Documenting Object-Oriented Requirements and Designs on Military Projects

13 December 1991
by

Donald G. Firesmith, President
Advanced Software Technology Specialists (ASTS)
17124 Lutz Road
Ossian, IN 46777-9406 USA
Phone: (219) 639-6305
Fax: (219) 747-9389

Abstract

This paper addresses documentation issues from both the military and object-oriented viewpoints, the real and perceived incompatibilities between the analysis and design products of Object-Oriented Development (OOD) and the most common documentation standards mandated on military projects, and makes specific recommendations for properly documenting object-oriented requirements and designs on military projects.

Keywords

Ada, Documentation, DOD-STD-2167A, MIL-STD-1521B, Object, Object-Oriented Development, Software Design Document, Software Requirements Specification

Introduction

The object-oriented paradigm is currently recognized as the best general approach to requirements analysis and design of non-trivial software, especially the large, complex, long-lived, state-of-the-art software of the type often developed for the military in the avionics, C3, and electronic warfare domains. Unfortunately, a great deal of confusion and disagreement exists as to whether or not the use of an Object-Oriented Development (OOD) method is encouraged, permitted, or even prohibited on military projects using current military software development standards such as Defense System Software Development [DOD-STD-2167A] and Technical Reviews and Audits for Systems, Equipments, and Computer Software [MIL-STD-1521B]. Software developers are rarely experienced in both modern OOD and the intent and specific requirements of military policy and standards. This paper will discuss the military expectations and requirements with regard to the documentation of object-oriented requirements and designs, the documentation of object-oriented requirements and design, the actual and perceived inconsistencies between the two, and specific recommendations for documenting object-oriented requirements and designs on military projects using current military standards.

Military Expectations and Requirements

Current military expectations and requirements with regard to the documentation of software requirements and designs are driven by federal law and past and present military policy, standards, and actual and perceived experience. Although military experience until recently was primarily with procedural languages such as CMS_2, JOVIAL, and TACPOL, United States public law [Public Law 101-511] and military policy [DODD 3405.1, DODD 5000.2] clearly mandate the use of the object-based language Ada-83 [ANSI/MIL-STD-1815A-1983]. Although Ada will not fully support inheritance,

polymorphism, and dynamic binding until approximately 1993 with Ada-9X, Object-Oriented Design is clearly very popular (and even preferred and mandated by the military) and many military projects are currently using Object-Oriented Requirements Analysis (OORA) methods in order to improve analysis and the transition to an object-oriented design. Relevant military standards include Defense System Software Development [DOD-STD-2167A] and Technical Reviews and Audits for Systems, Equipments, and Computer Software [MIL-STD-1521B], both of which are currently under revision whereby the issue of compatibility with object-oriented methods is a major topic of discussion. DOD-STD-2167A is organized in terms of software activities (reminiscent of the phases of the classic "Waterfall" development cycle and written in terms of the Software Organizational Entities of Computer Software Configuration Item (CSCI), Computer Software Component (CSC), and Computer Software Unit (CSU). DOD-STD-2167A also has several Data Item Descriptions (DIDs) that mandate the format and content of deliverable documentation, the most important of which for this paper being the Software Requirements Specification (SRS), the Interface Requirements Specification (IRS), the Software Design Document (SDD), and the Interface Design Document (IDD).

According to its forward, DOD-STD-2167A, Defense System Software Development, "establishes uniform requirements for software development that are applicable throughout the system life cycle. The requirements of this standard provide the basis for Government insight into a contractor's software development, testing, and evaluation efforts." When DOD-STD-2167A replaced DOD-STD-2167 on 29 February 1986, several modifications where made specifically to better support both the mandated, object-based language Ada and the increasingly popular object-oriented development paradigm. Although the original standard clearly mandated the use of hierarchical functional decomposition methods, these requirements were removed and the forward of the new standard states that "This standard is not intended to specify or discourage the use of any particular software development method. The contractor is responsible for selecting software development methods (for example, rapid prototyping) that best support the achievement of contract requirements." Furthermore, almost all references to the word "function" were removed to avoid promoting functional decomposition. Computer Software Units (CSUs) need no longer perform a single function, an inappropriate and often impossible requirement if objects and classes are CSUs and have multiple operations (i.e., methods). Support for object-oriented requirements analysis methods was improved because requirements are now decomposed into capabilities instead of functions and subfunctions and the all mention in the SRS of input-processing-output was removed. Paragraph 4.2.1 of DOD-STD-2167A merely requires that the "contractor shall use systematic and well documented software development methods to perform requirements analysis, design, coding, integration, and testing of the deliverable software." The waterfall development *cycle* and its *phases* were replaced in DOD-STD-2167A by a generic software development *process* and *activities* "which may *overlap* and be applied *iteratively* or *recursively* [emphasis mine]." The word "recursively" was added specifically to allow the recursive development cycles common with object-oriented methods.

