Certification of Safety-Critical Software Under DO-178C and DO-278A

Stephen A. Jacklin[1]
NASA Ames Research Center, Moffett Field, CA, 94035

The RTCA has recently released DO-178C and DO-278A as new certification guidance for the production of airborne and ground-based air traffic management software, respectively. Additionally, RTCA special committee SC-205 has also produced, at the same time, five other companion documents. These documents are RTCA DO-248C, DO-330, DO-331, DO-332, and DO-333. These supplements address frequently asked questions about software certification, provide guidance on tool qualification requirements, and illustrate the modifications recommended to DO-178C when using model-based software design, object oriented programming, and formal methods. The objective of this paper is to first explain the relationship of DO-178C to the former DO-178B in order to give those familiar with DO-178B an indication of what has been changed and what has not been changed. With this background, the relationship of DO-178C and DO-278 to the new DO-278A document for ground-based software development is shown. Last, an overview of the new guidance contained in the tool qualification document and the three new supplements to DO-178C and DO-278A is presented. For those unfamiliar with DO-178B, this paper serves to provide an entry point to this new certification guidance for airborne and ground-based CNS/ATM software certification.

I. Introduction

RTCA DO-178B[1] has long been regarded as a document providing the premier means or path to obtain FAA certification of software to be used in airborne systems. DO-178B was not intended to be a process guide for software certification, but rather a description of what high-quality software development processes should be put in-place to create airborne software that performs its desired function. In principal, if life cycle evidence can be produced to demonstrate that these processes have been correctly and appropriately implemented, then such software should be certifiable. The document is maintained by the RTCA (Radio Technical Commission for Aeronautics, established 1935), which is a private association of over 250 aeronautical organizations, including the FAA, NASA, DOD, other government agencies, airline manufacturer, airline operators, aircraft equipment suppliers, and various pilot associations.

Seven years ago, the RTCA created special committee 205 (SC-205) to produce a revision of DO-178B to account for new software development and verification technologies that were deemed immature at the time DO-178B was written. The new version, DO-178C "Software Considerations in Airborne Systems and Equipment Certification"[2,] was released in December 2011. Rather than placing all of the new guidance in DO-178C, the special committee decided to place the vast majority of the new guidance in six other documents. These documents were released together with DO-178C. They are:

- RTCA DO-278A[3]: Software Integrity Assurance Considerations for Communication, Navigation, Surveillance and Air Traffic Management (CNS/ATM) Systems
- RTCA DO-248C[4]: Supporting Information for DO-178C and DO-278A
- RTCA DO-330[5]: Software Tool Qualification Considerations
- RTCA DO-331[6]: Model-Based Development and Verification Supplement to DO-178C and DO-278A
- RTCA DO-332[7]: Object-Oriented Technology and Related Techniques Supplement to DO-178C and DO-278A
- RTCA DO-333[8]: Formal Methods Supplement to DO-178C and DO-278A

[1] Aerospace Engineer, Intelligent Systems Division, Mail Stop 269-2, Senior Member AIAA

