Documenting Object-Oriented Software Requirements Specifications and Designs

15 December 1992

by

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

Director of Object Technology, Software Consulting Specialists (SCS)
1943 Lakeview Drive
Fort Wayne, IN 46808
voice: (219) 432-3975
fax: (219) 432-3651

1) Introduction

High-quality software documentation is critical to the successful development, use, reuse, and maintenance of software. It has many uses and is central to the way large projects are often developed:

- Customers, users, and system analysts and designers must document their software-related requirements and designs as input to software developers.
- Software analysts must analyze, trace, and specify software requirements for software designers, testers, and maintenance personnel.
- Software designers must design, trace, and document the software logical and language-specific design for programmers and maintenance personnel.
- Software quality assurance personnel (SQA) and independent verification and validation (IV&V) personnel must evaluate the documentation for quality.
- The acceptance of documents is often used as an intermediate milestone with scheduling and contractual implications (e.g., partial payments, authorization to continue).

Modern software development paradigms (e.g., incremental object-oriented development), methods (e.g., the ASTS Development Method ADM3), and languages (e.g., Ada, C++, Eiffel, Smalltalk) use different concepts, models, diagrams, and modules than do older approaches (e.g., functional decomposition, Structured Analysis and Design) using older, procedural languages (e.g., C, COBOL, FORTRAN). They produce different intermediate products in accordance with different development cycles and milestones. They therefore require different documentation standards if the requirements specifications and design documents are to capture the appropriate intermediate products and are to have the same organization as the software they document. Although object-oriented software has radically different structures than functionally-decomposed procedural software, traditional documentation standards are based on software developed using older paradigms, methods, and languages.

Documentation should also be developed and reviewed in accordance with the software development cycle used to produce it. Object-oriented software is typically developed

incrementally using both iteration and recursion, whereas traditional software is usually developed in accordance with the waterfall development cycle.

Unfortunately, software documentation has always been problematic. American National Standards Institute (ANSI), Institute of Electrical and Electronics Engineers (IEEE), United States military, and NASA documentation content and format standards are inappropriate for documenting the requirements and designs of object-oriented software and often are used to control a document driven approach based on the classic waterfall development cycle. Large projects often produce Gothic novel sized documents, the size and complexity of which makes a credible manual review within project deadlines very difficult. Completeness and consistency is often impossible to guarantee. Software documentation is typically developed manually. Although much of the documentation should be automatically derivable from a project requirements and design repository, most Computer Aided Software Engineering (CASE) tools currently develop documentation of questionable value and drive the developer into using older, functional decomposition methods. Few software developers like developing documentation, and it shows in the quality of the resulting documentation. Most CASE tools (e.g., analysis, design, documentation, and coding) do not yet adequately support object-oriented develop and are not properly integrated. Most developers still develop documentation by hand. Documentation and the software that it documents are therefore typically not consistent when the software is delivered, and they become even less consistent as time progresses and the code is updated whereas the documentation is not.

2) Recommended Content and Format Standard

The remainder of this paper is the ASTS content and format standard for documenting object-oriented software requirements and designs. This standard is based on the ASTS Development Method 3 (ADM3) which is documented in *Object-Oriented Requirements Analysis and Logical Design: A Software Engineering Approach* authored by Donald Firesmith and published by John Wiley and Sons. Because the same concepts, models, diagrams, and specification and design language is used for both requirements analysis and design and because ADM3 uses an incremental "analyze a little, design a little, code a little" development process, the requirements are specified and the design is documented in the same object-oriented software requirements specification and design document (OOSRSDD). Projects using a separate document for requirements and design may divide the following standard into 2 standards with analogous organization and requirements. Because the following standard is intended for use on multiple projects and at the organizational level, certain words and phrases enclosed in angle brackets (e.g., "<" and ">") may need to be replaced with the appropriate project- or organizational-specific words. An electronic copy of the standard and associated template as well as an example corresponding OOSRSDD may be purchased from ASTS for use on projects and as part of training. The following standard and the ADM3 object-oriented software development method are in the public domain and are recommended by ASTS for use on projects using object technology.

