation, dependency, inheritance, and message hierarchies based upon objects classes, and their important relationships.

The military expects, and sometimes requires, specific documentation formats (e.g., the SDP Data Item Description (DID)) without proper tailoring. Software engineers and methodologists naturally prefer appropriate document formats organized around subassemblies, objects and classes, their important relationships, and their attributes and operations. At best, these two formats are indirectly related because of the difference in fundamen-

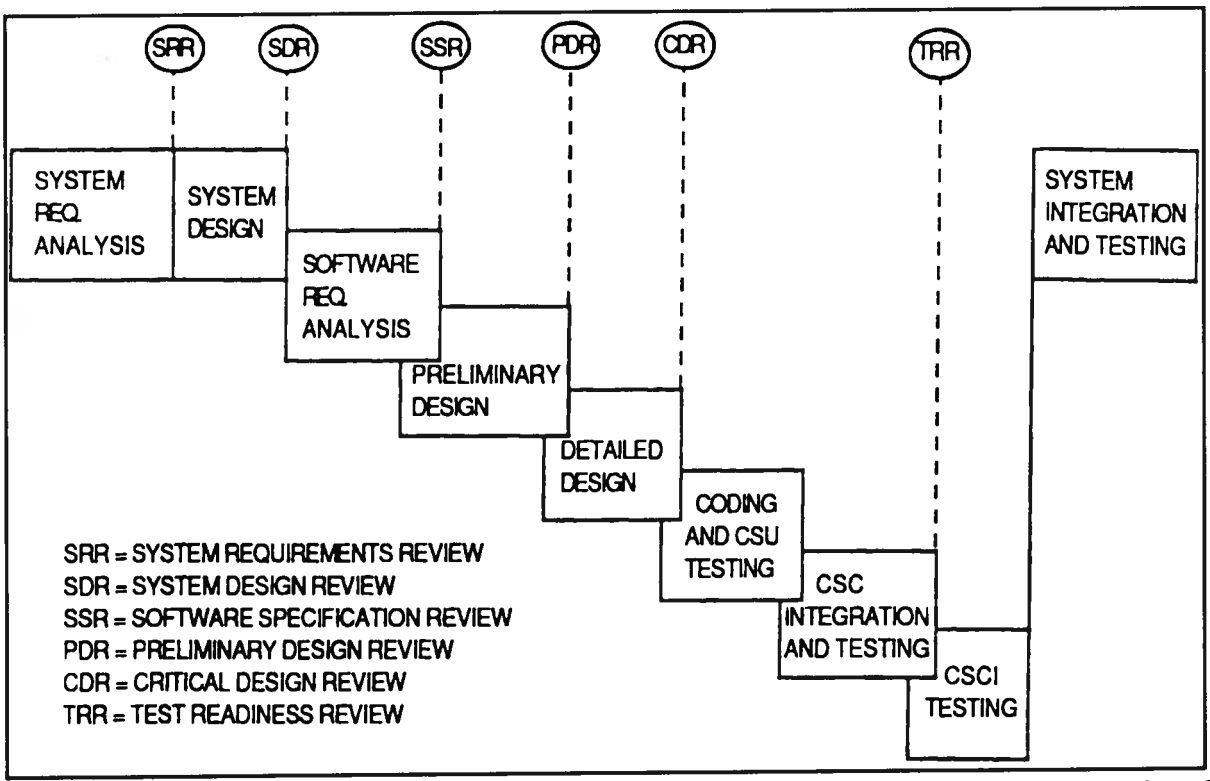

Figure 2: DOD–STD–2167A's Example Waterfall Development Cycle

tal concepts and require proper mapping (e.g., from CSU to object and class). At worst, the two formats are fundamentally incompatible and require significant tailoring or replacement. For example, the DOD–STD–2167A Software Requirements Specification (SS) and Software Design Document (SDD) DIDs still emphasize the distinction between, and separate the documentation of, data and functions on that data. On the other hand attributes (i.e., data) and operations (i.e., functions) are combined and encapsulated under an object–oriented paradigm.

In spite of significant changes to the military standards. the software design document and MIL–STD–l 521B still partially mandate (and many military personnel expect) the use of the obsolete "waterfall" development cycle. However, most object–oriented development methods use an incremental iterative and recursive development cycle that is topologically inconsistent with the waterfall development cycle. This has profound impact on formal reviews, which tend to be In–Process Reviews (IPRs) rather than traditional Software Specification Reviews (SSRs), Preliminary Design Reviews (PDRs). and Critical Design Reviews (CDRs). See Figure 2: DOD–STD–2167A's Example Waterfall Development Cycle.