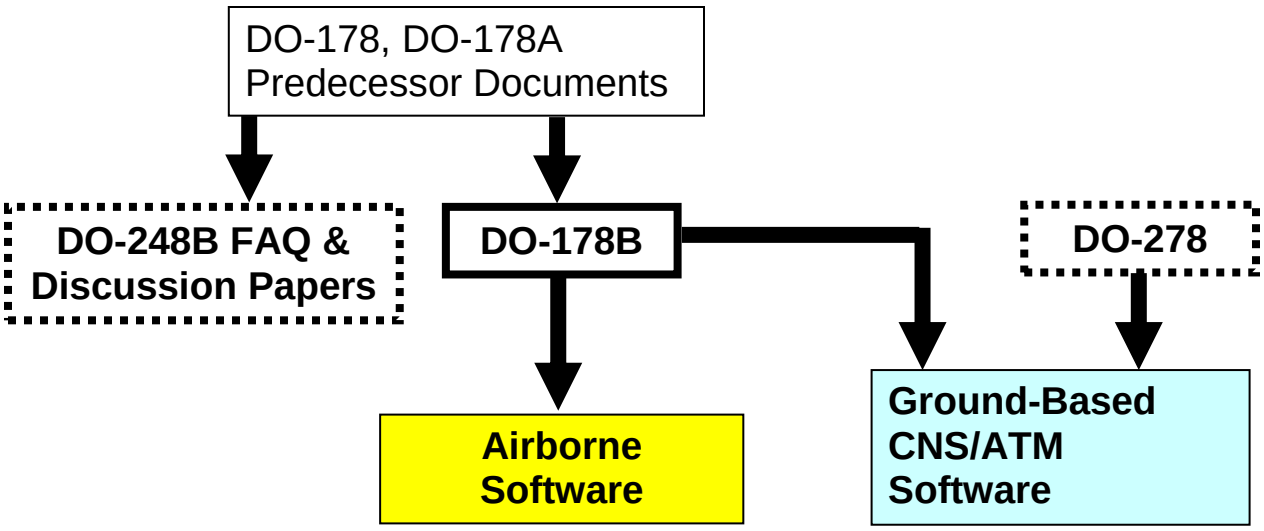

Figure 1. RTCA airborne and CNS/ATM software certification-related documents introduced in December of 1992. Dashed lines indicate supplements.

Figure 1 illustrates the functional relationship of airborne and CNS/ATM software certification-related documents published by RTCA prior to December 2011. DO-178B was a derivative product of DO-178A, DO-178, and other documents and was released in December 1992. The guidance contained in DO-178B was intended to be applicable to both airborne and ground-based software development. DO-278 was intended to be a supplemental document to modify the guidance of DO-178B for CNS/ATM software. Hence, both DO-178B and DO-278 together were to be referenced for the ground side. DO-248B was an additional supplement that provided no additional certification guidance, but contained an appendix of frequently asked software certification questions, several discussion papers of key DO-178B concepts, and the rationale used to create the DO-178B and DO-278 documents. The boxes in dashed lines indicate supplemental documents that were not intended to be complete in themselves.

Figure 2 illustrates how the new documents introduced by RTCA in December 2011 for DO-178C and DO-278A relate to each other. In this diagram, the dashed boxes indicate supplemental documents that are not intended to be used on their own. The supplemental documents modify the guidance contained in DO-178C and DO-278A. On the airborne side, DO-178C is the key document and it is a direct derivative of DO-178B. On the ground side, DO-278A is the key document, but it is not a direct derivative of DO-278. Rather, DO-278A combines the guidance of DO-178C and DO-278 to produce a stand-alone reference for ground-based software verification. For both airborne and ground-based software, DO-331, DO-332, and DO-333 provide additional guidance for software using model-based development, object-oriented programming, and formal methods, respectively. These supplements provide additional guidance for both DO-178C and DO-278A, but need not be used if not applicable. DO-330 is a stand-alone document containing guidance for tool qualification and is intended to be used not only by persons using tools to verify software or auto-generate code, but also by persons developing the software tools. The tool developers need not read DO-178C or DO-278A because the guidance contained in those documents that is relevant for tool software development is repeated in DO-330.

The purpose of this paper is to provide an overview of the new guidance for safety-critical airborne and ground-based CNS/ATM software contained in DO-178C, DO-278A, and the other documents. In section II, the similarities of DO-178C to DO-178B will be presented by reviewing the basics of the DO-178B verification philosophy. In section III, an overview of the major new guidance contained in DO-178C is presented to highlight what has been changed. Section IV discusses the relationship of DO-278A developed for ground-based CNS/ATM software to the guidance presented in DO-178C for airborne software. The remaining sections of the paper provide a discussion of the new guidance contained in the other documents; section V for DO-248C, section VI for DO-330, section VII for DO-33, section VIII for DO-332, and section IX for DO-333.

Within the scope of this paper it is not possible to cite all or even most of the guidance contained in the new DO-178C document set from RTCA. Taken as a whole, the new documents comprise over 1000 pages of new documentation. The interested reader must download these documents from RTCA[9] in order to fully appreciate and apply the new guidance. This paper provides an entry point for those interested in understanding the scope of these publications.

