

# DO-178C Best Practices

For Engineers & Managers

*By Vance Hilderman*



## DO-178C Best Practices: Introduction.

Practice: We've all engaged in it: piano, math, golf, flying... Usually practice involves a modicum of coaching, self-help and repetition. In avionics development, however, there is little time for practice; instead, everything counts. And the result has little margin for error: Schedules, budgets and particularly safety are all on the line. How then can "practice" be reconciled with "avionics development"? The best answer is to understand the breadth of worldwide development and glean the best knowledge and solutions from the aviation ecosystem. Welcome to DO-178C Best Practices.

In flying, there are tradeoffs between payload, range, speed and costs. The vast breadth of aircraft types for sale today belies the simple fact that many persons prioritize these tradeoffs differently. However, the variations in avionics software development practices are much more constrained: Everyone wants to minimize the following attributes:

- Cost
- Schedule
- Risk
- Defects
- Reuse Difficulty
- Certification Roadblocks

The following pages provide the DO-178C Best Practices which can minimize all six of these important attributes in your development.



## DO-178C Best Practices: Prelude

Certain good avionics software development practices are self-evident. Similar to improving human health, educated persons know that improved diet, exercise, sleep and stress relief are all "best practices". For software, the obvious good practices include utilizing defect prevention, experienced developers, automated testing and fewer changes. This paper isn't about the obvious, as it is assumed the reader is educated by virtue of making it to this page. Instead, the DO-178C Best Practices identified herein are subtler and considerably less practiced.

The following figure summarizes the Top 10 not-always-obvious DO-178C Best Practices:



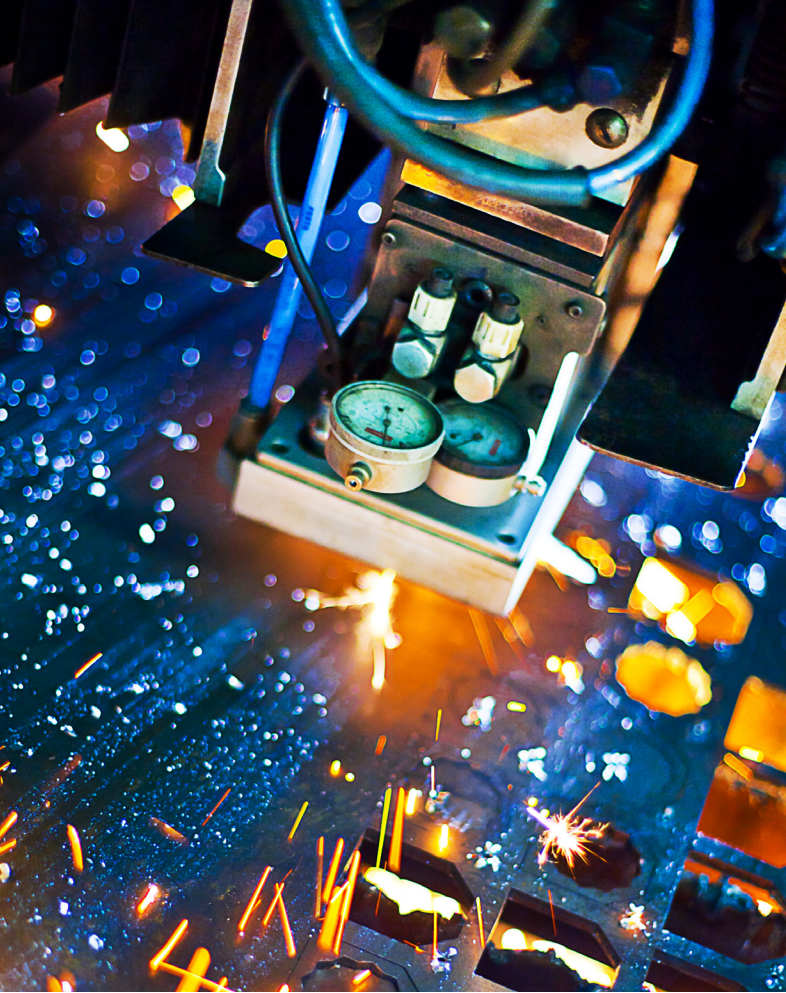
Top 10 Not-Always-Obvious DO-178C Best Practices

