Using MIL-STD-498 and ISO/IEC 12207 for OOD and RAD

1. Object-Oriented Development Methods (OOD)

Object-oriented methods have been used for software as different from each other as military mission-critical systems and commercial data processing systems. There are O-O methods for software requirements analysis (OOA), software design, and for programming (OOP). Popular guidelines for object-oriented analysis and design are found in Booch [1] and Coad and Yourdon [2]. Probably, they are familiar to most engineers who practice OOD. I summarize the guidelines below to establish a baseline description of OOD. In sections 3 and 4 below, I show how MIL-STD-498 and ISO/IEC 12207 can be used with software development efforts that follow these guidelines.

1.1 Booch’s Model for OOD.

Booch [1] has modeled OOD as a repeating process within a repeating process. A repeating micro development process is embedded in several major activities of a repeating macro development process. Figure 1 and Figure 2 below from Booch show the macro process and the micro process. The macro process is the total project context for OOD. It consists of five activities: establish core requirements; develop a model of the desired behavior; create an architecture; evolve the implementation; manage postdelivery evolution. Booch says that Figure 1, “represents the activities of the entire development team on the scale of weeks to months at a time.” Within the figure, OOA, object-oriented design, and OOP are the middle three activities.
Abelia Corporation offers Lewis Gray's authoritative, on-site courses on the standards that are mentioned in this presentation. For detailed information about our courses, contact lewis@abelia.com.
Here are some key characteristics of Booch’s macro process:

Mac1. Technical managers plan it, not individual developers.

Mac2. It includes standard development practices that apply equally well to non object-oriented development. These include project planning, risk management, reviews, staffing, project organization, release management (including configuration management), testing, reuse, process and product evaluations, documentation, installation of the development environment, domain-specific practices (e.g., for real-time systems), and technology transfer.

Mac3. It tends to repeat itself after major product releases.

Mac4. Its basic principle is incremental development, which means that it provides successive enhancements of functionality.

The micro process, shown in Figure 2 is Booch’s explanation of what makes O-O activities O-O rather than some other development method. This figure that Booch says, “represents the daily activities of the individual developer or a small team of developers,” shows the O-O core of OOA and object-oriented design. Booch lists four main O-O activities in Figure 2:

mic1. **Identify classes and objects.** During OOA, this identifies the elements of the development problem. During design, it invents new elements of the solution.

mic2. **Identify class and object semantics.** This step establishes the behavior and the attributes of each abstraction. During OOA, this step allocates responsibility for different software behaviors to different abstractions. During design, it separates the concerns of the separate elements of the design.

mic3. **Identify class and object relationships.** This step clarifies the boundaries and the collaborations between the abstractions. During OOA, this step specifies associations among classes and objects (for example inheritance). During design,
it specifies collaborations among separate elements of the design that form larger design mechanisms. For example, it specifies the clustering of classes into categories and modules into subsystems.

**mic4. Specify class and object interfaces and implementation.** Booch says that there is only a single, primary activity here: to choose the algorithms and the data structures that provide the semantics identified in activity mic2 above. During OOA, this step is used to reveal new classes and objects at the next lower level of abstraction. During design, this step creates tangible representations of the abstractions. As Figure 1 suggests, the micro process commences when analysis commences and repeats in each later macro process activity until coding is finished.

### 1.2 Object-Oriented Analysis Method.

OOA develops the required model of the problem domain. As described by Coad and Yourdon, OOA carries out the following five steps which I have mapped to Booch’s core O-O activities:

1. **Identify objects (mic1).** To identify objects, engineers obtain descriptions of desired software behavior from prospective software users and from experts in the software's domain of application. Engineers look for explanations of the problems the software is intended to solve. They might find the explanations in technical literature about the application domain or perhaps in popular publications like an encyclopedia. According to Coad and Yourdon, nouns in sentences often point to potential objects of interest. Potential objects might include other systems that are mentioned, and devices that are mentioned. Objects may be indicated by events that users remember, or by roles that they play.

2. **Identify structures (mic3).** Engineers identify the classification structure and the assembly structure of the software. The classification structure is where the engineer identifies classes and their members. The assembly structure shows the relevant component parts of each object. For example, the assembly structure of an aircraft might consist of engines and avionics systems, or it might only show engines if the software were an engine maintenance tracking system.

3. **Define subjects (mic3).** This step partitions the notation into chunks for easier comprehension. It is done to reduce complexity in the OOA notation. In the...
unusual case that the development problem is already simple enough that it can be modeled with only a few symbols on the diagrams, subjects are not needed.

4. Define attributes (and instance connections) (mic3). Engineers define the attributes, which are the data elements that are used to describe an object or class. They also define the instance connections between objects and classes, which show relationships between objects, and their multiplicity (for example, one-to-many, many-to-many), and whether participation in the relationship is mandatory or optional.

5. Define services (and message connections) (mic2, mic4). Engineers define the processing associated with the objects and classes and the message connections. Coad and Yourdon describe a message connection as a mapping of one object instance (in the executing software) with another in which a sender sends a message to a receiver to have some processing done.

1.3 Object-Oriented Analysis Records.

Coad and Yourdon’s OOA produces diagrams that model the desired software behavior in five major layers corresponding to the five major OOA activities:

1. a subject layer that shows the subjects -- Figure 3 is a sample subject layer

2. an object layer that shows the objects and clusters them by subject

3. a structure layer that shows the objects and the classes and assembly structures in which they participate, and clusters them by subject -- Figure 4 is a sample structure layer

4. an attribute layer that shows the objects and classes and their attributes, and the instance connections between the objects and classes, and clusters them by subject

5. a service layer that shows the objects and classes and the processing services they provide, and the message connections between the objects and classes, and clusters them by subject -- Figure 5 is a sample service layer.

1.4 Object-Oriented Design Method.

In object-oriented design, Booch’s four O-O activities invent the software components that make up the solution domain. The components provide the behavior that the software model from OOA requires. Booch describes three major steps of object-oriented design:
1. Architecture planning (mic1, mic2, mic3, mic4). This includes creation of a logical decomposition of the planned software into classes and objects, and creation of a physical decomposition of the software into software modules that are allocated to target processors. Booch argues that this step must early on create a, “domain-specific application framework,” that can be refined successively into the delivered software product.

2. Tactical design (mic2, mic3, mic4). In tactical design, engineers will define the policies that will guide the interactions between the elements of the architecture. Sample policy topics include memory management, error handling, security, computer-human interface, and many domain-specific topics such as transaction processing, and others.

3. Release planning (mic3). This is where engineers and managers decide how the behavior of the software system will be partitioned into separate releases and delivered in increments. If all of the software will be delivered at once, this step is quick. However, if the system is large, it may take too long to deliver any of it if all of it must be developed first. Also, modern systems are often so complex that users cannot define their needs until they have been able to interact and experiment with partial implementations or prototypes. Optional output of this step is a formal plan that identifies the series of architectural releases, the resources assigned to each of them, and the risks.

Booch says that object-oriented design is complete when the architecture has been validated by a prototype and through reviews, when there are designs for the tactical policies, and when there is a plan for how the software will be released and for which software requirements will be allocated to each release.