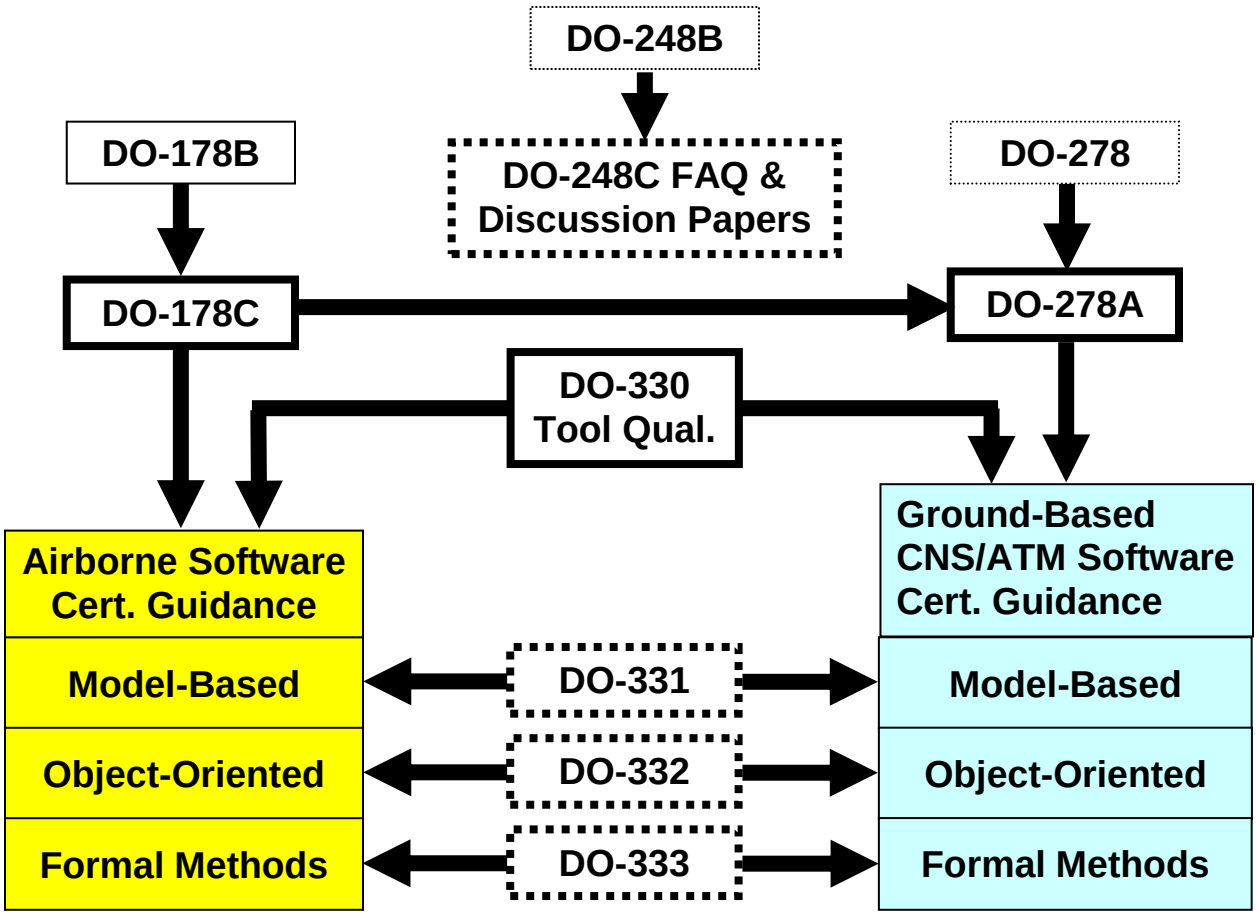

Figure 2. Relationship of the new RTCA document set for airborne and CNS/ATM software certification introduced in December 2011.

II. The A-B-Cs of the DO-178C Software Verification Philosophy

The purpose of this section is to identify the similarities of the guidance contained in DO-178C to past versions of the document. DO-178C is built on the principles established by its predecessor documents, DO-178, DO-178A, and DO-178B. Since testing can never prove the absence of software errors, the primary DO-178C philosophy is to demonstrate the quality of the software development process from beginning to end in an effort to minimize the creation of error. DO-178C, like DO-178B, calls for an extensive amount of requirements-based software testing to be performed, but equally important is the emphasis placed on system safety analyses, software analyses, software reviews, and formal proofs used to augment and support the development process. The subsections below identify the guidance presented in DO-178B that is retained in DO-178C.