4. General Recommendations for Planning Military Projects

First of all, software engineers on military projects should learn the documentation requirements and expectations of DOD–STD–2167A and MIL–STD–1521B. Software engineers on projects using OOD should learn how to use OOD effectively to properly document requirements and designs in terms of objects and classes, their resources (e.g., attributes, operations, exceptions, component objects), their relationships, and the inheritance relationships between the classes. In accordance with professional ethics and current military policy, contractors should propose the best documentation solution for object–oriented specifications and designs. Software engineers on military projects using OOD should understand the inconsistencies between 1) the traditional military project requirements and expectations and 2) the methods for producing object–oriented specifications and designs. Contractor technical experts should develop or acquire standards for OOD–method–specific DID formats and contents. They should also provide a requirements trace to ensure that all relevant requirements of the military DIDs are met. Contractor technical experts should map the OOD entities to the DOD–STD–2167A software organization entities (e.g., assembly or "system" => CSCI; subassembly, cluster, subject, or 'subsystem" => CSC, object and class => CSU). Software engineers, technical managers, and proposal staffs should work with and educate their contracting agencies to ensure that the standard and DIDs are properly tailored and interpreted. They may use military policy, the recommendations of experts, and organizations like the SIGAda Software Development Standards and Ada Working Group to provide objective rationale and guidance. Software engineers should also ensure that the OOD experts they choose have experience with military policy and standards. All parties should understand that DOD–STD–2167A is intended to be method– and paradigm–independent. While some requirements inadvertently imply a partial waterfall development cycle, the body of the standard does not. Finally, contractors should evaluate CASE tools carefully to ensure that they properly support the project–specific OOD method (perhaps via customization) and that they "automatically" generate deliverable documentation that both properly documents object–oriented requirements and designs while meeting military documentation standards.

5. General Planning Requirements of DOD-STD-2167A

The military standard Defense System Software Development, or DOD-STD-2167A as it is better known, establishes uniform requirements for software development that are applicable during the acquisition, development, and support of software systems throughout the system life cycle. Although DOD-STD-2167A is primarily used to control, and provide insight into, a contractor's software development, testing, and evaluation efforts, it is often applied to government agencies that perform software development or support. DOD-STD-2167A was approved for use by all departments and agencies of the US Department of Defense and is also a de facto international standard used by the military and non-military governmental agencies in many countries.

According to DOD-STD-2167A, the "contractor shall develop plans for conducting the activities required by this standard. These plans shall be documented in a Software Development Plan (SDP). Following contracting agency approval of the SDP, the contractor shall conduct the activities required by this standard in accordance with the SDP. With the exception of the scheduling information, updates to the SDP shall be subject to contracting agency approval."

DOD-STD-2167A was developed to be applicable to all projects from the very small and simple to the very large and complex. Its requirements were therefore designed to be complete and tailored (by deletion of non-applicable requirements) for each contract by the contracting agency.

The Software Development Plan (SDP) describes a contractor's plans for conducting software development. The SDP is used to provide the government insight into the organization(s) responsible for performing software development and the methods and procedures to be followed by these organizations. The SDP is also used to monitor the procedures, management, and contract work effort of the organizations performing software development.

During systems requirements analysis and design, DOD-STD-2167A requires the contractor to perform an evaluation of the SDP and place it under configuration control prior to delivery to the contracting agency. This evaluation includes checking for internal consistency, understandability, and traceability to the statement of work (SOW) and contract data requirements list (CDRL). Because the contractor must perform the required activities in accordance with the SDP, the SDP should be considered a living document that is updated as the contractor's knowledge of OOD increases during the project. This is especially important for the contractor's first few object-oriented projects.

Content and format requirements for the Software Development Plan are documented in DI-MCCR-80030A, the associated DID. Although OOD does not significantly impact every section and paragraph of the SDP, important changes should be made to ensure that OOD's impacts are adequately taken into account and that requirements of the SDP DID do not adversely impact the use of OOD. Some SDP requirements need to be tailored out of the DID. Because the developers of DOD-STD-2167A could not anticipate everything, the SOW should modify some SDP requirements and add others. Finally, some requirements of the DID require reinterpretation. Section 6 of this paper will provide specific tailoring and interpretation guidelines.

