QRA

Aerospace Requirements Guide & Checklist

The 10 Essential Best Practices for Assuring do-178C Compliance

DO-178C doesn't specify a specific software process. Instead, it creates a flexible development framework designed to lead to system certification by relevant authorities. DO-178C specifies software lifecycle process objectives, along with activities for meeting those objectives. It also provides guidance for tailoring process objectives and activities to the level of safety the software must provide and for collecting evidence to show the process objectives have been met.

INTRODUCTION

Software Considerations in Airborne Systems and Equipment Certification, better known as DO-178C, has in recent years become the de facto standard for avionics software development.

As its title implies, DO-178C doesn't specify a specific software process. Instead, it creates a flexible development framework designed to lead to system certification by relevant authorities. DO-178C specifies software lifecycle process objectives, along with activities for meeting those objectives. It also provides guidance for tailoring process objectives and activities to the level of safety the software must provide and for collecting evidence to show the process objectives have been met.

While DO-178C focuses on the software development process, it has implications at the system level, as well. In particular, the software requirements process is directly impacted by the system requirements process, which dictates the high-level software requirements.

This guide describes ten requirements engineering (RE) best practices aerospace organizations can apply to help assure their avionic software complies with DO-178C. The accompanying checklist is meant to help those organizations embed these best practices, both within their RE process and in the minds of their engineers.

1. DOCUMENT YOUR PROCESS FOR REQUIREMENTS ANALYSIS AND REVIEW

Paragraph 5.1 of DO-178C provides guidance for the software requirements process. It's first two recommendations are:

- "The system functional and interface requirements that are allocated to software should be analyzed for ambiguities, inconsistencies and undefined conditions."
- "Inputs to the software requirements process detected as inadequate or incorrect should be reported as feedback to the input source processes for clarification or correction."

Since DO-178C focuses on building software processes that assure adequate safety, it's a good idea to have solid, well-documented system-level RE processes feeding your software processes. You should be able to show certification authorities you have standardized, repeatable processes that comply with their standards.

Your requirements analysis process documentation should provide an overview of the process and a description of each step. Describe the steps in terms of entry and exit criteria, procedures, tools to be used in your analysis, and any data or reports that are to be produced.

Further recommendations for this process will be provided throughout the remainder of this guide.

2. DEFINE YOUR METHODS FOR ASSURING REQUIREMENTS TRACEABILITY

To comply with DO-178, your software requirements and design processes must demonstrate traceability. High-level software requirements must trace to system requirements. Low-level software requirements to high-level requirements, and so forth.

It's important to plan how you will do this and to be able to show how you do it.

Decide what tool or tools you are going to use to maintain and demonstrate traceability. All commercially available requirements management (RM) tools have facilities for this. If you choose such a tool, be sure you understand its traceability mechanisms.

If you use a custom-built database or document-based RM system, your organization must define its own requirements traceability system. Typically, this is done by assigning a "unique identifier" number or code to each requirement and building tables or matrices that demonstrate the traceability of each requirement —both upward to its original source requirement and downward to the verification process.

3. DEFINE YOUR CRITERIA FOR EVALUATING REQUIREMENTS

Another DO-178C "activity" (or requirement), from paragraph 5.1.2, drives several of the best practices in this document: "The high-level requirements should conform to the Software Requirements Standards and be verifiable and consistent."

To assure that your requirements are consistent, you need to define your criteria for evaluating requirements.

These criteria should include rules for the use of imperatives like shall, will, must and should—which of these are allowed and what each means in the context of the requirements document. Your criteria will also specify:

- The form and placement of unique identifiers in requirement statements
- Any templates to be used in forming requirement statements
- Words to avoid or to use with caution due to their tendency to introduce ambiguity
- How rationale and other explanation should be separated from the requirement statement

If you use a requirements analysis software tool, it will likely come initialized with a default set of evaluation criteria. You should adjust these settings to match your organization's own criteria.

You may need to reference applicable documents. For example, your organization might decide to base your analysis criteria on those listed in the INCOSE Guide for Writing Requirements. Or you might adopt the Easy Approach to Requirements Syntax (EARS) as templates for your requirement statements. If so, you should reference those documents in your process documentation.

Some of the best practices that follow provide additional criteria for evaluating requirements.