1.5 Object-Oriented Design Records.

The notation of Booch’s approach to object-oriented design includes:

1. class diagrams, like that shown in Figure 6
2. state transition diagrams
3. object diagrams
4. interaction diagrams
5. module diagrams, like that shown in Figure 7.10
6. process diagrams.

Booch also describes non-graphical specifications that complete the definitions of the entities in the diagrams.

1.6 OOD Summary.

Booch’s macro process and micro process, combined with the development records produced by analysis, design, coding and the other development activities, are OOD. The processes and the records constitute the methodology. I will show in sections 3 and 4 below how the processes and records described above relate to MIL-STD-498 and ISO/IEC 12207.
2. Rapid Application Development (RAD)

James Martin [3] describes RAD as a developmental life cycle. Figure 8 from Martin[11] shows a sample thirteen-week RAD life cycle for a software system equivalent to approximately 110,000 lines of COBOL code. RAD is intended to produce software much faster, with higher quality, and at lower cost than familiar life cycle models like the “waterfall” model. RAD's intended application domain is business information systems.

Martin is the best-known source of RAD development guidelines. Using the approach that I followed for OOD above, I summarize his guidelines below to establish a baseline description of RAD. Then, in sections 3 and 4 below, I show how MIL-STD-498 and ISO/IEC 12207 can be used for software development efforts that follow the guidelines.

2.1 Martin’s Model for RAD.

Before a RAD project begins, two preconditions should be satisfied. First, tools and people to be used for RAD should be selected, and the RAD methodology should be established within the organization. This includes training the people on the tools and on RAD practices.

Second, the end-user organization must have demonstrated adequate commitment to participating as needed in the project. Adequate end-user commitment is a characteristic risk for RAD projects.

When the end-user has committed, project team members are selected from the people who have been chosen for RAD work, and the team members tailor RAD for the project. The Project Manager and other team members commit to using the method, and begin development.

Once a RAD life cycle is under way, it has

Figure 8. RAD Timetable

four sequential phases:

1. Requirements planning (specification)
2. User design (design)
3. Construction (implementation), and
Within the four phases of RAD, no looping or iteration is expected. Also, Martin treats evolutionary development as a technique that is needed only for the most impressive, complex systems. The technique is not a part of RAD itself. So, unlike OOD above, RAD has no built-in tendency to repeat itself. As Martin describes it, RAD is inherently straight-line, requirements-design-code-test-and-you’re-done.

2.2 Requirements Planning Phase.
The major activity in requirements planning is the Joint Requirements Planning (JRP) workshop. It is needed when system requirements are not well known and when they are not obvious. In other cases, the JRP workshop is combined with a design workshop in the user design phase.

Martin’s main goals for this phase, to be met before anyone decides to build the system, are:

1. to correctly establish the requirements for a system (getting them right the first time)
2. to justify the requirements.

The JRP workshop attempts to achieve a consensus about the requirements and their rationale among the participants who may have different backgrounds and different expectations of the new system. In order to meet the goals, the workshop participants must be people with the right mix of knowledge and authority who understand the requirements for the new system well and have the authority to establish them.

A JRP workshop has five stages:

i. **Initiate the requirements planning activity** by establishing who the executive sponsor of the system is and what the scope of the system will be.

ii. **Research the overall objectives of the new system** and the characteristics of the current system.

iii. **Prepare for the workshop** by selecting participants and preparing the material and facilities for the meeting.

iv. **Conduct the workshop.**

v. **Finalize the documentation** of the workshop results and present them to the executive sponsor.

2.3 Requirements Planning Records.
The typical outputs from a JRP workshop are:

1. a list of system objectives
2. details of possible system functions
3. several different representations of the system's process, such as Figure 9\(^2\) and Figure 10\(^3\) from Martin
2.4 User Design Phase.

Martin describes eight major stages of user design:

1. **Prepare for the first Joint Application Design (JAD) workshop.** Martin's goal for each workshop is to enter or update the system's design in an integrated CASE tool that can be used during the construction phase to automatically generate the code to implement the system.

2. **Conduct the workshop,** which includes developing a design and planning for conversion and production. Martin recommends that workshops follow a top-down approach to system design. The workshops are intended to involve the end users who want the system. However, the group of workshop participants should partially overlap the group of participants in the JRP workshop to provide continuity between the first two RAD phases. For example, the JRP workshop leader should be the same as the JAD workshop leader, and the executive sponsor of the system should attend both workshops also.

3. **Complete the output from the first workshop.** At the first workshop, the initial data model and the process design are developed. After the first workshop, prototypes are built.

4. **Establish teams for the construction phase.**
5. Review and refine the design including testing the usability of the prototype.
6. Establish the contents of the project repository to include the outputs of the first JAD workshop and reusable information obtained from outside the project.
7. Conduct the second JAD workshop. At the second workshop, participants review their experiences with the prototypes and enhance the design.
8. Finalize the design.

### 2.5 User Design Records.

Typical outputs from user design include:

1. a system design stored in a CASE tool, for example in the form of entity-relationship diagrams like Figure 11\textsuperscript{14}, function or process decomposition diagrams, state transition diagrams like Figure 12\textsuperscript{15}, data flow diagrams, or in other forms

![Figure 11. Entity-Relationship Diagram](image)

2. a project repository that integrates the project’s planning, requirements, and design information to a sufficient degree that the resulting database can guide automatic code generation during the construction phase

3. lists of user training needs and user documentation needs.

These are presented to the executive sponsor. If they are acceptable, the executive sponsor initiates the construction phase.

### 2.6 Construction Phase.

Martin says that construction should take place within a timebox. A Review Board consisting of the executive sponsor, a user, and a project quality professional controls the development. The Review Board approves the system design prior to entering the timebox and then decides at the end of the timebox whether or not the system should be approved for installation.

This phase begins by extending the user design as needed and it ends by deciding whether or not to install the new system. All together, there are ten stages in this phase:
1. Do a detailed design of the system structure, by extending the process and data modeling and the prototyping that was carried out during the user design phase. This is when the project completes the detailed design of the system's databases. At this time, the project looks for reuse opportunities also.

2. Build transactions. The system is built one transaction at a time from reusable templates. Templates can include such things as specimen screens and dialogs, data structures, objects, and documents. The transactions are coded with a code generator and checked with a symbolic debugging tool.

3. Perform usability tests. Transactions are grouped and checked for usability. It should be clear how to use the application.

4. Perform ongoing integration. Through the project repository, engineers coordinate the design of transactions so they can be smoothly integrated. The engineers generate load modules, and start integration testing.

5. Prepare for cultural changes. First identify the expected changes. They could include changes in policies, or procedures, changes in staffing levels, changes in job content, or other changes. Then, plan the changes and identify responsibilities and a schedule for making them happen.

6. Design and prepare for cutover. Review the alternatives for installing the hardware and for performing data conversion. Go over the procedures for moving the new system to production mode. Design the facilities for cutover.

7. Do integration testing. Finish linking and testing debugged subsystems.

8. Create the physical design. Do physical database design. Determine traffic volumes, and review how to optimize performance. Finish documenting the final design.