3) Conclusion

The object community needs production quality upperCASE tools that capture and document object-oriented requirements and designs and that are integrated with object-oriented lowerCASE tools and environments. This paper provides a method-specific documentation standard that can be used as a starting point for such CASE tool development and may be used manually until such CASE tools are available.

Object-Oriented Software Requirements Specification And Design Document Standard

14-Dec-92

Advanced Software Technology Specialists (ASTS)
17124 Lutz Road
Ossian, IN 46777
(219) 639-6305

1 Introduction

This standard documents the content and format requirements for the Object-Oriented Software Requirements Specification and Design Document (OOSRSDD), which is developed for each assembly to:

- Specify its engineering and qualification requirements
- Document its complete design

The OOSRSDD is incrementally:

- Developed and evaluated by the development organization on a subassembly by subassembly basis in accordance with the *ASTS Object-Oriented Requirements Analysis and Logical Design Procedure*.
- Evaluated by the customer at the project In-Process Reviews (IPRs)

The template for the OOSRSDD is defined in the *ASTS Object-Oriented Software Requirements Specification and Design Document Template*. Guidelines concerning the concepts and models to be documented in the OOSRSDD are documented in the *ASTS Object-Oriented Concepts and Modeling Guideline*. This procedure is based on and takes precedence over the book *Object-Oriented Requirements Analysis and Logical Design: A Software Engineering Approach* by Donald G. Firesmith [Firesmith 1993].

1.1 Objectives

The objectives of this standard are to:

- Document the content and format requirements for the OOSRSDD
- Standardize the project deliverable documentation
- Significantly improve the:
  - Quality and uniformity of the deliverable documentation
  - Productivity of the project software engineers
- Be a living document subject to continuous product improvement as object technology evolves

The objectives of the OOSRSDD are to:

- Specify the engineering and qualification requirements for an assembly
- Document the complete design of an assembly
- Document the allocation of requirements to design
- Be used as a basis for:
  - Assembly design, coding, testing, and maintenance
  - Understanding and evaluation of the assembly requirements and design
- Describe the assembly as organized in terms of:
  - Architecture
    - Subassemblies
      - Classes and objects of anonymous class
        * Attribute types
        * Attributes
        * Messages
        * Operations
        * Exceptions
  - Scenarios
  - Inheritance Hierarchies

1.2 Applicability

The intended audience for this standard is all <project/organizational> software engineering managers and software engineers.

The requirements of this standard shall apply to the documentation of all:

- New software

- Major upgrades to software involving 20% or more new or modified software

1.3 Glossary

Assembly n. a logically-related collection of subassemblies that (1) is identified during system requirements analysis and design, (2) is typically developed incrementally, (3) is a major software configuration item, and (4) typically implements a significant, cohesive set of system requirements.

Assembly diagram (AD) n. a directed graph documenting the subassemblies of an assembly, whether they are visible outside the assembly, the dependency relationships between them, and optionally their specifications and bodies.

Attribute n. a discrete inherent data abstraction of a characteristic, property, trait, quantity, or quality of an object or class that identifies, describes, or provides the state of an object or class.

Attribute type n. a template for creating attributes that specifies the implementation of the attribute and the allowed operations on attributes of the type.

Class n. a template from which objects (i.e., instances) can be instantiated (i.e., constructed). All instances of the same class share the same or similar attribute types, attributes, messages, exceptions, operations, and component objects (if any).

Context diagram (CD) n. a directed graph documenting the assembly as its primary node, the assembly's primary and secondary terminators (e.g., assemblies, databases, devices, persons, systems) and the association relationships between and among them.

Control flow diagram (CFD) n. a directed graph documenting objects, classes, subassemblies, their terminators, the attributes and operations of the objects and classes, and the control flows (e.g., message passing, exception raising) and optionally the data flows between the objects, classes and terminators.

Essential adj. an object, class, or subassembly (1) whose existence and capabilities are required, (2) which is typically known and understood by the customer or application domain specialist, and (3) which is identified during software requirements analysis and specified in the appropriate software requirements specification and design document, either as a required capability or as a design constraint.