The intent of DOD-STD-2167A is to be language- and method-independent. The SDP must therefore document the contractor's plans concerning the development method to be used in the development of the software (e.g., ADM3 [Firesmith 1992], OMT

[Rumbaugh et al. 1991], OOSD [Colbert 1989], and Structured Design [Yourdon and Constantine 1979]). The SDP must provide the contracting agency with sufficient detail to properly evaluate proposed methods that may be radically different.

Although DOD–STD–2167A states that the software shall be developed in accordance with the SDP, contractors often supply incomplete SDPs during the initial activities of the projects. Because these initial SDPs often do not adequately document the contractor's method(s) and associated standards and procedures, contractor personnel often waste valuable project time trying to decide how to do their work when they should instead be performing analysis and design. A well documented and complete SDP should therefore be delivered as part of the proposal and used as part of the technical evaluation of the bidders. The contracting agency should explicitly specify this in the SOW and CDRL.

Because a draft SDP is often required as part of the proposal, the contractor must understand and document the software development method earlier and in more detail than is traditional. Traditional boilerplate SDPs, that merely parrot back the requirements of DOD–STD–2167, are no longer adequate.

6. Detailed Tailoring and Interpretation Guidelines

The following paragraphs and subparagraphs of the untailored DOD–STD–2167A SDP and SDP DID (DI–MCCR–80030A) are impacted by the use of an OOD method and require method– or OOD–specific tailoring, modification, addition, or reinterpretation:

SDP	SDP DID	Current Paragraph Title
3.2.1	10.2.5.2.1	Activities
3.2.2	10.2.5.2.2	Activity network
3.3	10.2.5.3	Risk Management
3.8	10.2.5.8	Formal reviews
4.1.2	10.2.6.1.2	Personnel–software engineering
4.1.3	10.2.6.1.3	Software engineering environment
4.1.3.1	10.2.6.1.3.1	Software items
4.2	10.2.6.2	Software standards **and procedures**
4.2.1	10.2.6.2.1	Software development techniques and methodologies
4.2.2	10.2.6.2.2	Software development files
4.2.3	10.2.6.2.3	Design standards
4.2.4	10.2.6.2.4	Coding standards
4.3	10.2.6.3	Non–developmental software
6.5.X	10.2.8.5.1	Software products evaluation – (activity name)
8	10.2.10	Other software development functions
8.X	10.2.10.1	(Function name)

6.1. *Activities*

The DOD–STD–21 67A SDP DID currently states:

"10.2.5.2.1 Activities This subparagraph shall be numbered 3.2.1 and shall briefly describe each software development activity of the project and its associated schedule, based on the contract master schedule (if applicable). The development schedule shall indicate all significant events, such as reviews, audits, key meetings, etc. The schedule may be provided graphically. For each activity, the schedule shall indicate:

a. Activity initiation
b. Availability of draft and final copies of formal and informal documentation
c. Activity completion
d. Areas of high risk."

This SDP DID paragraph does not require tailoring, but it does require interpretation. The corresponding SDP paragraph should:

- summarize the project–tailored version of OOD development process and development cycle on a DOD–STD–2167A activity by activity basis.