Unfortunately, what the Government gives with one hand, it may a inadvertently take away with the other. MIL-STD-1521B, "Formal Reviews and Audits" was not updated at the same time as DOD-STD-2167 and is both contractually and philosophically inconsistent with DOD-STD-2167A. DOD-STD-2167A states that "The contractor shall conduct one or more Software Specification Review(s) (SSR) in accordance with MIL-STD-1521. ... The contractor shall conduct one or more Preliminary Design Review(s) (PDR) in accordance with MIL-STD-1521. ... The contractor shall conduct one or more Critical Design Review(s) (CDR) in accordance with MIL-STD-1521." Paragraph 4.2.1 of DOD-STD-2167A further states that "The contractor shall implement software development methods that support the formal reviews and audits required by the contract." Paragraph 40.1 of MIL-STD-1521B states that the Preliminary Design Review

(PDR) "shall be held ... prior to the start of detailed design," thus prohibits the overlap of the preliminary and detailed design activities, is inconsistent with DOD-STD-2167A which allows the activities to "overlap and be applied iteratively or recursively."

Object-Oriented Development (OOD)

There are numerous object-oriented software development methods including, but not limited to, [Booch 1991], [Coad and Yourdon 1989], [Colbert 1989], [Firesmith 1989], [Rumbaugh et al. 1991], and [Shlaer and Mellor 1988]. Because these different methods use slightly different models, graphics, and development cycles, the documentation approach may vary from method to method.

There have, however, a great many similarities that differentiate them from the older methods. First of all, they are based upon the concept of an object, an abstraction or model of an application domain entity that localizes and encapsulates both attributes (i.e., data) and operations (i.e., functions). Objects are not data, are in fact more that the sum of data and related operations, and do not exist in the older methods. Similar or identical objects are instantiated from (i.e., created by) classes, that are templates that exist in inheritance hierarchies of superclasses and subclasses. Classes, inheritance, and the related concepts of polymorphism and dynamic binding are also new to many in the military (and civilian industry). Object-oriented development methods use new and modified models and graphics (see next section).

Most object-oriented methods employ an incremental, iterative, and recursive 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 CSCs and CSUs. Object-oriented methods often incrementally develop assemblies (e.g., CSCI), a subassembly (e.g., CSC) at a time, using a "Analyze a little, design a little, code a little, test a little" recursive development cycle. The concept of a single Preliminary Design Review 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 SRSs and SDDs, are not developed during the same phases of the traditional development cycles

Actual and Perceived Inconsistencies

Numerous inconsistencies, both real and perceived, exist between current military documentation requirements and the proper documentation of object-oriented requirements and designs. 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, represent misunderstandings, or result from human inertia. These inconsistencies are best treated with early and adequate training.

The military considers software to consist of Computer Software Configuration Items (CSCIs) that are decomposed into Computer Software Components (CSCs) and Computer Software Units (CSUs) whose requirements are decomposed into capabilities and whose designs are primarily 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 is organized by CSC (i.e., Preliminary Design) and CSU (i.e., Detailed Design). 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 document the object-oriented entities according to the military standards.

The military is primarily used 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 (see Figure 1), Message Diagrams (see Figure 2), Composition Diagrams (see Figure 3), and the specification and body of all objects (see Figure 4). Class models consist of Classification Diagrams (see Figure 5) and the specification and body of all classes (see Figure 6). State Models are similar, although the State Transition Diagrams (see Figure 7) are restricted to the state of individual objects or classes. Control Models use object-oriented Control Flow Diagrams (CFDs) (see Figure 8) and operation bodies (see Figures 9 and 10) rather than functional decomposition Data Flow Diagrams . Timing Models consist of Timing Diagrams (see Figure 11) and/or STDs and CFDs annotated with temporal information.