9. Examine the system. The Review Board decides whether or not to approve the system for possible installation.

10. Obtain a decision whether to go ahead with installation. The executive sponsor decides whether or not to install the system.

### 2.7 Construction Records.

Typical construction outputs include:

1. an updated system design stored in the project repository
2. code for individual transactions
3. load modules
4. test results
5. a plan for cutover.
2.8 Cutover Phase.

Cutover is the process of putting the system into operation. Martin describes four steps to prepare for it:

1. **Build systems for automatic conversion of existing data**, or for manual conversion of data that cannot be converted to the new system automatically.
2. The project's quality assurance function **performs a quality audit** of the new system to determine whether it is well structured, well documented, and in compliance with applicable installation standards. Defects found in the product are corrected, and other needed corrections are completed, such as changes to job control language.
3. **Conduct final testing of the system** in the production environment.
4. **Train the users**.

With the preliminaries out of the way, cutover has five additional stages:

5. **Set up the production procedures**, for normal systems operation and for restart and recovery, set up the fallback procedures, the security procedures and the audit procedures.
6. **Install the production system environment**.
7. **Perform data conversion**.
8. **Implement the new system in production**, by moving the new system to production mode, possibly in parallel with the old system, and by phasing out the old system as confidence is developed in the new system.
9. **Review the system installation**.

2.9 Cutover Records.

Martin mentions the following outputs of the cutover phase:

1. the final development product
2. test libraries and results
3. a training program for users
4. production procedures
5. converted data
6. system evaluation records, for example, trouble reports, records of adjustments to make.

2.10 RAD Summary.

The five phases of RAD and the records that result from them constitute the RAD methodology. I will show in sections 3 and 4 below how these phases and records relate to MIL-STD-498 and ISO/IEC 12207.
3. MIL-STD-498, "Software Development and Documentation"


3.1 What Does MIL-STD-498 Do?

MIL-STD-498 does what DOD-STD-2167A and DOD-STD-7935A did together, it provides guidelines for developing military software systems of all kinds. The main part of MIL-STD-498 itself is organized around major engineering activities, as Figure 13 shows. Most of these activities will be familiar to people who have used DOD-STD-2167A. They include all of the engineering activities of both OOD and RAD. Once again, as with DOD-STD-2167A, unless it is tailored out each of the engineering activities in the new standard is required. As Figure 14 and Figure 15 show, MIL-STD-498 and DOD-STD-2167A are alike also in the way they organize topics. Appendix topics of DOD-STD-2167A have been incorporated in MIL-STD-498 appendixes in most cases.

![Figure 13. Major Activities in MIL-STD-498](image-url)
MIL-STD-498 has twenty two associated data item descriptions (DIDs) that incorporate nearly all of the data requirements from the seventeen data item descriptions associated with DOD-STD-2167A and from the eleven figures in DOD-STD-7935A that define its data item requirements. Unless the DIDs are tailored, or the engineering activities that call for the information in the DIDs are tailored out, all of the information in all of the DIDs must be recorded during development, although the standard does not require that the information be delivered in separate documents.
Despite the outward similarities of appearance and data requirements between MIL-STD-498 and DOD-STD-2167A, the new standard is very different from the two earlier standards that it replaced. I will show several major differences below. In many presentations on MIL-STD-498 to software engineers and managers across the country, I have noticed that their common reaction to it is that its biggest difference from the older standards is that it offers more options and opportunities than older standards to match it to a particular software development situation, and it does this by putting fewer constraints on software developers.

For many people, the most important consideration in evaluating MIL-STD-498 is that it is more important for users of the standard to be trained in software development and software acquisition than it was with the older standards, because there are more decisions to make with the new standard. It is clear now, since MIL-STD-498, that software process standards for military software development are being written for technically competent managers and developers. They are not intended to be do-it-yourself guides on software development for people in other fields. They are being written like checklists for trained, skilled software developers, software development managers, and software acquirers.

### 3.2 Why is There MIL-STD-498?

As Chair of ACM SIGAda’s working group on software development standards and Ada, and as a member of the joint EIA, NSIA, and AIA working group on the development of MIL-STD-498, I can report from personal experience that as early as 1989 there was deep concern among military software developers that DOD-STD-2167A was not well suited to use with object-oriented methods. This was one of several problems with DOD-STD-2167A that motivated developing a new standard. Radatz [8] and Newberry [9] review some of the others. MIL-STD-498 was intended to correct these problems. Because DOD-STD-7935A was a documentation standard and not a process standard, there were no process-related problems with it.

Table 1 below shows my personal choice of the most important issues with DOD-STD-2167A that were related to OOD and RAD.

<table>
<thead>
<tr>
<th>Issue Number</th>
<th>Issue</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Perceived preference for “waterfall” development model</td>
</tr>
<tr>
<td>2</td>
<td>Compatibility with incremental/evolutionary development models</td>
</tr>
<tr>
<td>3</td>
<td>Dependence on formal reviews and audits</td>
</tr>
<tr>
<td>4</td>
<td>Compatibility with Ada/object-oriented methods</td>
</tr>
<tr>
<td>5</td>
<td>Distinction between requirements and design</td>
</tr>
<tr>
<td>6</td>
<td>Emphasis on preparing documents</td>
</tr>
<tr>
<td>7</td>
<td>Use of CASE tools</td>
</tr>
</tbody>
</table>

Table 1. Issues Related to OOD and RAD Identified While Using DOD-STD-2167A
The characteristics of MIL-STD-498 that result from making these improvements in the older standard make the new standard easily compatible with both OOD and RAD. Let’s look at the improvements individually to see how.

### 3.3 Removing the “Waterfall” Bias

To understand how MIL-STD-498 responds to Issue 1 and Issue 2 above and avoids a bias toward a waterfall software life cycle, it helps to understand the big difference between major activities in DOD-STD-2167A and in MIL-STD-498. I will begin with DOD-STD-2167A. Figure 16 shows the eight major activities in the standard:

1. system requirements analysis/design
2. software requirements analysis
3. preliminary design
4. detailed design
5. coding and CSU testing
6. CSC integration and testing
7. CSCI testing
8. system integration and testing.

![Software development activities (major)](image)

**Figure 16. DOD-STD-2167A Development Process**
Also, Figure 17 shows the six categories of general requirements in the standard. They are, software development management, software engineering, formal qualification testing, software product evaluations, software configuration management, and transitioning to software support. As Figure 18 shows, in DOD-STD-2167A, each category except the last is a component of a major activity. So, system requirements analysis/design (paragraph 5.1) requires some software development management activities (paragraph 5.1.1), some software engineering activities (paragraph 5.1.2), some formal qualification testing (paragraph 5.1.3), and so on. So do the other seven major activities shown in the figure.

Each major activity of DOD-STD-2167A relates to another major activity through work products. For example, software requirements analysis defines engineering and interface requirements that complete the preliminary sets of engineering and interface requirements that were previously defined in system requirements analysis/design. Then preliminary design
allocates the engineering and interface requirements defined in software requirements analysis to the CSCs of each CSCI. This linking of major activities over time, and the fact that each major activity has several categories of component activities, as shown in Figure 18, transforms DOD-STD-2167A’s major activities into phases of development rather than simple development activities. The particular way that these phases are linked gives Figure 16 and Figure 18 their characteristic “waterfall” shape that follows the “waterfall” development model shown in Figure 19.