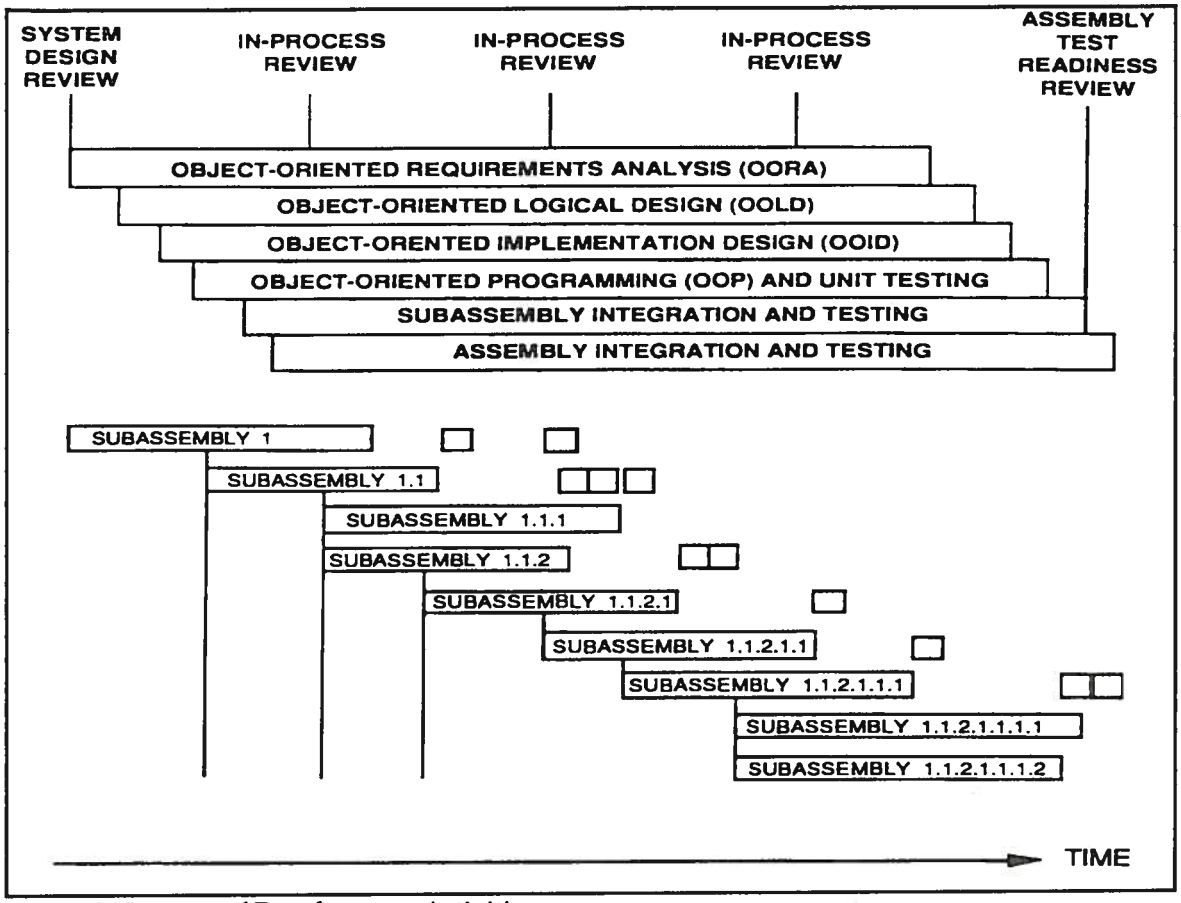

Figure 3: Overlap of Development Activities

- make it clear that the DOD–STD–21 67A activities greatly overlap and are performed both iteratively and recursively.
- summarize the method–specific relationship between the activities and the formal (in–process) reviews, including how those reviews are scheduled in accordance with the project OOD process, development cycle, and method.
- not attempt to schedule the development of individual CSCs (subassemblies) or CSUs (classes or objects), but rather remain at the assembly (e.g., CSCI) and build level. This is because the software is incrementally analyzed, designed, coded, and tested on a subassembly by subassembly basis if a recursive development cycle is used.
- discuss the relationship between the incremental development of the:
  - Assemblies and subassemblies.
  - Software development folders (SDFs), software requirements specifications (SRSs), and software design documents (SDDs) development.
- list the high risk aspects of assembly and build scheduling.

6.2. *Activity Network*

The DOD–STD–2167A SDP DID currently states:

"10.2.5.2.2 Activity network This subparagraph shall be numbered 3.2.2 and shall describe the sequential relationship among the activities of the project. This subparagraph shall include identification of those activities that impose the greatest time restrictions on project completion and those activities with an

opment method(s), the method–specific development cycle, and the corresponding formal reviews. This paragraph shall describe the contractor's plans for the formal reviews including the method–specific: number of reviews, type of reviews,products to be reviewed, and the contractor's proposed acceptance criteria for passing the reviews."

6.5. Personnel–Software Engineering

The DOD–STD–2167A SDP DID currently states:

"10.2.6.1.2 Personnel–software engineering This subparagraph shall be numbered 4.1.2 and shall describe the number and skill levels of personnel who will perform the software engineering activities. The personnel shall be described by title and minimum qualifications for the position. In addition, this subparagraph shall specify any requirements unique to particular positions, such as geographic location, security level, extended hours, etc."

This SDP DID paragraph does not require tailoring, but it does requires interpretation.