Exception n. an error condition of (and raised) by an operation of an object or class.

General semantic net (GSN) n. a directed graph documenting objects, classes, subassemblies, their terminators, and the association relationships between and among them.

Inheritance n. the relationship among classes that allows subclasses to be built as extensions or specializations of superclasses.

Interaction diagram (ID) n. a directed graph documenting the objects, classes, subassemblies, their terminators, and the interaction relationships (e.g., message passing, exception raising, visibility of exported attribute types) between and among them.

Message n. a sequential, synchronous, or asynchronous call, possibly bound at runtime, to an object or class that (1) requests a service, (2) provides data, or (3) provides notification of an event.

Module diagram (MD) n. an implementation language-specific diagram documenting the architecture of a subassembly in terms of black-box modules and their dependency relationships.

Object n. an abstraction (i.e., model) in the requirements, design, and/or code of a *single* tangible or intangible object, entity, or thing from the real-world, application, problem domain, or solution space that is typically implemented as (1) an encapsulation of attributes, operations, and exceptions that sends and/or receives messages requesting services, providing data, or notifying of the occurrence of events and (2) an instance of one or more classes.

Operation n. a discrete activity, action, or behavior that (1) implements a functional (i.e., sequential) or process (i.e., concurrent) abstraction and (2) is performed by, belongs to, and encapsulated in an object or class. Also known as a member function and service.

Scenario n. a specific collaboration of objects, classes, and terminators used for analysis, design, and/or testing purposes. Also known as a use case.

State transition diagram (STD) n. a directed graph documenting the states and the transitions between states of an object or class.

Subassembly n. a small, logically cohesive collection of objects, classes, and subassemblies that are analyzed, designed, coded, tested, integrated, and documented as a group. Also known as a cluster, computer software component, subject, and subsystem.

Submodule diagram (SMD) n. an implementation language-specific diagram documenting the architecture of a subassembly in terms of white-box modules, their subunits, and their interactions.

System semantic net (SSN) n. a directed graph documenting the system-level objects (e.g., assemblies, concepts, databases, devices, documents, events, interactions, locations, persons, systems) and the association relationships between and among them.

Timing Diagram (TD) n. a diagram documenting the temporal interactions between and within objects and classes. Duration timing diagrams (DTDs) document the duration of operations, whereas Event timing diagrams (ETDs) document the timing of interactions.

1.4 Modifications

1.4.1 Tailoring

This standard should be tailored for each project. Tailorings of this standard:

- Shall be justified in writing
- May be based on contractual requirements, sound software engineering reasons, total life-cycle costs, and schedule requirements
- Must be approved by the Director of Object Technology prior to implementation

Tailoring may include:

- Removing requirements that are unnecessary or not cost-effective
- Modifying requirements to meet the specific needs of the project
- Adding additional requirements where necessary or cost-effective

1.4.2 Deviations

Deviations to this standard:

- Shall be justified in writing
- May be based on contractual requirements, sound software engineering reasons, total life-cycle costs, and schedule requirements
- Must be approved by the Project Manager prior to implementation

1.3.3 Updates

This standard is a living document that must continually improve as object technology improves. Requests for changes to this standard shall be provided to the Director of Object Technology. Requests for changes shall be acted on in a timely manner; the Director of Object Technology shall incorporate improvements and notify the <project/organizational> personnel of the disposition of the requests for changes. At the discretion of the Project manager, upgrades to this standard may, but need not, be applied to ongoing projects.

1.4 References

1.4.1 ASTS Documents

Procedures:
- Software Development Process Procedure
- Object-Oriented Requirements Analysis and Logical Design Procedure

Standards:
- Object-Oriented Models Standard
- Object-Oriented Diagrams Standard
- CASE Tool Requirements Standard

Templates:
- Object-Oriented Software Specification and Design Document Template

Guidelines:
- Object-Oriented Concepts and Modeling Guideline

1.4.2 Other Publications

[Firesmith 1993]
Firesmith, Donald G. *Object-Oriented Requirements Analysis and Logical Design: A Software Engineering Approach*, John Wiley and Sons, New York, NY, 1993.