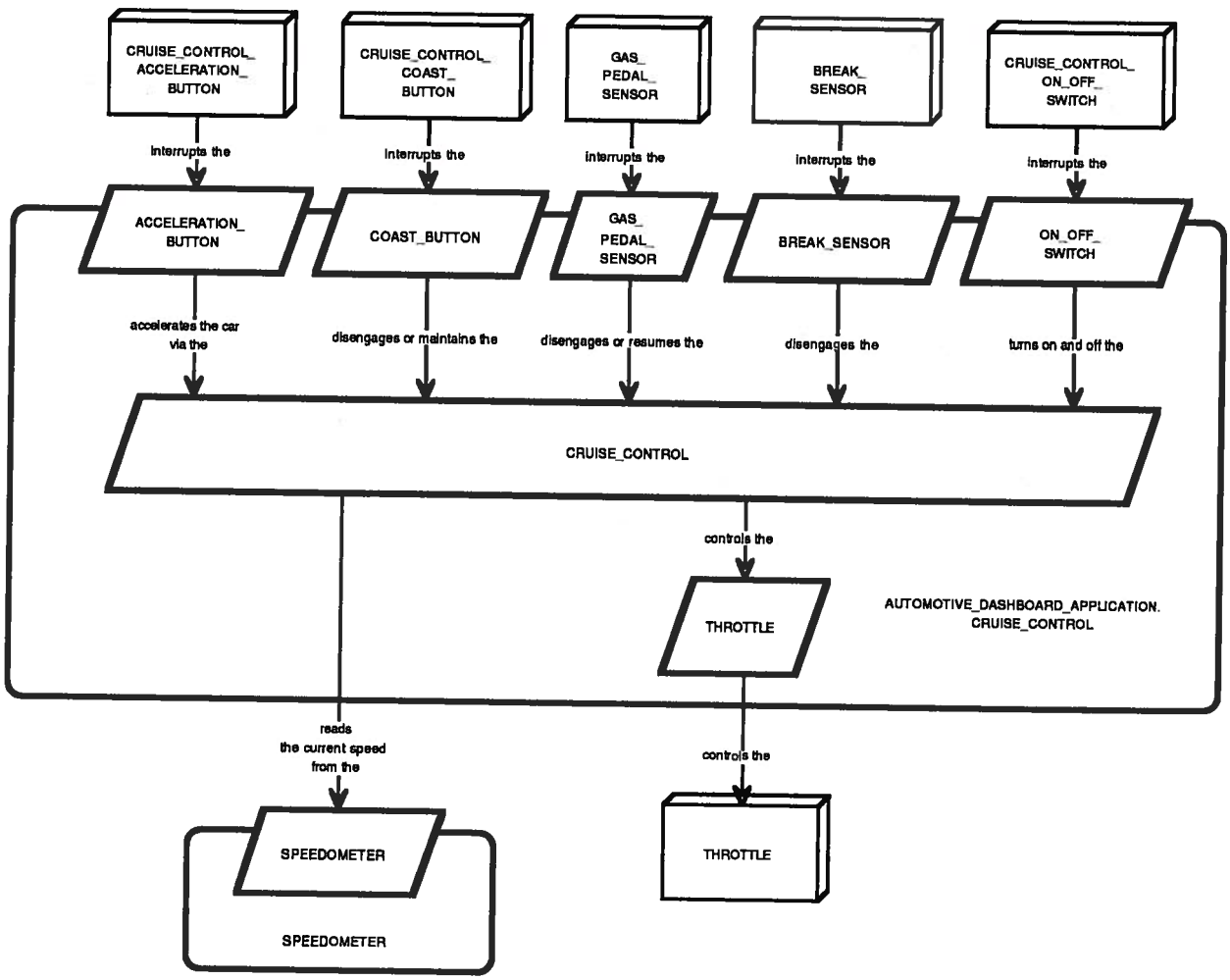

Figure 1: GSN for a `CRUISE_CONTROL` subassembly

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.