The corresponding SDP paragraph should document:

- the OOD–specific qualifications (e.g., software engineering, OOD, and Ada experience or training) required of the developers.
- whether the contractor plans to hire at least a minimum of one or two developers with prior OOD experience.
- whether the project methodologist (or OOD expert) is properly experienced with the project–specific OOD method.
- the planned type and amount of training and schedule of training for each class of personnel.

6.6. Software Engineering Environment – Software Items

The DOD–STD–2167A SDP DID currently states:

"10.2.6.1.3 Software engineering environment This subparagraph shall be numbered 4.1.3 and shall be divided into the following subparagraphs to identify and describe the plans for establishing and maintaining the resources (software, firmware, and hardware) necessary to perform the software engineering activities."

"10.2.6.1.3.1 Software items This subparagraph shall be numbered 4.1.3.1 and shall identify the software items, such as operating systems, compilers, code auditors, dynamic path analyzers, test drivers, preprocessors, test data generators, postprocessors, etc., necessary to perform the software engineering activities. This subparagraph shall describe the purpose of each item and shall identify any classified processing or security issues associated with the software items."

This SDP DID paragraph does not require tailoring, but it does requires interpretation.

The corresponding SDP paragraph should document plans concerning the evaluation, selection, acquisition, and training concerning the following tools:

- method–specific upper CASE tools (e.g., ObjectMaker) for analysis and design.
- class browsers.
- debuggers that understand the impacts of inheritance (e.g., that the definition of an object may be scattered among several classes in an inheritance hierarchy).
- automatic code generators for object–based (e.g., Ada) and object–oriented (e.g., C++) languages.
- automatic document generation tools that understand both:
  - DOD–STD–2167A DID requirements.
  - The proper content and format of documentation of object–oriented software.

excess of time for completion. This information may be provided graphically."

This SDP DID subparagraph requires both modification and interpretation. The corresponding SDP paragraph should:

- take into account that the OOD development activities greatly overlap, are applied recursively, and therefore do not occur sequentially. See Figure 3: Overlap of Development Activities.
- remain at the assembly (e.g. CSCI) and build level.
- be tailored to replace the word "sequential" with "temporal".

Project management CASE tools that are used to graphically schedule projects do not typically handle recursive development approaches well and may therefore not be useful for developing the information required by this subparagraph.

6.3. *Risk Management*

The DOD–STD–2167A SDP DID currently states:

"10.2.5.3 Risk management This paragraph shall be numbered 3.3 and shall describe the contractor's procedures for managing areas of risk to successful project completion. This paragraph shall:

a. Identify the areas of risk to successful project completion and prioritize them.

b. Identify the constituent risk factors that contribute to the potential occurrence of each risk.

c. Document procedures for monitoring the risk factors and for reducing the potential occurrence of each risk.

d. Identify contingency procedures for each area of risk, as appropriate."

This SDP DID paragraph does not require tailoring, but it does require interpretation. The corresponding SDP paragraph should identify and prioritize OOD–specific risks such as lack of or inadequate:

- developer experience and training with OOD.
- developer experience and training in the project–tailored version of the OOD method.
- production–quality CASE tools to support OOD.
- proper tailoring of DOD–STD–2167A, the DIDs, MIL–STD–1521B, etc.

6.4. *Formal Reviews*

The DOD–STD–2167A SDP DID currently states:

"10.2.5.8 Formal reviews This paragraph shall be numbered 3.8 and shall describe the contractor's internal procedures for preparing for and conducting formal reviews."

Different software development methods produce different intermediate products at different times according to different development cycles. This is especially true of globally recursive OOD methods such as ADM3 [Firesmith 1992]. The corresponding SDP paragraph should therefore:

- document the impact of the project–specific OOD method on the formal reviews.
- document whether the reviews are to be:
  - held incrementally and simultaneously or
  - are renamed and redefined.
- list the planned number and type of formal reviews.
- describe the products to be reviewed.
- document the contractor's proposed acceptance criteria.

The preceding paragraph implies that this SDP DID paragraph requires the following requirements to be added6:

"This paragraph shall describe the relationship between the contractor's software devel-

• configuration management tools (e.g., the Rational R–1000 development environment's support for "subsystems").

• requirements tracing tools that understand:

– The impact of incremental, recursive development cycles.

– The building blocks of OOD (e.g., objects, classes, attributes, operations).

• stub generators (some OOD methods are top–down, whereas Ada has bottom–up compilation order restrictions).