In striking contrast to the “waterfall”-like links among DOD-STD-2167A’s major activities, MIL-STD-498’s activities are independent of one another. There are no internal links among activities that order them in a “waterfall” sequence and there are no requirements anywhere in the standard that dictate which of them must be executed first, and which next. In fact, as Figure 13 shows, MIL-STD-498 says explicitly, in paragraph 4.1, “major activities may overlap, may be applied iteratively, may be applied differently to different elements of software...need not be performed in the order listed.” While there is similar language in DOD-STD-2167A, other requirements in the old standard conflict with the language and link the activities in a “waterfall” order. On the other hand, MIL-STD-498 activities can be pictured as mixed up in a bag, so to speak, without order as shown in Figure 20. Unless the software acquirer dictates a tailoring of the standard and a model for ordering the engineering activities, the developer is responsible for creating or adopting a software process for their project that is consistent with contract requirements by pulling some or all of the activities out the bag and putting them in an order that suits the development conditions. The developer then describes the process in the project’s software development plan.
MIL-STD-498 includes descriptions of three sample development models. They are based on strategies for acquiring software that appeared in high-level DoD software policy at the time MIL-STD-498 was developed. The three sample models are:

1. Grand Design (“waterfall”), shown in Figure 19 above
2. Incremental, shown in Figure 21
3. Evolutionary, shown in Figure 22.

The standard also provides guidelines to acquirers for choosing a model. The guidelines are generally applicable to not only the three sample models but also to any other collection of development models that are appropriate for the acquirer’s project. MIL-STD-498 requires that some development model be chosen to order the chosen development activities, but it does not dictate, or constrain, or suggest which model should be used. The choice is left to the acquirer, although developers are allowed to suggest a choice.
Finally, MIL-STD-498 describes software development as a series of software builds that implement subsets of the system or CSCI requirements. A Grand Design development has only one build. Incremental and evolutionary developments have more than one.

The great difference between MIL-STD-498 and DOD-STD-2167A that results from great flexibility in ordering activities and from a wide choice of development models removes the “waterfall” bias of the earlier standard.

Looking back to the characteristics of OOD and RAD that I described in paragraphs 1.1 (for example, Mac3, Mac4) and 2.1 above, it is clear that OOD has an inherent incremental or evolutionary development model, while RAD’s model is much closer to a Grand Design. MIL-STD-498 activities may be arranged as needed within these sample models, or in another development model, to precisely suit the project conditions in which the OOD or RAD methodologies are used. Unlike DOD-STD-2167A, MIL-STD-498 creates no obstacles whatsoever to planning and executing this.

### 3.4 Alternatives to MIL-STD-1521B Reviews and Audits

Figure 16 shows that DOD-STD-2167A depends upon MIL-STD-1521B reviews as gates to exit from most of its major activities. For example, a system design review (SDR) closes system requirements analysis/design, and a software specification review (SSR) closes software requirements analysis. This is Issue 3 in Table 1 above. Unfortunately, the requirements in MIL-STD-1521B were not revised when DOD-STD-2167A superseded the earlier, less flexible DOD-STD-2167. Certain out-of-date requirements in MIL-STD-1521B, such as paragraphs 30.1, 40.1, and 50.1.2 that link software development activities in “waterfall” relationships, were incorporated by reference into DOD-STD-2167A with the result that they reinforced the links between DOD-STD-2167A’s major activities and created a strong “waterfall” bias. MIL-STD-498 does not reference MIL-STD-1521B. As Figure 20 shows, MIL-STD-498 defines its own joint reviews as alternatives to the formal reviews in MIL-STD-1521B. By avoiding ties to the review requirements in MIL-STD-1521B, MIL-STD-498 avoided an additional “waterfall” linking problem. As we saw in the paragraph above, this makes the new standard compatible with the incremental or evolutionary development model that is inherent in OOD.

Figure 23 suggests another problem with MIL-STD-1521B reviews and audits. They are complicated, formal affairs that require elaborate preparation and completion. They are often called “dog and pony shows” referring both to the tone of the meetings and to the frequent preoccupation of the meetings with superficial outward characteristics of the software products to be reviewed rather than with the complicated, subtle, engineering decisions that caused the products to be how they are. In contrast, as Figure 23 suggests, MIL-STD-498 calls for smaller, more frequent, less elaborate joint reviews of interim software work products by developers and acquirers as the work products are developed. The acquirer and the developer must decide when such reviews are warranted.

By their nature, joint reviews of the type described in MIL-STD-498 are easier to schedule and carry out than the formal reviews defined by MIL-STD-1521B. This is a major advantage for RAD projects. RAD loses its reason for being when projects are slowed down or stopped for weeks by each formal review, which will happen under the combination of DOD-STD-2167A and MIL-STD-1521B. Imagine trying to develop the equivalent of 110,000
lines of COBOL code in thirteen weeks according to Martin’s timetable in Figure 8 while completing system design review (SDR), software specification review (SSR), preliminary design review (PDR), critical design review (CDR), test readiness review (TRR) and two configuration audits (FCA, PCA) in accordance with MIL-STD-1521B. The MIL-STD-498 approach to joint reviews avoids this problem.

- DOD-STD-2167A relied on MIL-STD-1521B guidelines for holding joint reviews attended by developers and by acquirer reviewers.

- MIL-STD-498 describes joint reviews that are more frequent, smaller, more limited in scope, and individually less influential on the project.

Figure 23. Replacing MIL-STD-1521B Formal Reviews

Figure 24 shows that there are two categories of joint reviews in MIL-STD-498, joint technical reviews and joint management reviews. In addition, audits are defined in the software configuration management activity. Between them, joint reviews and audits, they can accomplish everything that MIL-STD-1521B’s formal reviews were intended to accomplish, and they can do it quicker, with more accuracy than the older approach. This new approach to joint reviews in MIL-STD-498 is compatible with the RAD methodology while the approach to reviews in older standards was not.

- Joint Technical Reviews collect people with technical knowledge of the software products to be reviewed.

- Joint Management Reviews collect "persons with authority to make cost and schedule decisions."

Figure 24. Categories of Joint Reviews in MIL-STD-498
3.5 Compatibility with Non-Hierarchical Software Design Methods

In 1990, my SIGAda working group on software development standards published SIGAda [10] to document many problems with the software design requirements in DOD-STD-2167A and its predecessor DOD-STD-2167. The problems were discovered first by Ada developers doing OOD. This is Issue 4 in Table 1 above. The core cause of the problems was that both standards required software developers to decompose their CSCIs into computer software components and low-level units that were hierarchically related to one another. This position was a formal requirement in paragraph 4.2 of the earlier standard. In DOD-STD-2167A it was understood through reference to the diagram in the standard that showed a CSCI decomposition (the diagram is Figure 3 in DOD-STD-2167A). Figure 25 shows the problem diagram and indicates that the decomposition requirements in the older standards do not appear in MIL-STD-498.