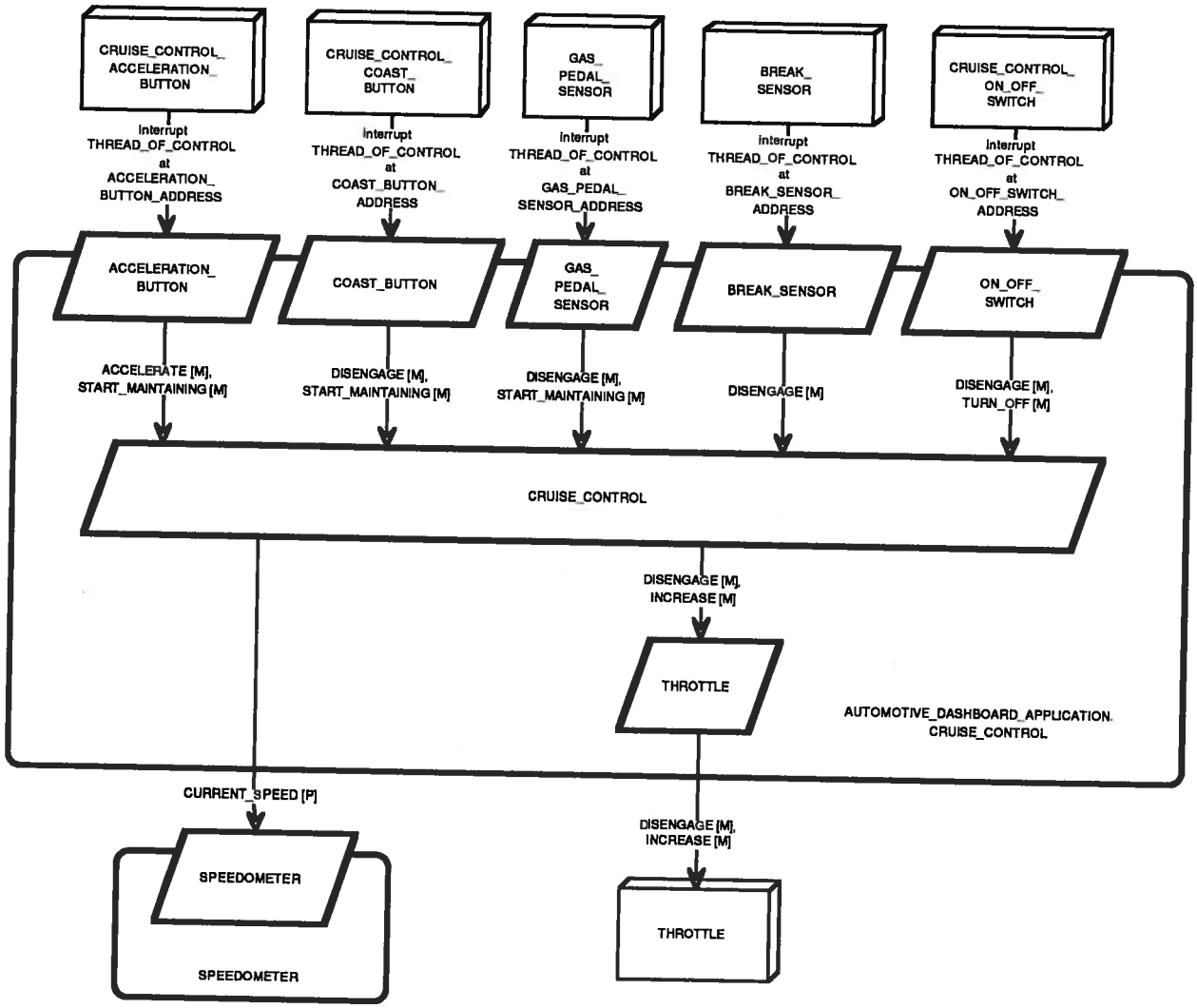

Figure 2: Example Subassembly Message Diagram

The military expects, and sometimes requires, specific documentation formats (e.g., the DOD-STD-2167A SRS [DI-MCCR-80025A] and Software Design Document [DI-MCCR-80012A]) 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 fundamental 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 SRS and SDD DIDs still emphasize the distinction between and separate the documentation of data and functions on that data, while attributes (i.e., data) and operations (i.e., functions) are combined and encapsulated under an object-oriented paradigm.

Many military personnel expect, and in spite of significant changes military standards still partially mandate, 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 SSRs, PDRs, and CDRs.

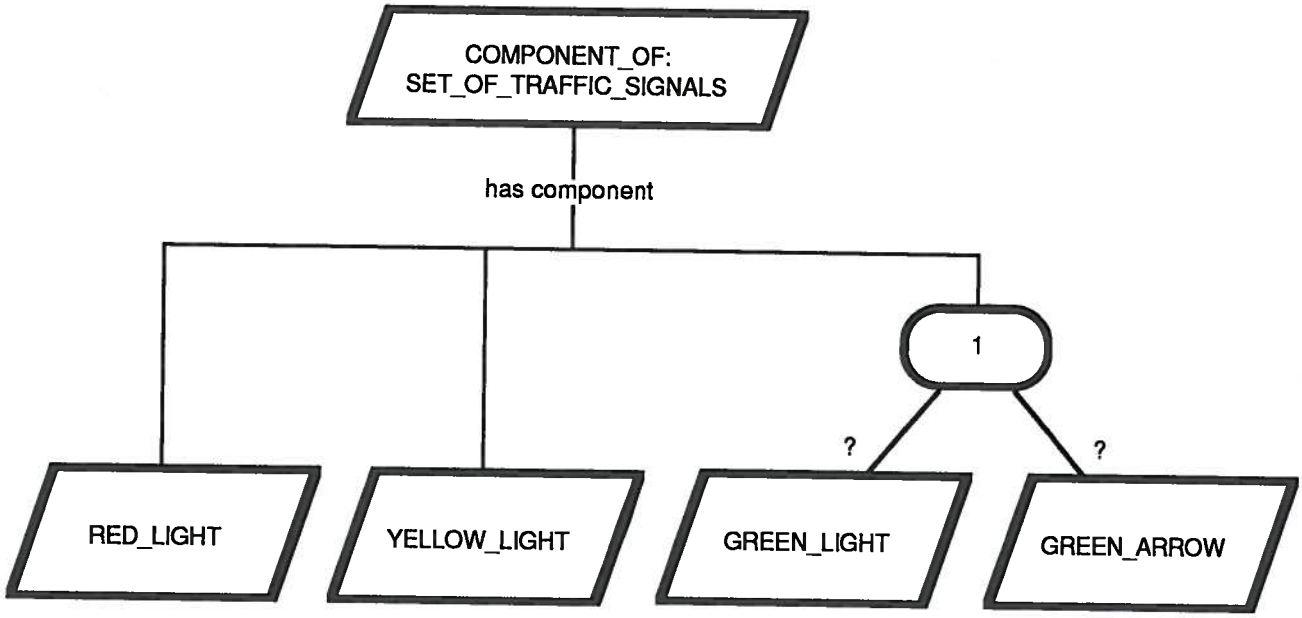

Figure 3: Example Composition Diagram (CMD) of an Aggregate

```
object SPEEDOMETER is concurrent
  parent subassembly SPEEDOMETER;
  needs COMMON.SET_OF_INTEGERS;
specification
  type KILOMETERS_PER_HOUR is new SET_OF_INTEGERS.VALUES range 0 .. 500;
  preserver operation CURRENT_SPEED return KILOMETERS_PER_HOUR;
  modifier  operation UPDATE;
  exception FAILED;
end;

object SPEEDOMETER
body
  variable The_Current_Speed       : KILOMETERS_PER_HOUR := 0.0;
  constant THE_WHEEL_CIRCUMFERENCE : METERS              := TBD;
  modifier  operation DISPLAY is concurrent;
start
  DISPLAY;
  FOREVER : loop
    select
      CURRENT_SPEED return KILOMETERS_PER_HOUR;
    or
      UPDATE;
    end select;
  end loop FOREVER;
end;
```