6.7. *Software Standards, Procedures, Development Techniques, and Methodologies*

The DOD–STD–2167A SDP DID currently states:

"10.2.6.2 Software standards and procedures This paragraph shall be numbered 4.2 and shall be divided into the following subparagraphs to describe the software standards and procedures the contractor plans to use."

"10.2.6.2.1 Software development techniques and methodologies This subparagraph shall be numbered 4.2.1 and shall identify and describe the techniques and methodologies the contractor plans to use to perform:

a. Software Requirements Analysis
b. Preliminary Design
c. Detailed Design
d. Coding and CSU Testing
e. CSC Integration and Testing f
f. CSCI Testing."

"10.2.6.2.3 Design standards This subparagraph shall be numbered 4.2.3 and shall describe the design standards the contractor plans to use in developing the software."

"10.2.6.2.4 Coding standards This subparagraph shall be numbered 4.2.4 and shall describe the coding standards the contractor plans to use in developing the software."

The current structure of this part of the SDP DID is illogical, incomplete, and inconsistent. Although the development process drives the development method(s) and the method(s) drive the standards and procedures, methods are currently documented as a subparagraph under standards and procedures. DOD–STD–2167A should require standards for requirements analysis, testing, and integration in addition to standards for design and coding. Because Ada is both a design and implementation language and because the OOD methods used in the Ada Community are highly Ada–oriented, it is practically impossible to develop a credible Ada design standard that does not contain coding standards or Ada coding standards containing no design standards. The SDP DID does not clearly require procedures at all! The terminology of DOD–STD–2167A is methods, not techniques and methodologies. The SDP should map well to the development process used, and many OOD methods have different and more detailed activities than DOD–STD–2167A.

These four SDP DID paragraphs require both significant modification and interpretation. Specifically, these DID paragraphs should be replaced with the following single paragraph:

"10.2.6.2 Software development process. methods. standards. and procedures This paragraph shall be numbered 4.2 and shall identify and describe the contractor's planned software development process and associated development cycle. For each activity of the contractor's planned development process, this paragraph shall identify and document the contractor's planned software development method(s) including the associated standards and procedures."

This corresponding paragraph of the SDP should contain:

- analysis and design standards for:
  - the object–oriented graphics (e.g., semantic nets, control flow diagrams).
  - the subassemblies.
  - the objects and classes.
  - the corresponding program design language (PDL) and code.
- coding standards for implementing the objects and classes in Ada.

6.8. Software Development Files

The DOD–STD–2167A SDP DID currently states:

"10.2.6.2.2 Software development files This subparagraph shall be numbered 4.2.2 and shall define the contractor's plans, including the responsible organization(s), for the creation and maintenance of software development files (SDFs). This subparagraph shall define the format and contents of the SDFs and describe the procedures for maintaining SDFs."

Each OOD method produces (slightly) different intermediate products according to (slightly) different development cycles. The SDFs should contain the working products (e.g., graphics, PDL, code, requirements traces, etc.) specific to the project–OOD method. SDFs should be created for each subassembly (CSC) and not for each object–class (CSU). This DID subparagraph also should be moved to be consistent with the rewriting and reorganization of its surrounding paragraphs.

This SDP DID paragraph therefore requires the following modifications to be made via the project statement of work (SOW):

"10.2.6.2.1 Software development files This subparagraph shall be numbered 4.2.1 and shall define the contractor's plans, including the responsible organization(s), for the creation and maintenance of software development files (SDFs). This subparagraph shall define the method–specific format and contents of the SDFs and describe the procedures for maintaining SDFs."

6.9. Non–Developmental Software (NDI)

The DOD–STD–2167A SDP DID currently states:

"10.2.6.3 Non–developmental software This paragraph shall be numbered 4.3 and shall identify and describe each non–developmental software item, such as commercially available, reusable, and Government furnished software, to be incorporated into the deliverable software. This subparagraph shall briefly describe the rationale for the use of each non–developmental software item."

Reusable software is a central tenet of any OOD method, and this is especially true of methods (e.g., ADM3) that have explicit steps for the searching and updating of the reuse repository.

This paragraph should be tailored to require documentation of the contractor's plans for reuse and how reuse fits into the contractor's proposed software development method(s). It is a mistake to attempt to list each reusable software item. This is inappropriate design information that is impossible to know at proposal time when the draft SDP is often due, and the number of reusable non–developmental item (NDI) software items may run into the hundreds on a large project.