The decomposition requirement in DOD-STD-2167A caused a serious translation problem. Figure 26 diagrams it. The problem is this: regardless of how components of a software design are actually related to one another, for example in a network or in a star centered on a database, the design must be presented in a software design document (SDD) as though the components are hierarchically related. Often, components of object-oriented designs are not hierarchically related to one another. Also, it often happens that classes participate in one set of relationships (classification relationships, shown by a class diagram like Figure 6) and modules in another (assembly relationships, shown in module diagrams like Figure 7). So no single hierarchical structure describes both. Object-oriented designs often have to be translated into a form consistent with the DOD-STD-2167A diagram. This step is time consuming and error-prone. It often obscures the real engineering decisions that produced the design. The frequent need for translation caused several design methodologists, for example Grady Booch at Rational, to develop mappings between the designs produced by their methods and the elements of

MIL-STD-498 has solved the translation problem. The result is suggested by Figure 27. The new standard calls for developers to record the actual components of their software designs, whatever they may be and however they may be related, hierarchically or not. The new standard
uses the general term ‘software unit’ for an element of the design of a CSCI. Unlike the computer software components and low-level units in the older standards, MIL-STD-498’s ‘software unit’ is a name for the actual elements that the developer uses to construct designs, it is not a type of design element in itself. Also, despite the word ‘software,’ ‘software unit’ is the name for an element of design, not an element of software as Figure 28 shows. So, ‘software unit’ does not appear in MIL-STD-498’s software design records, the actual design elements that they represent do. Joint reviews don’t talk about ‘software unit,’ they talk about software designs. The biggest benefit of this approach for most people is that they don’t have to know anything at all about MIL-STD-498 or its software units to correctly explain their software designs. The new approach is compatible with any design method and any relationship between design elements. Thus, unlike the older standards, it is easily compatible with the design activities of OOD.

3.45 **Software unit.** An element in the design of a CSCI…

- 5.6.2 "The developer shall define and record the architectural design of each CSCI (identifying the software units comprising the CSCI, their interfaces, and a concept of execution among them) and the traceability between the software units and the CSCI requirements…"

- 5.6.3 "The developer shall develop and record a description of each software unit…"

3.6 **Clearer Distinction Between Requirements and Design**

It has been common in software development practice to say that requirements are descriptions of what the software does, while design describes how the software does it. This distinction has always been somewhat confusing, and the confusion has caused rework and at least a few engineering change proposals that would not have been needed otherwise. This is Issue 5 in Table 1.

With OOD and RAD, the common distinction between requirements and design, the “what” versus the “how,” can lead to unproductive thrashing between requirements records and design records as information is first classified a requirement, then reclassified a design decision, then reclassified a requirement, with no clear test available to decide conclusively what it really is. OOD and RAD are especially susceptible to such problems because they both blend their requirements analysis activities and their software design activities. For example, OOD follows...
the same micro process in paragraph 1.1 above for OOA and for object-oriented design. First, identify classes and objects. Then identify class and object semantics, and so on. How is one to know conclusively which classes and objects belong to the development problem and which belong to the solution? RAD faces similar questions about which decisions made at the JRP and JAD workshops described in paragraph 2.2 and paragraph 2.4 above define requirements and which define design decisions. There are overlaps between the two workshops’ members, that is designers attend the requirements workshop. How is one to know conclusively whether the designers have slipped in a design decision as a requirement, or perhaps misunderstood a requirement as non-binding design advice?

In MIL-STD-498, a requirement is defined to be a characteristic that the acquirer insists the system or software must possess if it is to be acceptable. On the other hand, a design characteristic is one that is chosen by the developer. Any system or software characteristic may be either a requirement or design or neither depending upon whether the acquirer insists upon it, or the developer chooses it, or neither is true. These definitions in MIL-STD-498 provide a simple operational test for identifying requirements. Because both OOD and RAD blur together requirements analysis and design activities, and because it is critical to keep track of requirements in contractual situations, these definitions in MIL-STD-498 are likely to make it easier to use OOD or RAD for military software development.

### 3.7 Better Use of Documentation and CASE Tools

Many software developers and acquirers would agree that the typical types and quantity of documentation required by military software development projects are not right. That is Issue 6 in Table 1. Developers and acquirers might not agree on what is wrong though. Developers often complain that acquirers expect them to produce too many paper documents, including documents that are useless. Acquirers complain that developers are unwilling to record enough information about their work and their software products for acquirers to track development progress in a contract period, and for maintenance programmers to understand and change the software products after delivery.

Both developers and acquirers say that they want to automate more of the software development process, but both saw DOD-STD-2167A as a barrier because the standard emphasizes the production and delivery of many paper documents. This is Issue 7 in Table 1.

RAD projects carefully control the amount of time they spend on documentation, and they make heavy use of CASE tools. RAD workshops for developers and users satisfy some of the acquirers’ needs for documentation and review. These practices are designed to achieve quick development times and high productivity.

What is essential for RAD is also attractive to many developers and acquirers. While presenting MIL-STD-498 to over a thousand developers, development managers, and software acquirers in the United States, Canada, and Europe, I have heard from them that they prefer to limit software documentation to three kinds of records suggested by Figure 29: first the software products themselves in various stages of evolution in CASE tools; second, personal data about input from users, prototypes, design and programming goals and status, test plans and test software, test results, design and coding alternatives checked and not checked -- in short, the kinds of personal information that developers collect while they do requirements analysis,
design, coding and testing, that might appear in a software development file (unit development folder); third, software development plans, possibly on-line, where they can find their likely task assignments, what resources will be available to them, what the project schedules are, and deadlines, what the software products will be, and so on.

Figure 29. How Ada Software Developers Want to Document Their Software

MIL-STD-498 responds to all of these concerns:
1. It discourages the acquisition of excessive paper documentation.
2. It requires that engineering work be done and recorded whether or not paper documents are delivered.
3. It encourages the use of CASE tools as alternatives to traditional documents.
4. It defines joint reviews as alternatives to traditional document and review cycles based on MIL-STD-1521B.

Here is how MIL-STD-498 does this. First, the standard clearly states in paragraphs 5.1.1 and H.3 that producing documents is recognized to be, “separate from the task of generating and recording the required [engineering] information and to require additional time and effort on the part of the developer...time and effort that would otherwise be spent on the engineering effort.” The guidelines in the standard for when to acquire documents read, “deliverables should be ordered only when there is a genuine need to have this planning or engineering information transformed into a deliverable.”

Second, in its engineering requirements, MIL-STD-498 tasks developers to develop and record information. It does not specify where to record the information, or how to record it. The engineering requirements in MIL-STD-498 (5.1.1) point to DIDs as checklists of information that must be developed and recorded whether or not the documents they describe are ever delivered. The standard says in paragraph G.5, “...the actual software product is the information called for by the DID, not necessarily in the form of a hard-copy document.”

Third, in several places, for example in paragraph 1.2.4.4, in paragraph 5.1.1, in paragraph H.5, and in section 7 of each DID, MIL-STD-498 directs developers to consider recording information in CASE tools rather than in traditional paper documents.
Fourth, in paragraph H.3, MIL-STD-498 states that it includes requirements for joint reviews specifically so that the results of planning and engineering work can be reviewed in their natural, technical form without the need for deliverables.