Figure 4: Example Object Specification and Body

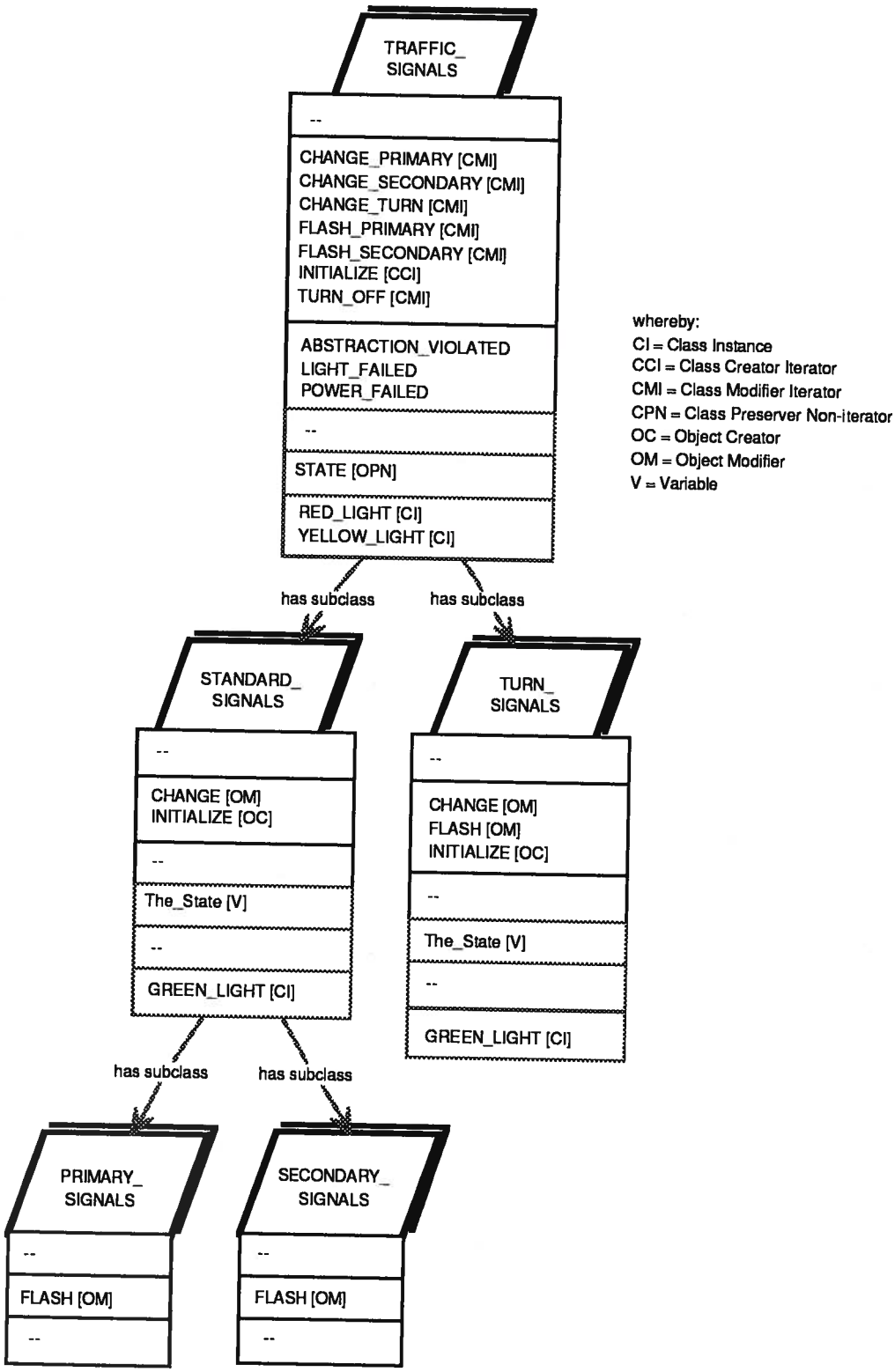

Figure 5: Example Detailed Classification Diagram

```
class TRAFFIC_SIGNALS is concurrent
 parent subassembly INTERSECTION_CLASSES;
specification is abstract
 introduces
  constructor operation CREATE  (A_TRAFFIC_SIGNAL)
   raise CONSTRUCT_FAILED;
  destructor operation  DESTROY (A_TRAFFIC_SIGNAL)
   raise DESTRUCT_FAILED;
  exception             ABSTRACTION_VIOLATED;
  exception             LIGHT_FAILED;
  exception             POWER_FAILED;
end;

class TRAFFIC_SIGNALS
 needs TRAFFIC_LIGHTS;
body
 introduces
  variable              The_State     : STATES := RED_LIGHT;
  object                RED_LIGHT     : TRAFFIC_LIGHTS;
  object                YELLOW_LIGHT  : TRAFFIC_LIGHTS;
start
 INITIALIZE;
end;
```