A. Software Levels and Coverage.

DO-178C (section 2) uses the same software levels categories (SL-A to SL-E) as are used in DO-178B. The meaning of these categories is unchanged from their meaning in DO-178B. Level A is the highest level of software criticality. Like DO-178B, DO-178C (section 6) requires extensive verification coverage testing for level A and B software. Coverage refers to the degree to which it can be proved that the verification activities cover all requirements, code structure, and object code. DO-178C divides coverage into two types, requirements-based coverage and structural coverage. Requirements-based coverage analyzes the software test cases to confirm that they provide proof that all requirements have been satisfied. Structural coverage is met by proving that the test cases execute all code statements and that all data coupling and control paths have been tested.

B. Software Development Plan.

The Software Development Plan identifies the method of software development. It specifies the coding standards, programming languages, software testing, debugging tools, software development procedures, and the hardware used to develop and execute the software. As stated by DO-178C (section 4), the purpose is "to choose requirements development and design methods, tools, and programming languages that limit the opportunity for introducing errors, and verification methods that ensure that errors introduced are detected". Tools include things like compilers, linkers, and V&V tools. Reviews are to be regularly conducted throughout the software development process to ensure that the Software Development Plan is being followed.

C. Software Development Process.

As was true in past versions of DO-178, DO-178C (section 5) views the software development process as a life cycle that starts with the planning and development of software requirements, continues through the software development and testing, and ends with the deployment and maintenance of the software. The software development process begins with recognition that the software is part of a larger system (e.g., an aircraft). The system defines the requirements of the software. In DO-178C, these are referred to as the system requirements allocated to software. The list of requirements includes not only the performance specifications of what the software is supposed to do, but also includes requirements derived from the System Safety Assessment and other documents. The decomposition of system requirements into software requirements remains a key step in the software development process. The incomplete or incorrect formulation of the software requirements will produce validation failures during software integration testing. Validation is the process of determining that the software requirements are correct and complete. DO-178C does not provide guidance for software validation testing because it is reasoned that software that is verified to be correct should, in theory, have no validation problems during software integration testing, unless the software requirements are incomplete or incorrect.

The software development process transforms high-level and derived high-level software requirements into code via a sequential process. In the first step, the high-level requirements are used to develop the software architecture. From the software architecture, low-level and derived-low-level requirements are developed. A derived requirement is anything that the software must do to function properly, yet is not stated as part of the software performance or safety requirements. Low-level requirements are used to produce the source code, and the source code is used to generate the object code for the target computer. It sometimes happens that high-level requirements are used to generate source code directly, in which cases those high-level requirements are also considered to be low-level requirements. Like DO-178B, DO-178C requires traceability from requirements to code.

D. Software Verification Process.

The software verification process is aimed at showing the correctness of the software. It consists of requirement reviews, code reviews, analyses, and testing. All steps in the decomposition of high-level system requirements to object code are considered in this process. DO-178B and DO-178C require examination of the output of all processes to check for software correctness and to find errors. DO-178C (section 6) requires that: 1) the high –level software requirements are correctly and completely formed from the system requirements, 2) the high-level requirements are complete and consistent, 3) the software architecture correctly and completely meets all high-level requirements, 4) the low-level requirements correctly and completely fulfill the software architecture, 5) the low-level requirements are consistent and correct, 6) the software code correctly satisfies all low-level requirements, 7) all code is traceable to one or more low-level requirements, and finally, 8) the object code correctly implements the software on the target computer, and it is traceable and complies with all low-level and high-level requirements.

E. Certification.

The certification process described in DO-178C (section 10) is the same as that presented in DO-178B. Software certification (technically, "approval") is obtained as the result of the certification authority agreeing that the Plan for Software Aspect of Certification (PSAC) has been fulfilled. In the United States, the authority responsible for certifying aircraft flight software is the Federal Aviation Administration (FAA). The PSAC is developed in collaboration between the software developer's Designated Engineering Representative (DER) and the FAA. The same certification liaison process presented in DO-178B is also contained in DO-178C (section 9).