2 General Content and Format Requirements

2.1 Document Structure. The OOSRSDD shall consist of the following sections:

1) Title page
2) Table of contents
3) Introduction
4) Context
5) Software engineering goals
6) Methods, Languages, and CASE Tools
7) Reuse
8) Requirements and design
9) Notes
10) Appendixes

2.2 Requirements. The OOSRSDD shall document all assembly requirements including the essential:
- Software engineering goals
- Architectural (e.g., existence of essential objects, classes, attributes, operations, associations)
- Behavioral:
  - Functional
  - Performance
- Interface
- Design constraints

The requirements shall be complete, consistent, and unambiguous. The OOSRSDD shall document requirements for both normal operation and exceptional situations.

2.2.1 Identification. All requirements shall be uniquely identified for purposes of requirements traceability and clearly identified as requirements by inclusion of the word "shall" (e.g., R-084: This class *shall*....). Text shall be used to identify any requirements documented in the form of diagrams or tables.

2.2.2 Traceability. For each requirement, the OOSRSDD shall document:
- The source of the requirement (e.g., document, person)
- The specific objects, classes, and their resources (i.e., attribute types, attributes, messages, exceptions, operations) that implement it

2.2.3 Validation. All requirements documented in the OOSRSDD shall be validatable. The OOSRSDD shall document the intended validation methods (e.g., analysis, formal proof of correctness, inspection, test) for each requirement.

2.3 Design. The OOSRSDD shall document the entire assembly design including:
- Logical design
- Implementation (e.g., language-, operating system-, and database-specific) design

2.4 Tailoring. In the event that a paragraph or subparagraph of this standard is tailored out, a statement to that effect shall be added directly following the heading of each corresponding (sub)paragraph in the OOSRSDD. If a paragraph and all of its subparagraphs are tailored out, then only the highest level paragraph heading need be included.

3 Specific Content and Format Requirements

3.1 Title Page. The title page shall contain the information identified in appendix A.1 of this standard in the indicated format.

3.2 Table of Contents. The OOSRSDD shall contain a table of contents listing the title and page number of each titled paragraph and subparagraph. The table of contents shall then list the title and page number of each figure, table, and appendix, in that order.

3.3 Introduction. This section of the OOSRSDD shall be numbered 1 and shall be divided into the following paragraph.

3.3.1 Document Overview. This paragraph shall be numbered 1.1 and shall briefly summarize the purpose and contents of the OOSRSDD.

3.3.2 System Overview. This paragraph shall be numbered 1.2 and shall identify and briefly describe the system containing the assembly to which this OOSRSDD applies. System semantic nets and system timing diagrams may be used.

3.3.3 Assembly Overview. This paragraph shall be numbered 1.3 and shall identify and briefly describe the assembly to which this OOSRSDD applies. This paragraph shall also identify all sources of higher-level requirements and designs from which the requirements and design of this assembly were derived.

3.3.4 References. This paragraph of the OOSRSDD shall be numbered 1.4 and shall begin with the following paragraph: "The following documents of the exact issue shown form a part of this OOSRSDD to the extent specified herein. In the event of conflict between the documents referenced herein and this OOSRSDD, the contents of this OOSRSDD shall take precedence." Where applicable, references shall be identified by title, identifier, author, publisher, and date.

3.3.4.1 Customer Documents. This subparagraph shall be numbered 1.4.1 and shall identify all referenced customer documents.

3.3.4.2 <Project/Organizational> Documents. This subparagraph shall be numbered 1.4.2 and shall identify all referenced <project/organizational> documents.

3.3.4.3 Other Documents. This subparagraph shall be numbered 1.4.3 and shall identify any other referenced documents.

3.4 Context. This section of the OOSRSDD shall be numbered 2 and shall document the context of the assembly. The following may be used:

- One or more system semantic nets (SSNs) of the subsystem containing the assembly
- One or more context diagrams (CDs) of the assembly

Each external interface shall be identified by association name and project-unique identifier. A brief description of each interface shall be given.