Figure 6: Example Class Specification and Body

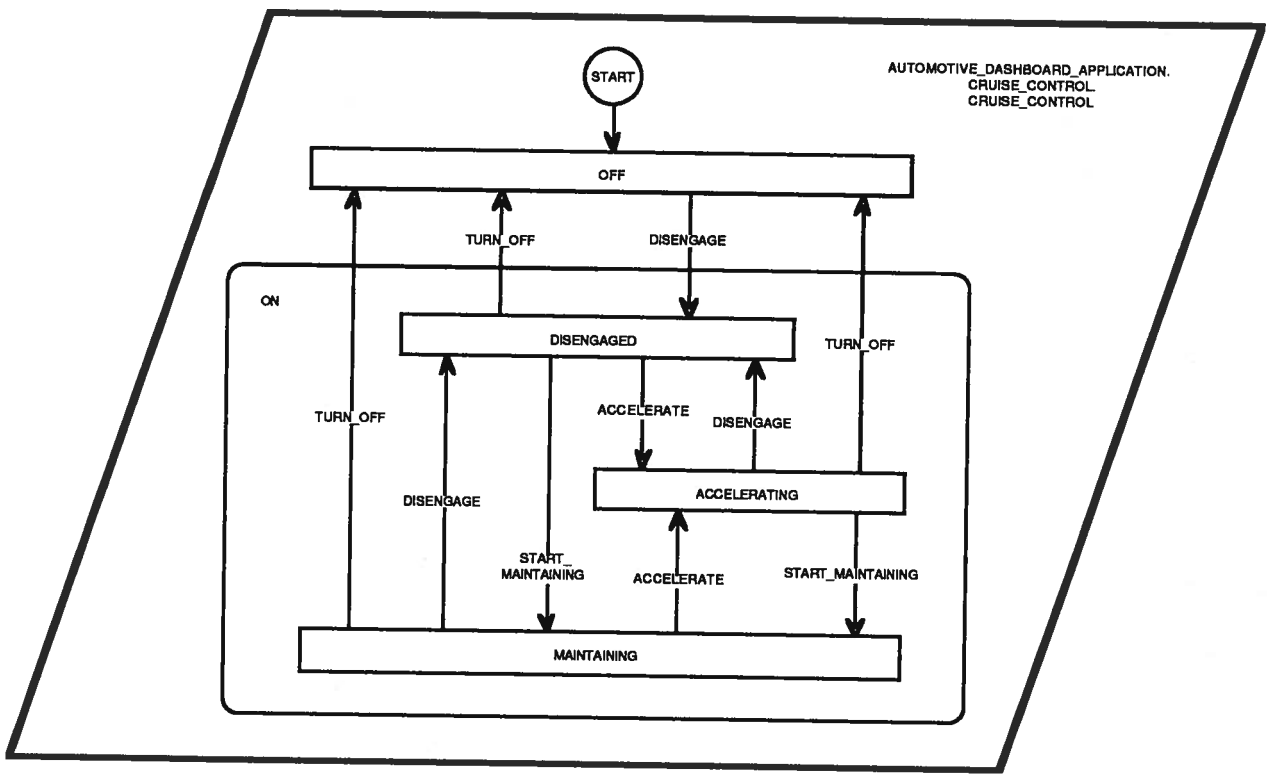

Figure 7: Example Object State Transition Diagram (STD)