4. STATE REQUIREMENTS IN VERIFIABLE, QUANTITATIVE TERMS

In paragraph 5.1.2, DO-178C warns: "High-level software requirements should also be stated in quantitative terms with tolerances where applicable." This is just a more detailed way of saying requirements should be verifiable.

To be verifiable, functional requirements need to meet two criteria.

First, functional requirements should be stated in terms of the inputs and outputs of the system being specified. System inputs and outputs are quantities, therefore quantifiable. This "black box" approach permits verification by testing and other methods, while leaving developers free to design their software as they see fit. To verify compliance with such black box requirements, testers need only apply the specified input quantities and compare the actual outputs with those specified.

Secondly, as stated in the DO-178C requirement above, tolerances should be specified where applicable. This applies not only to input and output quantities but also to system reaction times, as the latter need to account for data transmission rates and latencies.

For example, if your Engine Monitor Unit (EMU) must set a specific bit in a specific MIL-STD-1553B mux bus message any time a certain analog input exceeds a certain value, you'll need to include tolerances for (1) how long the input threshold needs to be exceeded (to account for how frequently the EMU reads the analog input and, perhaps, to filter out some harmless noise in the signal), and (2) the tolerance for latency between the input threshold being exceeded and the bit being set on the bus (to account for the frequency at which the bus message is transmitted).

If the EMU's output message is transmitted every 20 milliseconds (ms), such a requirement might take the following form: If Unit_Overtemp_Analog exceeds 250° C for more than 3 seconds, the EMU shall set the Unit Overtemp bit (Msg 04, Word 1, Bit 14) to 1 within 20 ms. A second requirement specifying when the Overtemp bit output shall be reset to 0 would also be required.

5. ENFORCE CONSISTENT USE OF UNITS AND TERMS

To ensure requirements can be objectively verified, it's important they be specified in unequivocal terms. Stakeholders must have a mutual understanding of the terms used within the specification and of the units of measurement to be used in requirement expressions.

For example, in a project using the metric system, an input quantity expressed in pounds in a functional or interface requirement might be misinterpreted (and cause an error) in a calculation where the other terms are in kilograms. It's best to express the input quantity and the requirement in the accepted units of the project, rather than expect the software designers to include a conversion from pounds to kilograms.

Related best practices for enforcing consistent use of terms and units include:

- Maintaining a glossary of technical terms used in the project
- Maintaining a list of acceptable units, sorted by quantity type (mass, volume, velocity, etc.)
- Checking requirements against those lists to make sure terms and units are used correctly

Implementing this best practice can be greatly simplified through automation with an analysis tool that tracks accepted terms and units. We'll dive deeper into analysis automation in best practice #8.

6. DO NOT SPECIFY IMPLEMENTATION DETAILS IN FUNCTIONAL REQUIREMENTS

Another DO-178C recommendation is "The high-level requirements should not describe design or verification detail except for specified and justified design constraints." This is a well-known best practice in requirements engineering.

System designers should not be limiting software designers' options any more than necessary. Software developers need freedom to do what's best for the overall project design.

Besides that, including implementation detail in functional requirements creates problems in verification. It clutters requirements with details that can't be verified by checking system response. Functional requirements that don't express their input/response relationships clearly are more likely to be misinterpreted by both software developers and verification engineers.

To give an example, let's say you're stating a requirement on the mission computer (MC) of a multi-role fighter aircraft for timing the release of a new air-to-surface weapon with a unique, multi-stage ballistic profile. You've received the ballistics algorithm for the weapon from your weapons technology group. In your release timing requirement, you should not state details of the algorithm as requirements (e.g., the MC shall do this, then this, then this, etc.). Those internal implementation details would not be testable and may not be the best way to implement the algorithm in software. Instead, simply state that when the requisite pre-conditions and trigger condition inputs exist, the MC shall issue the weapon release command output in accordance with the algorithm specified in [the applicable reference or directive]. This lets you're your software developers and your test team implement the algorithm in whatever manner is most efficient for their respective platforms, and it allows the implementation to be tested objectively.

7. INCLUDE RATIONALE, BUT SEGREGATE IT FROM THE REQUIREMENT STATEMENT

DO-178C specifies that, "Derived high-level requirements and the reason for their existence should be defined." While this statement addresses derived requirements, including rationale to justify a requirement's existence or clarify its meaning is a good idea when needed.