F. Other Similarities.

The sections contained in DO-178C describing the software life cycle (section 3), the software configuration management plan (section 7), the software quality assurance plan (section 8), the software development standards

(section 4.5), the software design standards (section 11), and the overall verification activities to be performed are generally the same as those presented in DO-178B.

III. What's New in DO-178C ?

Since DO-178C was seven years in the making, one might assume that the document has been substantially changed from DO-178B, but this is not true. Although DO-178C has many minor changes, these are mostly either editorial in nature or are clarifications made to help readers better understand key DO-178B concepts. These changes are frequent and helpful. However, the core structure and content of DO-178C is essentially the same as that seen in DO-178B. The utility of this similarity is to make DO-178C backward compatible with DO-178B by design. This means that existing software that has been previously approved under DO-178B is also approvable under DO-178C.

The major new guidance, additions, and modifications introduced in the DO-178C documentation set are contained in the supplemental documents and will be discussed in later sections of this paper. The subsections below highlight some of the new guidance contained in the DO-178C core document.

A. Activities, Guidance, and Guidelines.

One of the most frequent changes seen in DO-178C is that the word "activities" is used to replace the word "guidance" found in DO-178B, and wherever "guidance" appeared in DO-178B, DO-178C uses the word "guidelines". The reason for these changes is that because DO-178B described processes (e.g., software development process) in terms of a list of activities the software developer was to perform, it made sense to call them activities, not guidance. Commensurate with this effort, the tables in Annex A have also been modified to include a new "activity" column that lists the relevant activities associated with all verification activities supporting the objective. DO-178C reserves use of the word guidance to indicate the most important steps to certification authorities. The word guideline is still used in some places, but its meaning is intended to indicate a list of supporting information.

B. Parameter Data Item Files.

DO-178C treats parameter data items in the same manner as executable object code. DO-178C defines parameter data items as data that influences the behavior of the software without modifying the executable object code. The parameter data file is directly usable by the processing unit of the target computer. Parameter data file items can be used to configure the software or provide database information that the executable code can use in execution. In nearly all instances, DO-178C replaces the phrase "executable object code" used in DO-178B with "executable object code and parameter data items". In making this change, DO-178C calls for the same verification process to be followed for parameter data file items as that done for executable object code.

C. Bi-directional Software Traceability.

DO-178C emphasizes that two-way or bi-directional traceability is required between 1) system requirements (allocated to software) and high-level requirements, 2) high-level requirements and low-level requirements, 3) low-level requirements and source code, 4) software requirements and test cases, 5) test cases and procedures, and 6) test procedures and test results. Although DO-178B probably had this intent, the actual wording implied traceability only in the decomposition direction from high-level requirements to source code. DO-178C makes it clear that traceability needs to be verifiable in both directions and that verifiable trace data between these entities must exist. This assures that orphan source code and dead source code are not inadvertently produced.

DO-178C allows an exception to requirements traceability for compilers that produce object code that is not directly traceable to source code. However, the software planning process must provide a means to detect this code and ensure its verification.

D. Product Service History.

Under alternative methods (section 12.3), DO-178C provides expanded guidance for using product service history as a means to gain certification credit. Software that has been in service a length of time and whose executable object code has not been modified in an uncontrolled manner may be given certification credit. DO-178C states that the software developer should propose the amount of certification credit sought in the Plan for Software Aspects of Certification (PSAC). Product service history is identified as part of the PSAC in DO-178C.

E. Tool Qualification Levels.

DO-178C states that the tools used to generate software or to verify software must themselves be verified to be correct. This tool verification process is called qualification. Moreover, a tool such as a compiler qualified for one project is not necessarily qualified for a different project. DO-178C distinguishes between tools used to automatically generate software from tools used to automate some portion of the verification process. In general, DO-178C requires greater scrutiny of software generation tools. DO-178C sets forth five tool qualification levels (TQL-1 – TQL-5) based on the software level, and whether the tool is used to generate software, to verify software, or used to detect software errors. DO-178C refers the reader to the tool qualification supplement (DO-330) for specific guidance.