### 1. Improved LLR Detail

*Requirements are the foundation of good engineering. Detailed requirements are the foundation of great engineering.*

Smarter researchers than this author long ago proved that most software defects are due to weak requirements. In the book, *Mythical Man Month*, Brooks opined that assumptions were a leading cause of software defects. DO-178C was intentionally strengthened over its predecessor DO-178B to ensure acceptable requirements via 178C's mandate to trace structural coverage analysis to requirements-based tests (RBT). Remember: DO-178C doesn't provide strict requirements standards, but for DAL A, B, and C, the developer must. Those standards should define the scope and detail associated with High-Level Requirements (HLR's) and Low-Level Requirements (LLRs). Ideally, the Requirements Standard will include examples of HLR's versus LLR's. Requirements review checklists should likewise contain ample criteria for evaluating the level of detail within LLRs.





DO-178C explicitly requires standards for DAL A, B and C. Which standards? Requirements, Design and Code. Why doesn't DO-178C require a verification or testing standard? Supposedly, there should be less variation within testing, compared to the preceding lifecycle phases which are admittedly more variable between companies and projects. No one has ever accused DO-178C of requiring too few documents; given the traditional waterfall basis (inherited two decades prior from DO-178A), ample documents are required already. However, efficient companies recognize that verification is an expensive and somewhat subjective activity best managed via a Software Test Standard. Since it's not formally required, it would not have to be approved or even submitted. What would such a hypothetical Software Test Standard cover? At a minimum, the following:

- Description of RBT to obtain structural coverage
- Details regarding traceability granularity for test procedures and test cases
- Explanations of structural coverage assessment per applicable DAL(s)
- Definition of Robustness testing, as applied to requirements and code (per applicable DALs)
- If DAL A, explanations of applicable MCDC and source/binary correlation
- Coupling analysis practices including role of code and design reviews
- Performance-based testing criteria
- Examples of requirements and code, along with recommended associated test cases

## 4. Model Framework Templates

*Software modeling will eventually fade away ... when software functionality, complexity and size all decrease 90%.*

There are few safe bets in life, however this author claims continually increasing software functionality, complexity and size are all safe bets. As exercise will help manage a high-fat diet, software modeling better manages tomorrow's software. However, models and modeling techniques can vary greatly. Large variation within a project defeats much of the benefit of modeling, particularly within verification and reuse. Best practice? Use model frameworks and specify these in the project's Design Standard. Independently review these frameworks to explicit criteria (using a checklist), control them and require their usage.

## 2. Parallel Test Case Definition

*If a tester cannot unambiguously understand the meaning of a software requirement, how could the developer?*

DO-178C is agnostic regarding cost and schedule: The developer is freely allowed to be behind schedule and over budget. While transition criteria must be explicitly defined for all software engineering phases, it is normal for companies to define their test cases after the software is written. However, great companies define test cases before code is written. Why? Because it's better to prevent errors than detect them during testing. If a tester cannot unambiguously understand the meaning of a software requirement, how could the developer? Good companies verify requirements independently by having the software tester define test case as part of the requirements review, before any code is written. Requirements ambiguities or incompleteness are corrected earlier, yielding fewer software defects and expedited testing.

## 3. Implement Testing Standards

*Requirements Standard. Design Standard. Coding Standard. Testing Standard ...*

*Wait, there ISN'T a Testing Standard!?*



## 5. Fewer But Better Reviewers

*One great reviewer is better than many good reviewers.  
If more equaled better, airplanes would have 10 engines ...*

This author has never seen a ten-engine airplane, but he's seen many peer review teams with 10 engineers. Why so many reviewers? Was it optimism? Pragmatism? Perhaps simple naivety: Human nature is replete with more-is-better examples. However, in the case of software reviews, one great reviewer is the better choice. One great reviewer is more cost-effective, more productive, and with proper skill, better. When a rowboat is paddling away from an incoming torpedo, one oarsman will work harder than many. Human nature is human and reviewers perform better when they know they are the sole reviewer.