It's also important that rationale and other explanation doesn't detract from the clarity of the requirement. Lack of clarity increases the risk of misinterpretation. Therefore, explanatory text should be segregated from the requirement statement itself.

How? Place the rationale in a subsequent paragraph, separate from the requirement statement. Start the paragraph with a label or prefix, such as 'Rationale:', 'Comment:' or 'Note:', and don't include a unique identifier. The prefix and the lack of an identifier code will clearly separate the rationale from the requirement. This simplifies the requirement statement while clarifying its meaning.

8. AUTOMATE YOUR REQUIREMENTS ANALYSIS PROCESS

When performed manually, requirements analysis has long been a time-consuming and tedious task. As airborne software had grown exponentially in size and complexity over the past decades, the problem has grown exponentially worse.

Traditionally, analysis of requirements written in natural language has been performed manually, in two steps. First, the requirements authoring team pores over the requirements document, often with the aid of a review guide or checklist—the proverbial fine-tooth comb. Then the document undergoes a formal review involving all the relevant stakeholders. Both steps are labor-intensive procedures that have a significant impact on schedule and budget.

Fortunately, this task can now be streamlined using software tools built specifically for analysis of natural language requirements.

Modern requirements analysis tools, like QRA's QVscribe, help engineers and analysts find and correct requirements errors by automatically highlighting questionable uses of imperatives, potentially ambiguous words and phrases, unauthorized units of measurement, and other possible flaws that can lead to misinterpretation.

These tools also generate reports that can be used as artifacts of process fulfillment. Ultimately, they reduce schedule impact and free engineers from the tedious side for requirements analysis, leaving them more time for tasks that truly require their expertise.

To gain a better understanding of how these tools automate the requirements analysis process, consult our free guide: Automating the INCOSE Guide to Writing Requirements.

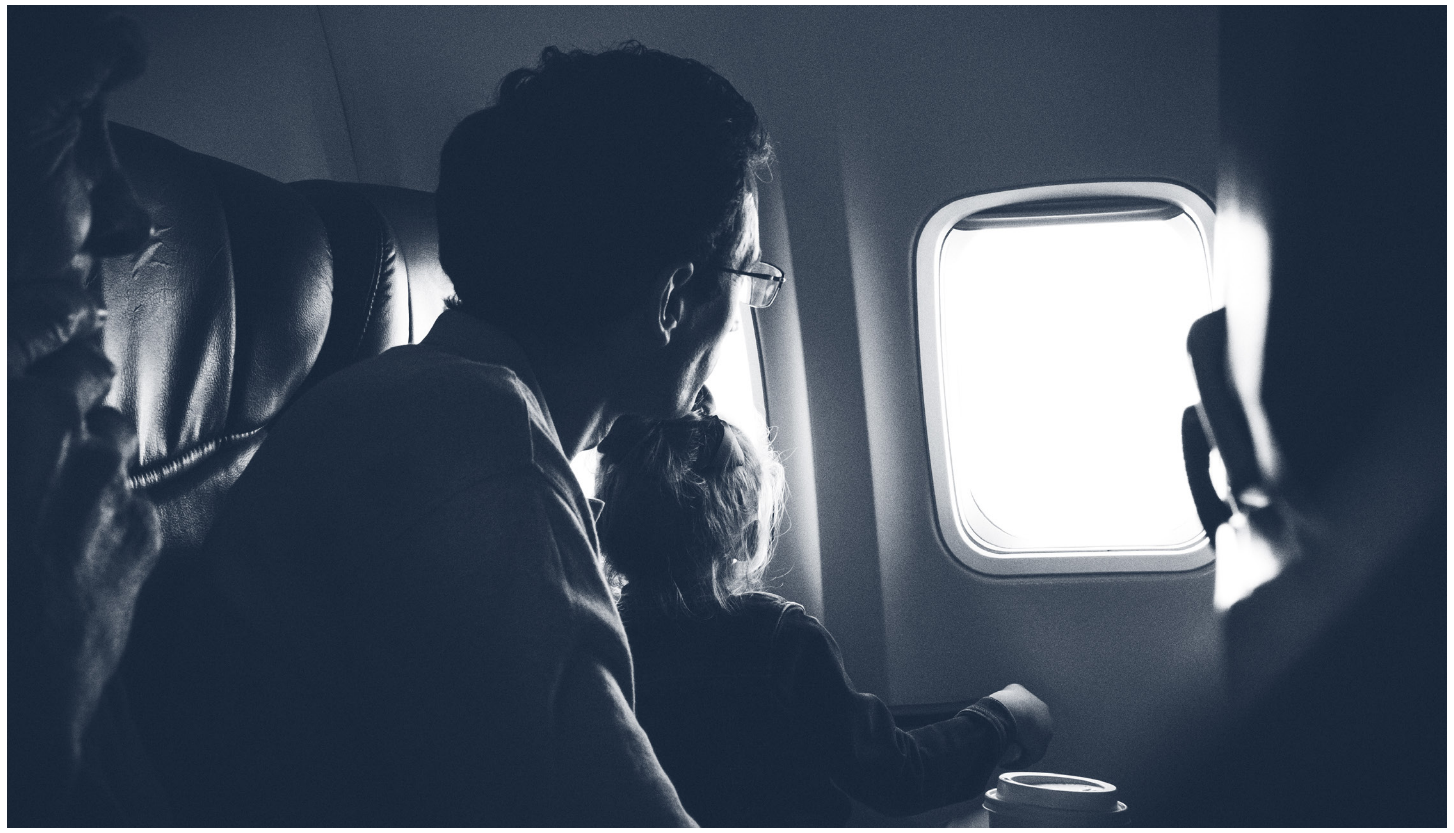
9. ELIMINATE CONFLICTS BETWEEN REQUIREMENTS