MIL-STD-498’s treatment of documentation and CASE tools is more suitable for most development methods than requirements in older standards like DOD-STD-2167A. For example, the new standard accommodates all of the OOD records and RAD records described in paragraphs 1.3, 1.5, 2.3, 2.5, 2.7, and 2.9 above, in their natural, technical form, without the need to transform them into additional deliverables. MIL-STD-498 is the best ever military standard for RAD where sophisticated integrated CASE tools play a critical role. But, there is a condition to these claims. In spirit, MIL-STD-498 seeks to avoid excessive documentation and to replace familiar paper documents and their kin, like word processing files, with native engineering work products such as O-O models in tools like Rational Rose, or structured models in tools like Software Through Pictures or Teamwork. In principle, MIL-STD-498 is specifically written to avoid the problems suggested by Issues 6 and 7 in Table 1, and it is very compatible with RAD. In practice, acquirers decide what type of documentation to buy and how much. Through poor tailoring, an unskilled acquirer could easily subvert the intent of the standard.

4. ISO/IEC 12207, "Information Technology: Software Life Cycle Processes"

ISO/IEC 12207 is an international standard on software life cycle processes. Singh [12] says that the standard was created to establish an international, common framework for acquiring, supplying, developing, operating, and maintaining software. The standard is intended to contribute to world trade in software.

ISO/IEC 12207 was proposed in 1988, and published in August 1995. Dr. Raghu Singh was its editor. Dr. Singh also managed the DoD project that developed MIL-STD-498. Therefore, it’s not surprising that the two standards are alike in some important ways. There are also important differences between them. First, I will describe ISO/IEC 12207 below. Then, I will show how it solves the OOD-related and RAD-related problems with DOD-STD-2167A. Like MIL-STD-498, the international standard is compatible with both methodologies.

By early in 1997, DoD will replace MIL-STD-498 with a commercial software standard now under joint development by the IEEE and the EIA, with Dr. Singh as IEEE CO-Chair. The planned commercial standard, tentatively labeled EIA/IEEE Trial Use Standard J-STD-016, “Software Life Cycle Processes,” at the time this paper was submitted, will combine engineering and data requirements from MIL-STD-498 with additional requirements from ISO/IEC 12207 on software life cycle processes, like acquisition, operation and maintenance, that are not covered in the military standard. Like its parent standards, J-STD-016 will be compatible with OOD and RAD. I will describe the planned path to the new standard below.

4.1 What Does ISO/IEC 12207 Do?

ISO/IEC 12207 describes the major component processes of a complete software life cycle, their interfaces with one another, and the high-level relations that govern their interactions. Figure 30[18] summarizes its contents. Figure 31[19] shows several different points of
view of the processes. Figure 32 suggests how the component processes are used during a system life cycle.

Figure 30. Technical Contents of ISO/IEC 12207

Figure 31. Views of Software Life Cycle Processes
As Figure 31 suggests, acquirers and suppliers see the contract view of software development. The supply process begins when a contractual relationship to supply software is formed between an acquirer and a supplier. Depending upon the terms of the contract, the supply process may employ the development process to develop new software, the operation process to provide software operation services, or the maintenance process to repair and improve the software. The figure shows also that operators and users, developers and maintainers, and others have their own distinctive views of the life cycle.

Figure 30 shows that organizations acquire software through projects that participate in contracts (usually) and carry out the activities and tasks associated with acquisition, supply, development, operation, and maintenance. In doing this, projects carry out processes for joint reviews, audits, quality assurance, and verification and validation. They also execute processes of documentation, configuration management, problem resolution, and tailoring from time to time. Meanwhile, the organization manages the project, providing the necessary development infrastructure and training for project members, and improving the project’s software process. These component processes and their relationships in Figure 30 provide a high-level summary of the technical content of ISO/IEC 12207.

To compare the scope of ISO/IEC 12207 to that of MIL-STD-498, compare Figure 13 above to Figure 31. The following major activities in MIL-STD-498 (Figure 13) correspond to processes in ISO/IEC 12207 (Figure 31):

1. Software configuration management (configuration management, audit)
2. Software quality assurance
3. Software product evaluation (verification, validation)
4. Joint technical and management reviews
5. Corrective action (problem resolution)
6. Project planning and oversight (management)
7. Establishing a software development environment (infrastructure).

Also, there are activities in MIL-STD-498 that correspond to the documentation and improvement processes in ISO/IEC 12207. Finally, all of the following major activities in MIL-STD-498 correspond to only a single process in ISO/IEC 12207, the development process:

8. System requirements analysis
9. System design
10. Software requirements analysis
11. Software design
12. Software implementation and unit testing
13. Unit integration and testing
14. CSCI qualification testing
15. CSCI/HWCI integration and testing
16. System qualification testing
17. Preparing for software use
18. Preparing for software transition.

As the comparison shows, there is a striking difference between the scope of ISO/IEC 12207 and the scope of MIL-STD-498. There are no activities in MIL-STD-498 that correspond specifically to the acquisition process, the supply process, the operation process, the maintenance process, or the training process in ISO/IEC 12207. This is a very big difference between the two standards.

Another big difference between ISO/IEC 12207 and MIL-STD-498 is that the requirements in the international standard are at a much more general level than those in the military standard. In fact, ISO/IEC 12207 says in paragraph 1.5 that it, “describes the architecture of the software life cycle processes but does not specify the details of how to implement or perform the activities and tasks included in the processes.” Here is an example: MIL-STD-498 and its DIDs contain 117 pages of engineering and data requirements related to major activities eight through eighteen above that correspond to the development process in ISO/IEC 12207 -- the international standard contains less than seven pages of requirements on that process! Unlike MIL-STD-498, ISO/IEC 12207 has no DIDs associated with it (yet). In most cases, the requirements in MIL-STD-498 are compatible with the corresponding requirements in ISO/IEC 12207, just more detailed. Like MIL-STD-498, ISO/IEC 12207 is not intended to be a do-it-yourself guide on how to develop software. It was written for trained, skilled software developers, software development managers, and software acquirers.
4.2 The “Waterfall” Bias

ISO/IEC 12207 responds to Issue 1 and Issue 2 in Table 1 above as MIL-STD-498 does. The two standards do differ though in what their development activities are. Like in DOD-STD-2167A, the development process described in ISO/IEC 12207 consists of activities that bundle together tasks that would have their own separate activities if the standard were MIL-STD-498. For example, in the list that follows, “eval” indicates a bundled product evaluation task, “revs” indicates a bundled joint review task, “cm” indicates a bundled software configuration management task, “user docs” indicates a bundled task from preparations for software use, and “test” indicates a bundled testing task. The twelve engineering activities are listed below and summarized in Figure 33:\textsuperscript{21}

1. system requirements analysis (eval)
2. system architectural design (eval)
3. software requirements analysis -- concludes with successful reviews followed by the establishment of a baseline for the software requirements (eval, revs, cm)
4. software architectural design (user docs, test, eval, revs)
5. software detailed design (user docs, test, eval, revs)
6. software coding and testing (user docs, eval)
7. software integration (user docs, eval, revs)
8. software qualification testing -- concludes with successful audits followed by the establishment of a baseline for the software design and code (user docs, eval, cm)
9. system integration (eval)
10. system qualification testing -- concludes with successful audits followed by the establishment of a baseline for the design and code of each software configuration item (eval, cm)
11. software installation
12. software acceptance test.