6.10 Software Products Evaluation–(Activity Name)

The DOD–STD–2167A SDP DID currently states:

"10.2.8.5.1 Software products evaluation – (activity name) This subparagraph shall be numbered 6.5.X (beginning with 6.5.1) and

shall describe the contractor's plans for conducting evaluations of each of the products of the activity. The description shall identify the specific products to be evaluated. For each product to be evaluated, the evaluation criteria to be used and the evaluation procedures and tool to be employed shall be identified. For evaluations performed on items contained in SDFs. the method of selecting the sample and the percentage of the items to be evaluated shall be specified."

Each OOD method produces slightly different intermediate products that need to be evaluated for completeness, (internal and external) consistency, correctness, and conformance to the OOD method.

This SDP DID paragraph requires tailoring. Object–oriented development methods should have specific product evaluation criteria associated with the method–specific intermediate products, and DOD–STD–21 67A should require these criteria to be identified. Examples include graphic–specific quality criteria and criteria for determining whether they truly document object–oriented designs or violate complexity limits.

7. Conclusion

Current military standards and DIDs, though improved over superseded ones, still require proper tailoring and reinterpretation when used to produce object–oriented software requirements specifications and design documents. This tailoring and reinterpretation is non–trivial and must be performed early during the development of the Request For Proposal (RFP). The proposal should include a draft Software Development Plan (SDP) which also requires appropriate tailoring and reinterpretation. Adequate and timely training of both contractor and contracting agency personnel is essential if proper tailoring and interpretation are to occur. Hopefully, the updates to DOD–STD–2167A and MIL–STD–1521B will remove the remaining roadblocks to planning on object–oriented military projects.

Notes

[1]Iteration is the repetition of some or all of the steps of a development method on existing products of the method, typically to correct errors and add new capabilities. Recursion is the repetition of the steps of the method to generate new products, typically at the next lower level of abstraction. Software is typically developed in increments of subassemblies (i.e., the collection of products developed during each recursive pass through the method) For this reason, DOD–STD–2167A explicitly states that the activities of the development process "may overlap and may be applied iteratively or recursively"

[2] The next generation of databases that store entire objects rather than merely data (e.g., attributes of objects)

[3] Although Figures 1 and 2 of DOD–STD–2167A imply the waterfall development cycle, the figures are not legally binding, are labelled "example", and the text explicitly allows iteration and recursion with result in different development cycles.

[4] A distinct part of a CSCI.

[5] An element specified in the design of a CSC that is separately testable. This is a misleading definition because "separately" implies "independently", when requirements traceability was the original intent.

[6] Although tailoring is by deletion only, additional requirements can be added and modifications to existing requirements can be made via the project statement of work (SOW).

References

[Booch 1991]
Grady Booch, *Object–Oriented Design with*

Applications, Benjamin Cummings, 1991.

[Coad and Yourdon 1989]
Peter Coad and Edward Yourdon, *OORA – Object–Oriented Requirements Analysis*, Prentice Hall, 1989.

[Colbert 1989]
Ed Colbert, "The Object–Oriented Software Development Method: A Practical Approach to Object–Oriented Development," *Proceedings of TRI–Ada '89 -- Ada Technology In Context: Application, Development, and Deployment, 23–26 October 1989*, pages 400–415, 1989.

[DeMarco 1978]
Tom DeMarco, *Structured Analysis and System Specification*, Yourdon Press/Prentice Hall, 1978.

[DI–MCCR–80030A]
Joint Logistics Commanders Computer Software Management Subgroup, "Software Development Plan", Department of Defense, 29 February 1986.

[DOD–STD–2167A]
Joint Logistics Commanders Computer Software Management Subgroup, *Defense Systems Software Development*, Department of Defense, 29 February 1986.

[Firesmith 1989]
Donald G. Firesmith, *Object–Oriented Requirements Analysis and Logical Design: A Software Engineering Approach*, Wiley and Sons, 1992.

[MIL–STD–1521B]
Joint Logistics Commanders Computer Software Management Subgroup, *Technical Reviews and Audits for Systems, Equipments, and Computer Software*, Department of Defense, 5 December 1983.

[Rumbaugh et al. 1991]
James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, and William Lorensen, *Object–Oriented Modeling and Design*, Prentice Hall, 1991.