3.5 Software Engineering Goals. This section of the OOSRSDD shall be numbered 3 and shall document, prioritize, and (where practical) quantify the general requirements associated with the software engineering goals of:

- Compatibility
- Correctness
- Efficiency
- Extendability
- Maintainability
- Portability
- Reliability
- Robustness
- Safety
- Security
- Reusability
- Understandability
- Validatibility

3.6 Methods, Languages, and CASE Tools. This section of the OOSRSDD shall be numbered 4 and shall identify the following which are to be used to analyze, specify, design, and document the assembly:

- The software development method(s) (e.g., ADM3)
- The specification, design, and implementation language(s)
- The upper and lower CASE tools
- The mapping of the concepts of the method (e.g., object, class, subassembly) to the concepts of the language (e.g., package, module) and the concepts of the customer (e.g., unit, component)

3.7 Reuse. This section of the OOSRSDD shall be numbered 5 and shall identify the main sources of reuse (e.g., repositories, frameworks, class libraries, and commercial off the shelf products) that are incorporated into the assembly. Individual objects, classes, and subassemblies that are reused shall be identified as such in the appropriate paragraphs of the OOSRSDD.

3.8 Requirements and Design. This section of the OOSRSDD shall be numbered 6 and shall document both the requirements and the design of the assembly. It shall be divided into the following paragraphs and subparagraphs.

3.8.1 Requirements and Design Organized by Architecture. This paragraph of the OOSRSDD shall be numbered 6.1 and shall document the requirements and design of the assembly that are completely allocated to the following architectural entities:

- Assembly

- Subassemblies
- Classes
- Objects of anonymous class
- Attribute types and attributes
- Messages
- Exceptions
- Operations

3.8.2.1 Assembly. This subparagraph of the OOSRSDD shall be numbered 6.1.1 and shall document the requirements and design of the assembly as a whole including its modes and states. The following may be used:

- The assembly specification and body
- One or more assembly diagrams (ADs)
- Event timing diagrams (ETDs)
- Timed event lists

3.8.2.2 Subassemblies. This subparagraph of the OOSRSDD shall be numbered 6.1.2 and shall document the requirements and design of the individual subassemblies. This subparagraph shall consist of the subparagraphs for each subassembly, sorted by subassembly name. The title of each of these subparagraphs shall be "Subassembly <subassembly name>." The subparagraph for each top level subassembly shall be nested within subparagraph 6.1.2 and numbered accordingly (e.g., 6.1.2.X). The subparagraph of any other subassemblies shall be nested within the subparagraph of its parent subassembly and numbered accordingly (e.g., 6.1.2.X.Y). The subassembly subparagraphs of the same subparagraph of the OOSRSDD shall be sorted alphabetically by subassembly name.

This subparagraph shall document the subassembly requirements and design including:

- Abstraction and purpose
- Contents:
  - Subassemblies
  - Classes
  - Objects
- Specification and body
- External interfaces:
  - Clients
  - Colleagues
  - Servers
- Diagrams (where cost-effective):
  - General semantic net (GSN)
  - Interaction diagram (ID)
  - Control flow diagram (CFD)
  - Timing diagrams (TDs):
    - Event timing diagram (ETD)
    - Duration timing diagram (DTD)
  - Module diagram (MD)
  - Submodule diagram (SMD)
- Memory budget (where appropriate)

This subparagraph shall be divided into the following subparagraphs to document the requirements and design of each of the subassemblies, classes, and objects in the subassembly. Each class and object that is contained in more than one subassembly shall be documented in detail under only one subassembly and then documented by reference in the other subassemblies.