F. Formal Methods and Assurance Cases.

DO-178C removes explicit reference to the use of formal methods as an alternative method to satisfy DO-178C objectives, but instead cites the use of assurance cases to provide adequate confidence and evidence that a product or process satisfies its requirements. DO-178C defines an assurance case as a technique in which arguments are explicitly given to link the evidence to the claims of compliance with the system safety objectives. A rationale for an assurance case may be included in the software plans if that means of verification is planned. Guidance on the use of formal methods is presented in the DO-333 supplement on Formal Methods.

IV. RTCA DO-278A: Software Integrity Assurance Considerations for CNS/ATM Systems

DO-278A provides guidance for the production of ground-based, Communication, Navigation, Surveillance, and Air Traffic Management (CNS/ATM) software, just as DO-178C provides guidance for the production of airborne software. Because the former DO-278 was intended to be used as a supplement to DO-178B (see Fig. 1), CNS/ATM software developers were required to be familiar with DO-178B. DO-278 described the additions, deletions, and modifications to DO-178B that applied to the verification of ground-side software.

In contrast, DO-278A was created by combining DO-178C and DO-278 to make a single, stand-alone document. As a result, DO-278A may be used without reference to DO-178C. Both DO-178C and DO-278A have the same section names and use the same section numbers. The differences are that some subsections have been added to DO-278A. A good many of the differences between DO-178C and DO-278A are produced by changes in terminology, for example:

- "Software level" in DO-178C was replaced with "assurance level" in DO-278A
- "Certification" authority in DO-178C was replaced with "approval" authority in DO-278A
- "Aircraft" or "airborne system" in DO-178C was replaced with "CNS/ATM system" in DO-278A
- "Adaptation" data in DO-178C was replaced with "parameter" data in DO-278A
- The "Plan for Software Aspects of Certification (PSAC)" in DO-178C is referred to as "Plan for Software Aspects of Approval (PSAA)" in DO-278A

A. Assurance Level Definitions.

Whereas DO-178C defines five software levels (A-E) to categorize the effect of software failure in airborne systems, DO-278A defines six assurance levels (A-F). DO-178C abbreviates software level with SL, whereas DO-278A uses AL for assurance level. Table 1 compares the DO-178C and DO-278A software level classification schemes. As cited in DO-278 (but removed from DO-278A), assurance level 4 (or AL-4) was developed to account for "certain CNS/ATM systems where AL-3 was too stringent and AL-5 was too lenient". DO-278A includes a column in the Process Objectives and Output Tables of Annex A to indicate the objectives that specifically apply to AL-4.

B. Tool Qualification.

DO-278A contains essentially the same tool qualification guidance contained in DO-178C. Like DO-178C, DO-278A requires software development and verification tools are qualified when the processes used in DO-278A are eliminated, reduced, or automated by the use of software tools. The main difference is that DO-278A takes into account the additional assurance level used for CNS/ATM systems. DO-278A refers the reader to DO-330 for an in depth discussion of the activities and guidance for tool qualification.

C. Service Experience.

The Product Service History section from DO-178C was expanded and added to DO-278A as the Service Experience section. This section describes what previous usage of a software product can be counted toward

Table 1: Comparison of DO-178C software levels and DO-278A assurance levels.

Software Failure Effect Category	DO-178C Software Level (Airborne Software)	DO-278A Assurance Level (CNS/ATM Software)
Catastrophic	SL-A	AL-1
Hazardous	SL-B	AL-2
Major	SL-C	AL-3
Less than major, more than minor	Not used	AL-4
Minor	SL-D	AL-5
No Effect	SL-E	AL-6

assurance credit. DO-278A specifies the requirements for receiving credit for product service history. The main objective is to verify that an equivalent level of safety is provided by the service experience history as would be otherwise obtained by following the standard DO-278A guidance. Whereas DO-178C identifies flight-hours as a useful metric for airborne software service experience, DO-278A cites in-service hours as an appropriate metric for CNS/ATM systems. DO-278A also provides guidance for systems having deactivated code and for those systems using recovery methods to recover from software or system errors.