In my opinion, bundling together tasks of different kinds in this way creates development phases regardless of whether the phases are called phases or called activities. Even so, ISO/IEC 12207 stops short of linking the activities internally as DOD-STD-2167A does. The activities of the ISO/IEC 12207 development process have the independence from one another that the activities in MIL-STD-498 have. As in MIL-STD-498, they are not ordered in a “waterfall” sequence and, in my opinion, there are no requirements in the international standard that dictate which of them must be executed first, and which next. In fact, ISO/IEC 12207 says explicitly, in paragraph 5.3.1.1, in language similar to that in MIL-STD-498, “these activities and tasks may overlap or interact and may be performed iteratively or recursively.”

In paragraph 1.5, ISO/IEC 12207 states that, “this international standard does not prescribe a specific life cycle model or software development method.” In language similar to that in MIL-STD-498, paragraph 5.3.1.1 states that unless the contract stipulates one, “the developer shall define or select a software life cycle model appropriate to the scope, magnitude,
and complexity of the project. The activities and tasks of the development process shall be selected and mapped onto the life cycle model.” The intent and effect of the language in the international standard are to provide flexibility in ordering activities and in choosing development models, to avoid the “waterfall” bias of other standards. So, the flexibility and the choices can be used to match a development model to the OOD or RAD methodologies. Although it is more obvious with MIL-STD-498 than with ISO/IEC 12207 that it is suited to use with OOD and RAD, the requirements in the international standard are fully compatible with the characteristics of the two methodologies that I described in paragraphs 1.1 and 2.1 above.

### 4.3 Reviews and Audits

I showed in paragraph 3.4 above that DOD-STD-2167A depends upon MIL-STD-1521B reviews as gates to exit from most of its major activities. This is Issue 3 in Table 1 above. Then I showed that MIL-STD-498 calls for smaller, more frequent, less elaborate joint reviews of interim software work products by developers and acquirers that are easier to schedule and carry out than the formal reviews defined by MIL-STD-1521B. This is a major advantage for RAD projects.

Like MIL-STD-498, ISO/IEC 12207 avoids “dog and pony” shows and defines two categories of joint reviews, joint technical reviews and joint management reviews. They are very similar to the reviews defined by the military standard. Audits are defined as a separate process in ISO/IEC 12207. Between them, joint technical reviews, joint management reviews, and audits, they accomplish everything that MIL-STD-498’s review and audits accomplish, and they are compatible with the RAD methodology.

![DEVELOPMENT PROCESS](image)

Figure 33. The ISO/IEC 12207 Development Process
4.4 Non-Hierarchical Software Designs

I said in paragraph 3.5 above that serious problems have been discovered with the software design requirements in DOD-STD-2167A and its predecessor DOD-STD-2167, especially when doing OOD. This is Issue 4 in Table 1 above. I claimed that the core cause of the problems was that both standards required software developers to decompose their CSCIs into computer software components and low-level units that were hierarchically related to one another. The problems have mostly gone away for new military software development because MIL-STD-498 dropped the offending requirements in the earlier standards.

ISO/IEC 12207 avoids the bad requirements in DOD-STD-2167 and DOD-STD-2167A also. Although the approach to software design in ISO/IEC 12207 is somewhat different from that in MIL-STD-498, the effect on the problems related to Issue 4 is the same, they have gone away. The international standard requires developers to create software components and to record their design. The requirements in the standard are deliberately simple and brief. Good designers will follow the requirements even though they may never know anything at all about them, regardless of what design method the designers use. The approach in ISO/IEC 12207 is compatible with any design method and any relationship between design elements, as the approach in MIL-STD-498 is. So, the requirements in ISO/IEC 12207 are compatible with the design activities of OOD.

4.5 Requirements vs. Design

I said in paragraph 3.6 above that it is confusing to distinguish requirements from design by saying that requirements are descriptions of what the software does, while design describes how the software does it. This is Issue 5 in Table 1 above.

Unlike MIL-STD-498, ISO/IEC 12207 does not attempt to clarify the difference between requirements and design. Although OOD and RAD tend to blur together requirements analysis and design activities, and although it is critical to keep track of requirements in contractual situations, requirements can be well tracked by projects that use these methodologies under ISO/IEC 12207, although probably not as easily as under MIL-STD-498.

4.6 Documentation and CASE Tools

Paragraph 3.7 above points out that many software developers and acquirers would agree that the typical types and quantity of documentation required by military software development projects are not right. That is Issue 6 in Table 1. Also, both developers and acquirers say that they want to automate more of the software development process, but both saw DOD-STD-2167A as a barrier because the standard emphasizes the production and delivery of many paper documents. This is Issue 7 in Table 1.

RAD projects use practices designed to achieve quick development times and high productivity through heavy use of sophisticated, integrated CASE tools. These projects are very vulnerable to excessive documentation requirements.

Like MIL-STD-498, ISO/IEC 12207 responds to these concerns:

1. It has no data item descriptions (DIDs).
2. It requires that engineering work be done and recorded whether or not paper documents are delivered.

3. It is neutral on the use of CASE tools as alternatives to traditional documents.

4. It defines joint management and technical reviews that are convenient alternatives to traditional document and review cycles based on MIL-STD-1521B.

Here is how ISO/IEC 12207 does this. First, paragraph 1.5 of the standard states its intent not to describe data items when it says, “this international standard is not intended to prescribe the name, format, or explicit content of the documentation to be produced.” ISO/IEC 12207 has no DIDs. Since the standard has no DIDs, acquirers are less likely than with earlier standards like DOD-STD-2167A to accidentally incorporate excessive documentation requirements in contracts.

Second, ISO/IEC 12207 explains in paragraph 4.1.1.2 and paragraph 6.1 that, “the documentation process is a process for recording information produced by a life cycle process or activity.” The information to be recorded includes that needed by, “all concerned such as managers, engineers, and users...” Implementation of the ISO/IEC 12207 development process, as described in paragraph 5.3.1, includes documenting (recording) the outputs of all of the activities and tasks of the process. Thus, the results of the engineering work must be recorded. This requirement is not in any way conditional upon whether paper documents, or other documents, must be delivered to the acquirer.

Third, paragraph 1.5 of ISO/IEC 12207 says that, “this international standard, however, does not imply that such documents be developed or packaged separately or combined in some fashion.” Paragraph 6.1.3.1 says that, “production and distribution of documents may use paper, electronic, or other media.” As I showed above, paragraphs 4.1.1.2 and 6.1 describe the documenting process simply as a process of recording information with no indication that the information must be recorded as a traditional document rather than as data in a CASE tool for example. Nevertheless, the bulk of the documentation process requirements within paragraph 6.1 and its subparagraphs suggest that the authors had traditional documents in mind. For example, paragraph 6.1.2.1 says,

“each identified document shall be designed in accordance with applicable documentation standards for format, content description, page numbering, figure/table placement, proprietary/security marking, packaging, and other presentation items.”