For requirements to be consistent, as DO-178C requires, there can be no conflicts between them.

Unfortunately, when large numbers of requirements are elicited from a host of diverse stakeholders then developed and augmented with derived requirements, it's only natural that conflicts arise. Finding and eliminating such conflicts in a large, complex requirement set can be a real chore.

One way to eradicate requirement conflicts is to compare requirements that have similar wording. And while it's possible to do so with manual string searches, finding similarly worded requirements can be greatly simplified and accelerated using a requirements analysis tool.

For example, QVscribe's requirement similarity feature allows users to quickly find and compare similar requirements. By adjusting the degree of similarity to maximum, you can first find and eliminate duplicate requirements. Then, by reducing the degree of commonality, you can cast a wider net to identify and correct requirements that conflict with one another.



10. CYCLE DERIVED REQUIREMENTS THROUGH YOUR ANALYSIS PROCESS

DO-178C defines derived requirements as "Requirements produced by the software development process which (a) are not directly traceable to higher level requirements, and/or (b) specify behavior beyond that specified by the system requirements or the higher level software requirements."

Depending on your process, derived high-level software requirements may be defined by systems engineers, software developers, or both. In any case, DO-178C requires that these be "provided to the systems processes," including the system safety assessment process and the requirements analysis process.

In other words, you'll want to apply your documented requirements analysis process (and all our previous recommendations) to every high-level software requirement you define.

BONUS TIP: REFINE YOUR PROCESS WITH ADDITIONAL RE BEST PRACTICES

DO-178C is all about having a high-quality process for developing safe airborne software. In this article we've listed our top ten RE best practices for assuring compliance with DO-178C—those that apply directly to its explicit requirements. But we could list many more.

Continue to improve your software requirements definition process. Seek out and adopt RE best practices from other sources.

Sources we recommend include the INCOSE Guide for Writing Requirements mentioned earlier and our own 21 Top Engineering Tips for Writing an Exceptionally Clear Requirements Document, which you can download here.

QRA

Aerospace Requirements Checklist

- ☐ 1) Document your process for requirements analysis and review
- ☐ 2) Define your methods for assuring requirements traceability
- ☐ 3) Define your criteria for evaluating requirements
- ☐ 4) State requirements in verifiable, quantitative terms
- ☐ 5) Enforce consistent use of units and terms
- ☐ 6) Do not specify implementation details in functional requirements
- ☐ 7) Include rationale, but segregate it from the requirement statement
- ☐ 8) Automate your requirements analysis process
- ☐ 9) Eliminate conflicts between requirements
- ☐ 10) Cycle derived requirements through your analysis process
- ☐ 11) Refine your process with additional RE best practices

To learn more about QVscribe, visit qracorp.com/qvscribe

qracorp.com

version 1