[Shlaer and Mellor 1988]
Sally Shlaer and Stephen J. Mellor, *Object–Oriented Systems Analysis, Modeling the World in Data*, Yourdon Press, 1988.

[Ward and Mellor 1985]
Paul Ward and Stephen J. Mellor, *Structured Development for Real–Time Systems, Volume 1: Introduction and Tools, Volume 2: Essential Modelling Techniques, Volume 3: Implementation Modelling Techniques*, Yourdon Press, 1985.

[Yourdon and Constantine 1979]
Edward Yourdon and Larry Constantine, *Structured Design*, Prentice Hall, 1979.

General Bibliography of Related Documents

Anderson, John A. and Elaine S. Ward Technology Transfer: Experiences in introducing Object–Oriented Methods to Government Projects", *Proceedings of the Eighth Washington Ada Symposium/Summer SIGAda Meeting*, pages 10–15, 17–21 June 1991.

Ellison, Dr. Karen S. and William J. Goulet, 'A Practical Approach to Methodologies, Ada, and DOD–STD–2167An, *7th National Annual Conference on Ada Technology Proceedings*, pages 51–57, 13–16 March 1989.

Firesmith, Donald G. "Ada Community Concerns Regarding DOD–STD–2167" *Defense Science and Electronics*, Volume 6, Numbers 4 and 7, April/July 1987.

"Ada and DOD–STD–21 67A" *National Institute for Software Quality and Productivity CASE Technology Conference*, 11–12 April 1988.

"DOD–STD–2167 and the Classic Waterfall Life–Cycle".*Tactical Communications Conference – 88*, 5 May, 1988

"The Management Implications of the Recursive Nature of Object–Oriented Development", *1987 AdaEXPO/SIGAda Conference Proceedings*, 7–11 December 1987.

"Resolution of Ada–related Concerns in DOD–STD–2167, Revision A", co-authored with 1 Lt Colin Gilyeat USAF, *Ada Letters*, September/October 1986.

"Software Development Standards and Ada Working Group Issues and Subissues Report,

SIGAda SDSAWG,

"Structured Analysis and Object–Oriented Development are not Compatible", *Ada Letters*, November/December 1991.

"Take a Flying Leap: The Plunge into Object Technology", *American Programmer*, October 1992.

Gray, Dr. Lewis "On Decomposing an Ada CSCI of a Large Command and Control System into TLCSCs, LLCSCs, and Units: With Suggestions for Using DOD–STD–2167A", *Proceedings of the Ninth Annual National Conference on Ada Technology*, pages 55–68, 4–7 March 1991.

Overmyer, Scott P. "The Impact of DOD–STD–2167A on Iterative Design Methodologies: Help or Hinder?", *Software Engineering Notes*, Volume 15, Number 5, pages 50–59, October 1990.

Sodhi, Jag, Guy Daubenspeck, and Thomas Archer, "Forward Entry Device Software Engineering and DOD–STD–2167A Experiences". *8th Annual National Conference on Ada Technology Conference Proceedings*, pages 63–68, 5–8 March, 1990.

Ward, Elaine S. and John A. Anderson "Documenting Object–Oriented Requirements Analysis Understandably for DOD–STD2167A, Structured Development Forum XI Proceedings, 30 AprIl–3 May 1990.

About the Author

Donald G. Firesmith is President of Advanced Software Technology Specialists (ASTS). Prior to founding ASTS in 1988, he was the Division–Level Software Methodologist for Magnavox Electronic Systems Company. There, he developed the OOD Manual and ran an OOD help desk for the Advanced Field Artillery Tactical Data Systems (AFATDS) project that generated approximately 1.2 million non–comment, non–blank lines of object–oriented Ada software. He was the founding chairman of the SIGAda Software Development Standards and Ada Working Group (SDSAWG) which represented the Ada Community during the formal government and industry review of DOD–STD–2167A. He was also an Electronic Industry Associations (EIA) representative on the Council of Defense and Space Industries Association (CODSIA) Software Development Standards Task Force. He received a personal commendation in 1988 from Major General David J. Teal, USAF for "his outstanding contribution to the Joint Logistics Commanders Computer Resource Management Group in support of the development of Defense System Software Development Standard, DOD–STD–2167A." He has developed the object–oriented software development method, ASTS Development Method 3 (ADM3), which was designed for large, complex, real–time systems and which is currently supported by the CASE tools ObjectMaker and Paradigm Plus.

ADA USER

VOLUME 14
1993

Contents