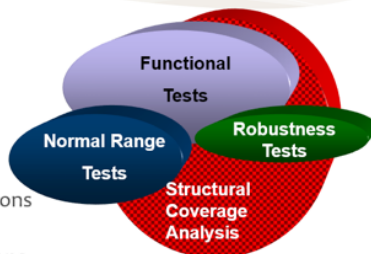
## 6. Automated Regression & CBT

In the non-critical software world, testing is a "great idea." Purchasing an exotic car or yacht can also seem like a great idea at acquisition time, as this author has personally experienced. However, unlike luxury cars and yachts, software testing should not be considered a luxury but rather a necessity. DO-178C requires a variety of necessary testing, with increased rigor per increased criticality. The basic types of testing are depicted below:



### Four Categories of Tests:

1. **Functional Tests**
  - All Requirements
2. **Normal Range Tests**
  - "Sunny Day" conditions
3. **Robustness Tests**
  - "Rainy Day" conditions
4. **Structural Coverage Analysis**
  - Cover all code



DO-178C requires regression analysis whereby software updates are assessed for potential impact to previously tested software with mandatory retests required where potential impact exists. Over the project life, and absolutely over the product life, more time will be spent on testing than on development. Many consider software testing to be the largest line-item expense in DO-178C. Devoting upfront time to develop a test automation framework can provide the single largest expense reduction ability. And continuous-based testing (CBT), which automatically retests changes continuously, is the best means to meet



regression objectives. Why? By continuously retesting all software the regression analysis is greatly simplified: Just repeat all the tests by pressing a button. Voilà.

## 7. Automated Design Rule Checker

*On their best days, humans perform satisfactorily when checking software design rules; in the safety-critical world, not all days are best days.*

The safety-critical software world is replete with tools and techniques for performing static code analysis, automating testing, structural coverage, and model-based design. Wonderfully helpful, even necessary. However, if an imperfection can be considered a defect, then the aforementioned activities miss an entire area of defective design. Consider: A leading cause of software defects is assumptions, hence the need for detailed requirements, traceability, coding standards, etc. However, different humans have different assumptions regarding software design and internal interfaces. These different assumptions yield imperfectly coordinated software components and these imperfections may mask defects. The remedy? Automated design rule checkers which help ensure consistent, deterministic interfaces and execution. Combined with the model framework templates of #4 above, a most powerful combination results.

## 8. Advanced Performance Testing

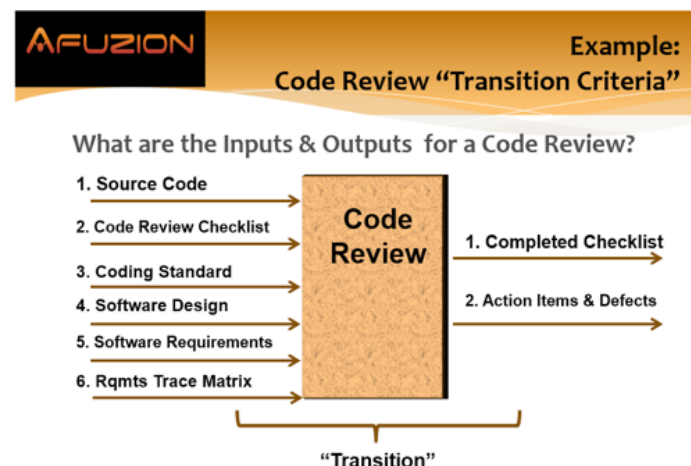
*Would you want to buy a new car model which has never been tested in aggressive driving conditions?  
I live in Los Angeles and Manhattan; me neither.*