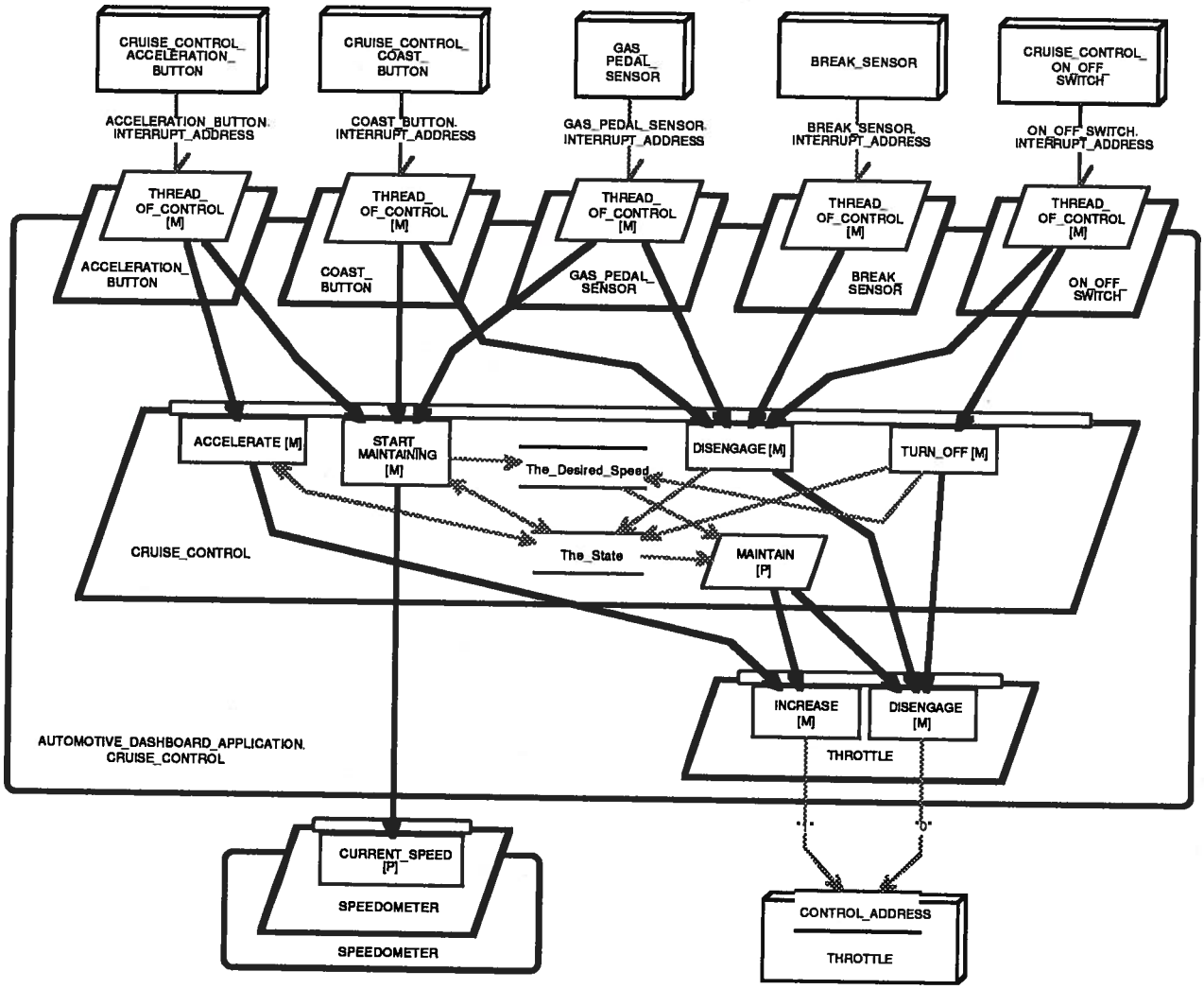

Figure 8: Example Subassembly-Level Control Flow Diagram (CFD)

```
body
operation PUSH (The_Item : in ITEMS)
preconditions
  The_Size < THE_MAXIMUM_SIZE          else raise FULL;
postconditions
  post(The_Size) <= THE_MAXIMUM_SIZE   else raise FULL;
  post(The_Size) =  The_Size + 1       else raise INCORRECT_SIZE;
  post(The_Top)  =  The_Item           else raise ITEM_NOT_ADDED;
end;
```

Figure 9: Example Operation Body specified using Conditions

```
body
  parent object DESIRED_TEMPERATURE;
  needs AIR_CONDITIONER;
  needs FURNACE;
  needs MEAN_TEMPERATURE;
operation MAINTAIN
statements
  FOREVER : loop
    INNER : while The_State = ON loop
      The_Mean_Temperature ::= MEAN_TEMPERATURES.CURRENT_VALUE_OF;
      if
        The_Mean_Temperature < The_Desired_Temperature - THE_MAXIMUM_DIFFERENCE
      then
        AIR_CONDITIONER.TURN_OFF;
        FURNACE.TURN_ON;
      end if;
      if
        The_Mean_Temperature > The_Desired_Temperature + THE_MAXIMUM_DIFFERENCE
      then
        AIR_CONDITIONER.TURN_ON;
        FURNACE.TURN_OFF;
      end if;
      TIME.DELAY(THE_DELAY_AMOUNT);
    end loop INNER;
    TIME.DELAY(THE_DELAY_AMOUNT);
  end loop FOREVER;
exception_handler
  when AIR_CONDITIONER.FAILED   then
    FURNACE.TURN_OFF;
  end when;
  when FURNACE.FAILED           then
    AIR_CONDITIONER.TURN_OFF;
  end when;
  when MEAN_TEMPERATURE.FAILED  then
    AIR_CONDITIONER.TURN_OFF;
    FURNACE.TURN_OFF;
  end when;
  when others                   then
    AIR_CONDITIONER.TURN_OFF;
    FURNACE.TURN_OFF;
  end when;
  raise FAILED;
end;
```

Figure 10: Example Operation Body designed algorithmically

Recommendations

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, their resources (e.g., attributes, operations, exceptions, component objects), their relationships, classes, their inheritance relationships. 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 the traditional military project requirements and expectations and the object-oriented application requirements and design. Contractor technical experts should develop or acquire OOD-method-specific DID formats and contents standards and