D. COTS Software.

DO-278A includes an extensive section on the use of Commercial Off-The-Shelf (COTS) software in CNS/ATM systems. This section expands the COTS material presented in DO-278. In DO-278A, COTS software is software that is sold by commercial vendors without modification or development of the software required. Any software needed to integrate COTS software into a CNS/ATM system (e.g., wrapper code) is approvable only if it is developed in a manner that fulfills all the objectives of DO-278A.

The guidance provided by DO-278A for COTS software aims to ensure that the level of confidence in COTS software is the same as that for software developed according to the standard guidance provided in DO-278A. In order to identify any software development weaknesses of COTS software, DO-278A recommends that a gap analysis be performed to identify the extent to which the objectives of DO-278A can be demonstrated to have been achieved by the COTS software. An assurance plan should be developed to specify how the gaps will be satisfied for the assurance level sought. DO-178A recommends that a COTS software integrity assurance case be developed that provides a rationale for demonstrating that the software meets its requirements through a rigorous presentation of claims, arguments, evidence, assumptions, justifications, and strategies. As such, COTS software must essentially be shown to meet all the objectives of DO-278A. DO-278A presents an extensive explanation of the software planning, objectives, activities, acquisition, verification, configuration management and quality assurance processes and objectives in Section 12 and in the tables of Annex A.

E. Additional System Considerations.

DO-278A addresses additional topics for ground software verification not considered in DO-178C. These are software communication, security, adaptability, and cutover (or hot-swapping). More than airborne software, ground-software is comprised of many distributed system components. System communication guidance is provided for ground software systems that are coupled together. The main concern when coupling systems is that software approved to a lower assurance level might potentially corrupt software approved to a higher level. The general fix for this situation is to specify further verification activities to increase the assurance level of the lower-level software. A section on hot-swapping specifies additional considerations to ensure software integrity for systems that are in use 24-hours a day and require real-time software and hardware updates.

V. DO-248C: Supporting Information for DO-178C and DO278A

DO-248C provides a list of frequently asked questions, discussion papers, and rationale for DO-178C and DO-278A. It is not intended that this supplement be read from front to back, but rather topically. A list of searchable key words is provided to help the reader find the most pertinent material for a topic of interest.

DO-248C provides a wealth of discussion papers that contain explanatory information supporting the guidance found in DO-178C and DO-278A. Those who worked on SC-205 will recognize that these discussion papers encapsulate the great debates held during the formulation of DO-178B and DO-178C. Discussion papers were the primary means SC-205 members used to facilitate discussion of proposed changes to DO-178B. Most are short (1-2 page) documents that describe the supporting rationale for a proposed change. Literally hundreds of discussion papers were written over the course of the project.

DO-248C also presents an appendix of 84 frequently asked questions and answers. Examples are:

- Does the DO-178C/DO-278A definition of COTS software include software option-selectable software?
- What needs to be considered when performing structural coverage at the object code level?
- How can all Level D (AL-5) objectives be met if low-level requirements and source code are not required?

The last section of DO-248C presents 11 rationale arguments, one to discuss the intended use of each section in DO-178C (2 thru 12) plus a rationale for the creation of the supplements to DO-178C and DO-278A.

It is important to note that while DO-248C is interesting and useful, it does not provide any additional certification or approval guidance for airborne or ground-based CNS/ATM software. It provides a large quantity of explanatory material and a record of the great arguments and rationale developed while writing the new guidance.

VI. DO-330: Software Tool Qualification Considerations

DO-330 is a stand alone document that provides guidelines to judge when tool qualification is necessary, and if so, what verification activities are recommended. DO-330 presents guidance for both tools used to create software and tools used to verify software. The goal of the guidance is to ensure that these tools are developed to the same software assurance level as the software they produce or verify. DO-330 repeats much of the same guidance contained in DO-178C because it is intended to be used by audiences that are not familiar with DO-178C or DO-278A. Hence, DO-330 is very similar in appearance and content to DO-178C, and has the same document organization. Software developers and automated verification tool developers, therefore, need not look at DO-178C. The subsections below highlight some of the tool qualification guidance provided by DO-330.