Interestingly, DO-178C provides very little guidance on performance testing so a common pitfall ensues: minimal software performance testing. However, thorough performance testing is a hallmark of quality software and the only means to find certain defects, which otherwise are not detected until potentially catastrophic in-flight failures. Often, correcting performance related software defects entails major architectural changes, which are always time-consuming and expensive. A better way? Advanced performance testing which does the following:

- Defines worst-case loads, with worst-case execution times (WCET)
- Utilizes continuous maximal rate of change for inputs across all interfaces
- Verifies the separate DO-178C-mandated discrete performance requirements
- Consider degraded-mode operations where primary inputs are not available mandating use of more computationally intensive secondary inputs
- Utilizes worst-case Parameter Data Items (PDI's) where each PDI is deliberately chosen for WCET impact.



## 9. Parallel Traceability / Transition Audits

*"Why do it right the first time when it's fun to keeping doing it over and over..." – Anonymous*

Where the amateur athlete focuses on the end result, the professional instead focuses upon optimizing the technique since the end result depends upon that technique. Amateur and professional software engineers both know minimizing defects is a goal, but the professional knows that technique matters: In avionics that is best summarized via DO-178C's traceability and transition criteria. While the amateur

avionics team assesses traceability and transition criteria at the end, e.g. SOI-4, the experienced team instead deploys proactive SQA and tools to monitor bi-directional traceability continuously. Emphasize audits of transition criteria early, fix process shortfalls, and record the audit results. Remember, each type of artifact review constitutes a transition: Engineers must follow the defined transition criteria and QA must audit to assess process conformance. An example of a Software Code Review transition is depicted below. Ensure all the inputs and outputs are perfectly utilized, under CM, and referenced in the results:

## 10. Technical Training Workshops

Regardless of profession, the ingredients for becoming the "best" include a combination of 1) education, 2) coaching and 3) practice. Avionics groups employing Best Practices address the first two ingredients via training. Whether procured internally or externally, the odds of DO-178C success can be enhanced via technical training. Which training? Best to focus on areas of high return-on-investment including improved productivity and consistency in the following critical processes:

- Requirements writing (emphasize consistent medium granularity for consistency)
- Software Testing (emphasize thoroughness and full, real-world scenarios which exercise cross-domain interfaces)
- Reviews (emphasize standard, detail, robustness, changes and identify meaningful defects)
- Auditing (emphasize transition criteria process) and finding/fixing actual defects

**For Advanced DO-178C Training information, see:**

<http://afuzion.com/avionics-training/workshops/avionics-software-advanced-do-178c-training-class/>

**For DO-178C Gap Analysis information, see:**

<http://afuzion.com/gap-analysis/>

**What is AFuzion? Fun One-Minute Video:**

<https://www.youtube.com/watch?v=RMzLRzcahJE>

For additional details on this white paper topic, or formal training, avionics software design, development, verification and certification, contact Vance Hilderman.

For DO-178 & DO-254 specific details, procure the book "Avionics Certification: A Complete Guide To DO-178 & DO-254", from major bookstores such as Amazon.com. (The author of this white paper is the primary author of that book.) Also, the new book "Avionics Development Ecosystem" by Vance Hilderman covers the big-picture view of avionics development from safety, to systems, and through all key regulatory and design aspects for modern avionics development. See the Afuzion website, [www.afuzion.com](http://www.afuzion.com), for advanced training modules relevant to DO-178C beginners and experts alike.



## About Vance Hilderman

Vance Hilderman is the founder of two of the world's largest avionics development services companies. He is also the developer of the world's first training in DO-178 and trainer of over 8,000 engineers in 45 countries in DO-178, DO-254, DO-278, and DO-200A. He is also the primary author of the world's first and bestselling book on DO-178 and DO-254.

## About Jama Software

Jama Software is the definitive system of record and action for product development. The company's modern requirements and test management solution helps enterprises accelerate development time, mitigate risk, slash complexity and verify regulatory compliance. More than 600 product-centric organizations, including NASA, Thales, and Caterpillar have used Jama to modernize their process for bringing complex products to market. The company is headquartered in Portland, Oregon. For more information, visit [www.jamasoftware.com](http://www.jamasoftware.com).