In my opinion, the requirements in ISO/IEC 12207 simply allow developers to use data in CASE tool storage or format to document their work, but do not explicitly encourage this as the requirements in MIL-STD-498 do.

Fourth, in paragraph 6.6, ISO/IEC 12207 defines the joint review process flexibly as, “a process for evaluating the status and products of an activity of a project as appropriate. Joint reviews are at both the project management and technical levels and are held throughout the life of the contract.” Here, the standard allows for reviewing the results of planning and engineering work in their natural, technical form.

Like MIL-STD-498 above, ISO/IEC 12207’s treatment of documentation and CASE tools is more suitable for most development methods than requirements in older standards like
DOD-STD-2167A. The international standard accommodates all of the OOD records and RAD records described in paragraphs 1.3, 1.5, 2.3, 2.5, 2.7, and 2.9 above, in their natural, technical form, without the need to transform them into additional deliverables. ISO/IEC 12207 has no required deliverable data. ISO/IEC 12207 is a suitable standard for RAD where sophisticated integrated CASE tools play a critical role. In its untailored form, it poses less of a risk of creating accidental, excessive documentation requirements in contracts than MIL-STD-498 does.

### 4.7 The U.S. Path to the Use of ISO/IEC 12207

The pattern of similarities and differences between ISO/IEC 12207 and MIL-STD-498 that I described in paragraphs 4.1 through 4.6 above suggests possible benefits from combining the two standards into a single, more comprehensive standard than MIL-STD-498 that has much more development detail than ISO/IEC 12207. In fact, that is happening now, and the combined standard, currently labeled J-STD-016, is intended to supersede MIL-STD-498 in less than a year. Figure 34 suggests the path ahead.

![Figure 34. Implementing ISO/IEC 12207 in the U.S.](image)

On June 29, 1994, Secretary of Defense William J. Perry's memorandum on, "Specifications and Standards -- A New Way of Doing Business," directed the DoD to use commercial software engineering standards in preference to military standards. When the memo was released, DoD and industry had almost finished developing a new standard that fixed the problems with DOD-STD-2167A suggested by the Issues in Table 1 above, and harmonized DOD-STD-2167A with DOD-STD-7935A. Since the memo, the DoD has been steadily moving out of the standards business. Nevertheless, the new standard was approved, and released on
December 5, 1994, as MIL-STD-498, because there was no commercial standard that could serve as an adequate replacement for DOD-STD-2167A. As part of the conditions for its approval and release, DoD limited MIL-STD-498 to a two-year life which will expire in December 1996. The goal of limiting its life in this way was to give industry time and motivation to develop an adequate commercial standard to replace it.

Anticipating the release of MIL-STD-498, the IEEE Software Engineering Standards Committee began a joint project with the Electronic Industries Association (EIA) in November 1994 to prepare a commercial replacement for it to be called IEEE Std 1498/EIA IS-640. I describe the history of this project here from my perspective as Co-Chair of the EIA committee that coordinates industry participation in the project. Project Co-Chairs are Dr. Raghu Singh, representing IEEE, and Perry R. DeWeese from EIA. Logicon in San Diego was hired to produce the document. On November 30, 1994, the project released the first draft of the commercial standard for review and comments. By July 1995, the project had produced a stable trial-use version which is available for use now. It is planned that an electronic copy will be available via anonymous ftp from “ftp://glider.logicon.com/pub/standards”.

In the Spring of 1995, looking beyond its initial goal to simply develop an industry replacement for MIL-STD-498, the IEEE/EIA project decided to accelerate the achievement of a second goal which was to develop a standard that would both replace MIL-STD-498 and implement the international standard on software life cycle processes, ISO/IEC 12207. The project’s plan was revised to schedule delivery of this enhanced commercial standard no later than December 1996 to coincide with the planned demise of MIL-STD-498. Recently, the enhanced standard has been labeled EIA/IEEE Trial Use Standard J-STD-016, “Software Life Cycle Processes,” or J-STD-016 for short.

When I submitted this paper, the implementation details of J-STD-016 had not been finalized. Nevertheless, it is very likely that the new standard will combine a modified version of the engineering and data requirements in MIL-STD-498 with the requirements in the international standard ISO/IEC 12207 to produce a standard that is much broader in scope than MIL-STD-498 and simultaneously much deeper in development detail than ISO/IEC 12207. All of the technical goodness of MIL-STD-498 is planned to survive in the new combination standard. It is planned that the benefits of investments in learning and adapting to MIL-STD-498 will transfer to J-STD-016. Thus, J-STD-016 will work as well with OOD and RAD as the best of either MIL-STD-498 or ISO/IEC 12207. Ideally, J-STD-016 users will also be compliant with ISO/IEC 12207 which is expected to make it easier for them to sell their software in international markets if they should ever choose to do that. It is planned that draft versions of J-STD-016 will be available via anonymous ftp from “ftp://glider.logicon.com/pub/standards/US12207”.

Summary

This paper has answered the questions, "How do OOD, RAD and MIL-STD-498 relate to one another and to ISO/IEC 12207?" and "Will the planned replacement for MIL-STD-498, tentatively titled J-STD-016, be an adequate replacement for MIL-STD-498 in the high-technology areas of OOD and RAD?"

By showing how OOD and RAD are supported by MIL-STD-498 and ISO/IEC 12207, the paper described both standards and explained key differences between them. It concluded
with a look into the near future at probable requirements in the planned combination of the two standards, J-STD-016, which is scheduled to supersede MIL-STD-498 within a year.

**Lewis Gray**

Lewis Gray is the President of API. He coaches software process improvement and consults on software engineering standards.

He is Co-Chair of the EIA committee that is developing EIA/IEEE Trial Use Standard J-STD-016, “Software Life Cycle Processes,” based on ISO/IEC 12207 to replace MIL-STD-498. He is Chair of ACM SIGAda’s working group on software development standards and Ada, and he was a member of the joint EIA, NSIA, and AIA working group on the development of MIL-STD-498. He has led or participated in military software development in the Ada programming language and in process improvement efforts in large and small companies.

Prior to founding API, he held key management or technical positions at TRW, GTE, and INTELLIMAC, Inc.

Dr. Gray earned the Ph.D. degree from Indiana University in the philosophy of science/technology assessment.

Dr. Lewis Gray  
API (Ada PROS, Inc.)  
President  
12224 Grassy Hill Court  
Fairfax, Virginia 22033-2819 USA  
Voice: 703-591-5247 Fax: 703-591-5005  
Internet: adapros@aol.com

**References:**


---

1Booch [1], page 248.
2Ibid, pages 248 - 249.
3Ibid, pages 267 - 286.
5Coad and Yourdon [2], page 99.
6Ibid., page 90.
7Ibid., page 154.
8Booch [1], page 256.
9Ibid, page 457.
10Ibid, page 353.
11Martin [3], page 500.
15McClure [4], page 89.
16Singh [12], slide 6-11.
17Ibid, slide 6-12.
18Ibid, slide 5-23.
19Ibid, slide 6-2A.
20Ibid, slide 6-6.
21Ibid, slide 5-4.