3.8.2.2 Classes. The title of this subparagraph shall be "Class <class name>." This subparagraph shall be nested within the paragraph of its parent subassembly. The subparagraphs documenting classes within the same subassembly shall be listed in alphabetical order by class name. This paragraph shall document the class requirements and design including:

- Abstraction
- Categorization:
  - Generic or nongeneric
  - Abstract or concrete
  - Concurrent or sequential
  - Transient, temporary, or permanent
- Specification:
  - Superclasses
  - Generic formal parameters
  - Attribute types
  - Constant attributes
  - Messages
  - Operations
  - Exceptions
  - Dependencies
- Body:
  - Attribute types
  - Attributes
  - Operations
  - Invariants
  - Dependencies
- Cardinality and known instances
- Diagrams (where cost-effective):
  - General semantic net (GSN)
  - Interaction diagram (ID)
  - State transition diagram (STD)
  - Control flow diagram (CFD)
  - Timing diagrams (TDs):
    - Event timing diagram (ETD)
    - Duration timing diagram (DTD)
  - Module diagram (MD)
  - Submodule diagram (SMD)
- Memory budget (where appropriate)

3.8.2.3 Objects of Anonymous Class. The title of this subparagraph shall be "Object <object name>." This subparagraph shall be nested within the paragraph of its parent subassembly. The subparagraphs documenting objects within the same subassembly shall be listed in alphabetical order by object name. This paragraph shall document the object requirements and design including:

- Abstraction
- Categorization:
  - Concurrent or sequential
  - Transient, temporary, or permanent
- Specification:
  - Attribute types
  - Constant attributes
  - Messages
  - Operations
  - Exceptions
  - Dependencies
- Body:
  - Attribute types
  - Attributes
  - Operations
  - Invariants
  - Dependencies
- Diagrams (where cost-effective):

  - General semantic net (GSN)
  - Interaction diagram (ID)
  - State transition diagram (STD)
  - Control flow diagram (CFD)
  - Timing diagrams (TDs):
    - Event timing diagram (ETD)
    - Duration timing diagram (DTD)
  - Module diagram (MD)
  - Submodule diagram (SMD)
- Memory budget (where appropriate)

3.8.2.4 Attribute Types and Attributes. The title of this subparagraph shall be "Attribute types and attributes." This subparagraph shall be nested within the paragraph of its parent class or object of anonymous class. Attributes types and attributes shall be documented in the same order as in the specification and body of their parent classes and objects. This paragraph shall document the requirements and design of each attribute type:

- Abstraction
- Categorization
  - State, description, pointer, or key
- Declaration including:
  - Units of measure
  - Range of values
  - Accuracy
  - Precision
  - Frequency of calculation or update
  - Base type
  - Source and destination (e.g., database, analog to digital converter)

This paragraph shall document the requirements and design of each attribute:

- Abstraction
- Categorization, for example:
  - State, description, pointer, or key
- Declaration including:
  - Type
  - Frequency of calculation or update
  - Source and destination (e.g., database, analog to digital converter)
- Initial value

3.8.2.5 Messages. The title of this subparagraph shall be "Messages." This subparagraph shall be nested within the paragraph of its parent class or object of anonymous class. Messages shall be documented in the same order as in the specification of their parent classes and objects. This paragraph shall document the requirements and design of each message:

- Abstraction
- Categorization, for example:
  - Service request, event notification, or data supply
  - Sequential, synchronous, asynchronous
  - Directed or broadcast
- Priority
- Timing budget (where appropriate)

3.8.2.6 Exceptions. The title of this subparagraph shall be "Exceptions." This subparagraph shall be nested within the paragraph of its parent class or object of anonymous class. Exceptions shall be listed in the same order as in the specification of their parent classes and objects. This paragraph shall document the requirements and design of each exception:

- Abstraction
- Allocated and derived requirements

- Condition on which it is raised

3.8.2.7 Operations. The title of this subparagraph shall be "Operations." This subparagraph shall be nested within the paragraph of its parent class or object of anonymous class. Operations shall be documented in the same order as in the body of their parent classes and objects. This paragraph shall document the requirements and design of each operation:

- Abstraction
- Categorization, for example:
  - Sequential or concurrent
  - Constructor, destructor, modifier, or preserver
- Preconditions, postconditions, and invariants
- Specification and body
- Exception handling
- Priority
- Dependencies
- Memory budget (where appropriate)
- Timing budget (where appropriate)

3.8.2.8 Local Scenarios. The title of this subparagraph shall be "Local Scenarios." This subparagraph shall be nested within the paragraph of its parent subassembly, class, or object of anonymous class. This paragraph shall document the requirements and design of each local scenario (i.e., a scenario that does not cross subassembly, class, or object boundaries):