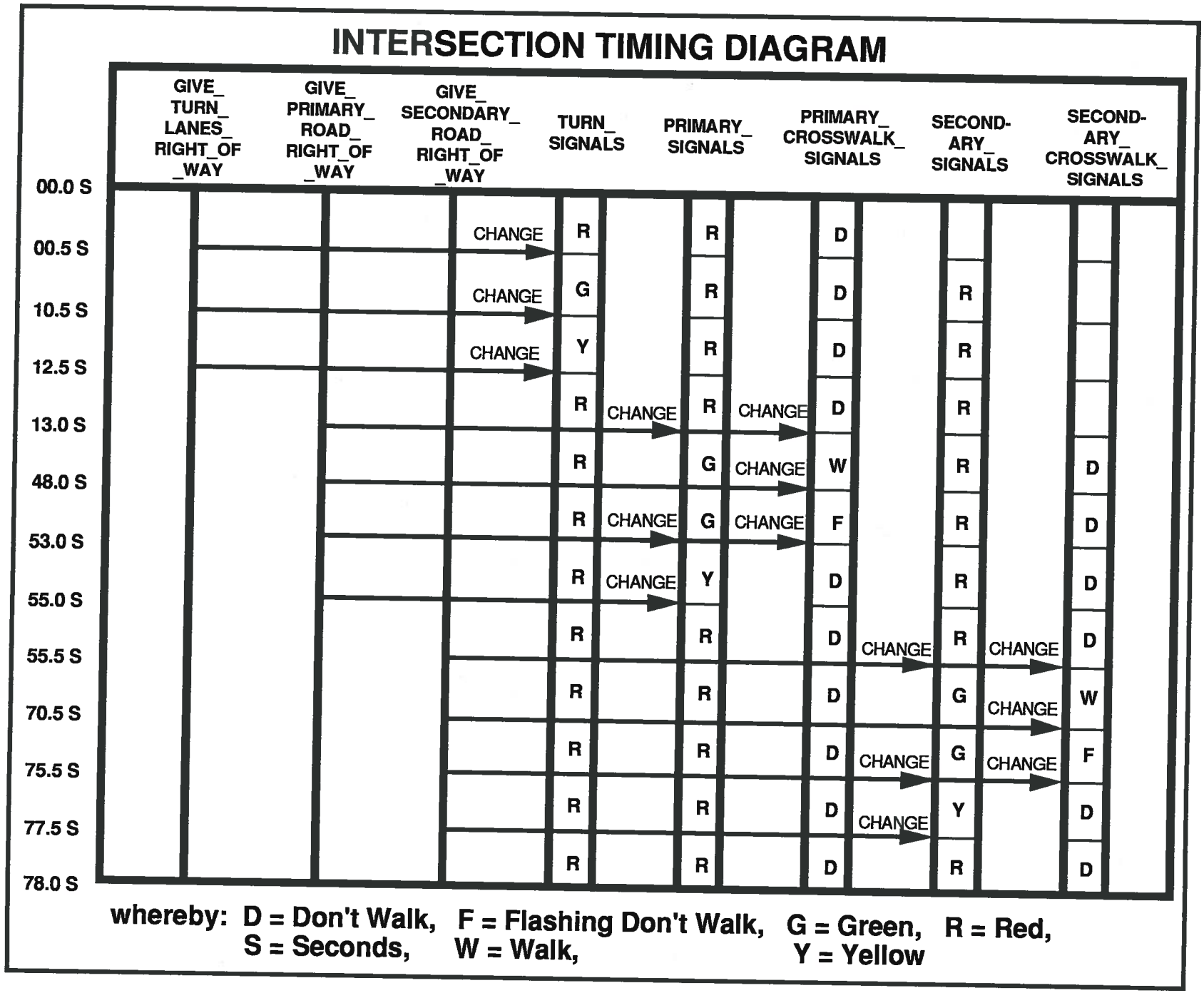

Figure 11: Example Timing Diagram (TD)

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 get the standard and DIDs properly tailored and interpreted using military policy, the recommendations of experts, and organizations like the SIGAda Software Development Standards and Ada Working Group to provide objective rationale and guidance. They should 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.

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) and the proposal which 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 is to occur. Hopefully, the updates to DOD-STD-2167A and MIL-STD-1521B will remove the remaining roadblocks to documenting object-oriented specifications and designs on military projects.

References

[ANSI/MIL-STD-1815A-1983]
Ada Joint Program Office, *Reference Manual for the Ada Programming Language*, United States Government, 1983.

[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-80012A]
Joint Logistics Commanders Computer Software Management Subgroup, "Software Design Document", Department of Defense, 29 February 1986.

[DI-MCCR-80025A]
Joint Logistics Commanders Computer Software Management Subgroup, "Software Requirements Specification", Department of Defense, 29 February 1986.

[DODD 3405.1]

William H. Taft IV, "Computer Programming Language Policy", Department of Defense, 2 April 1987.

[DODD 5000.2]

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

[Firesmith 1989]
"The Ada Development Method (ADM): An Object-Oriented Development Method for the Entire Development Cycle", *Summer 1989 SIGAda Conference Program and Abstracts,*
Pages xvii-xxvi, 8-11 August 1989.

[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.

[US 1990]
United States Congress, *Department of Defense (DOD) Appropriations Act 1991*, Public Law 101-511, Section 89092, United States Government, 5 November 1990.

[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-2167A",
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-2167A",
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",
coauthored with 1Lt 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.

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-STD-2167A",
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 where 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 EIA representative on the 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, which was designed for large, complex, real-time systems and which is currently supported by the CASE tools ObjectMaker and Paradigm Plus.