- Description
- Objective
- Preconditions, postconditions, and invariants
- Exception handling
- Diagrams (where cost-effective):
  - General semantic net (GSN)
  - Interaction diagram (ID)
  - Control flow diagram (CFD)
  - Timing diagrams (TDs):
    - Event timing diagram (ETD)
    - Duration timing diagram (DTD)
  - Submodule diagram (SMD)
- Timing budget (where appropriate)

3.8.3 Organized by Global Scenarios. This paragraph of the OOSRSDD shall be numbered 6.2 and shall document the requirements and design of each global scenario (i.e., scenarios that cross subassembly boundaries):

- Description
- Objective
- Preconditions, postconditions, and invariants
- Exception handling
- Diagrams (where cost-effective):
  - General semantic net (GSN)
  - Interaction diagram (ID)
  - Control flow diagram (CFD)
  - Timing diagrams (TDs):
    - Event timing diagram (ETD)
    - Duration timing diagram (DTD)
  - Submodule diagram (SMD)
- Timing budget (where appropriate)

3.8.4 Organized by Inheritance Hierarchies. This paragraph of the OOSRSDD shall be numbered 6.3 and shall document the requirements and design of each inheritance hierarchy:

- Description

- Categorization, for example:
  - Single or multiple inheritance
- Root classes
- Diagrams (were cost-effective):
  - Classification diagram (CLD)

3.8.5 Other Software. This paragraph of the OOSRSDD shall be numbered 6.4 and shall document the requirements and design of any software that is not object-oriented.

3.8.5.1 Common Global Operations. This subparagraph of the OOSRSDD shall be numbered 6.4.1 and shall document and justify the requirements and design of any common global operations (i.e., operations not encapsulated within classes or objects).

3.8.5.2 Common Global Data. This subparagraph of the OOSRSDD shall be numbered 6.4.2 and shall document and justify the requirements and design of any common global data (i.e., data and data types not encapsulated as attributes and attribute types within classes or objects).

3.8.6 Notes. This section of the OOSRSDD shall be numbered 7 and shall briefly document:
- Any general information that aids in understanding the document
- Major alternative designs that were considered and then rejected (e.g., to improve maintainability)

3.8.7 Appendices. The appendices of the OOSRSDD shall include:
- A project-specific abbreviations list
- A project-specific glossary
- A requirements trace to the customer documentation standards (if any)

About the Author

Donald G. Firesmith is President of Advanced Software Technology Specialists (ASTS) and Director of Object Technology at Software Consulting Specialists (SCS). He is the author of *Object-Oriented Requirements Analysis and Logical Design: A Software Engineering Approac* and is currently writing Object-Oriented Development Methods, Standards, and Procedures. H has developed the object-oriented software development method, ASTS Development Method (ADM3), which was designed for large, complex, real-time systems and which is currently supported by the CASE tools ObjectMaker and Paradigm Plus. He is also the developer of the OOSDL object-oriented specification and design language. He provides consulting, training, and independent verification and validation (IV&V) in the areas of object technology and Ada.

Prior to founding ASTS in 1988, he was the Division-Level Software Methodologist for Magnavox Electronic Systems Company. There, he developed the project OOD Manual and r an OOD help desk for the Advanced Field Artillery Tactical Data Systems (AFATDS) project tha generated approximately 1.2 million non-comment, non-blank lines of object-based Ada software. He was the founding chairman of the SIGAda Software Development Standards anc 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 contributic to the Joint Logistics Commanders Computer Resource Management Group in support of the development of Defense System Software Development Standard, DOD-STD-2167A."

Object-Oriented Software Requirements Specification and Design Document for the

<Assembly Name>

of

<System Name>

Version No. <Version Number>

<Date>

Prepared for:

<Customer Name>
<Customer Address>

PO # <Purchase Order Number>

Prepared by:

<Customer Name>
<Customer Address>

Approved by _____
<Developer Representative>

Date _____

Approved by _____
<Customer Representative>

Date _____