A. Tool Qualification Levels.

DO-330 defines five tool qualification levels (TQLs). Tool qualification level 1 (or TQL-1) is the highest qualification level and has the most objectives and verification activities. TQL-1 is required for software tools that are used to generate either DO-178C software level A (SL-A) software or DO-278A assurance level 1 (AL-1) software. TQL-2 is required for software tools that are used to generate either SL-B or AL-2 software. TQL-3 is required for software tools that are used to generate either SL-C or AL-3 software. TQL-4 and TQL-5 are required for software tools that are used to verify software (not generate it). DO-330 places more stringent verification requirements on tools used to generate code than tools used to verify code. This distinction is the same as that defined in section 12.2 of DO-178C and DO-278A.

B. Tool Verification Activities.

Annex A of DO-330 presents the Tool Qualification Objective Tables to indicate which objectives, processes, and activities need to be satisfied as a function of the TQL. TQL-1, TQL-2, and TQL-3 (the higher levels) generally require the satisfaction of every process, every activity, and every output verification step listed in the tables. TQL-1 also has the most objectives requiring satisfaction by independence. TQL-2 has less objectives requiring independent satisfaction than TQL-1, but requires almost the same activities. TQL-3 has fewer activities than TQL-2 and does not require independence of verification except for the quality assurance activities. TQL-4 (the special category that has no equivalence to a DO-178C software level) provides less stringent satisfaction of the tool planning process, the tool development process, and the verification of outputs. TQL-5 (equivalent to SL-D) further relaxes the requirements by requiring no tool development process or verification tool requirements process, integration process, or testing of tool outputs. TQL-6 is not addressed by the Annex of DO-330.

C. Tool Development Life Cycle and Processes.

DO-330 provides essentially the same software development guidance for tool qualification as that presented in DO-178C for software verification. An important distinction, however, is that DO-330 recognizes that there are two life cycles to consider. There is the life cycle associated with the development of the software tool, and there is the life cycle of the software on which the tool will find application. DO-330 provides guidance for both.

DO-330 requires that a tool operational requirements (TOR) document be written to specify the requirements of how the tool will be used within the software life cycle. The requirements set forth in the TOR are required to be

verified and validated. DO-330 recommends the same software development life cycle processes as those specified in DO-178C or DO-278A, but references the TOR rather than system requirements. The tool requirements are, in turn, developed from the TOR and are then used to develop the tool architecture and low-level requirements. The verification of the tool design process, the tool coding process, and the tool integration process are similar to those presented in DO-178C and DO-278A. Like DO-178C, DO-330 requires bi-directional traceability between requirements and code.

D. Tool Verification Process.

The tool verification process recommended by DO-330 consists of two parts. The first part is comprised of a combination of reviews, analyses, and test cases to verify that the requirements are correct, that the tool architecture meets the requirements, that the low-level requirements satisfy the software architecture requirements, and that the source code fulfills the high and low-level requirements.

The second part of tool verification ensures that the software tool meets its intended requirements as a software development or verification tool. The tool operational verification process is performed to provide confidence that the outputs and functionality of the tool comply with the tool operational requirements in the intended operational environment. The operational verification process consists of a combination of reviews, analyses, and tests to demonstrate full coverage of the software life cycle intended to be eliminated, reduced, or automated by use of the tool. The test cases and procedures used to show TOR satisfaction should be requirements-based tests. The test procedures should include tests of the object code in its operational environment. The set of inputs used in the test cases should represent those found in actual tool use.

VII. RTCA DO-331: Model-Based Development and Verification Supplement

The DO-331 supplement is intended to augment DO-178C and DO-278A when model-based development and verification are used as part of the software life cycle. The supplement does not provide guidance on the use of models for verification purposes. It does not discuss how model-based development may be used to support automated code generation, automated test generation, or the automated verification of software requirements and architectures. Rather, the aim of the supplement is to identify how the guidance in DO-178C and DO-278A may be modified when software is developed using model-based methods.

In DO-331, a model can be an abstraction of actual software code or a portion of the verification process. The model may also contain requirements and/or definition of the software architecture so that it may be used for direct analysis of the software behavior.

DO-331 repeats some of the guidance contained in DO-178C to show precisely where modifications and additions for model-based design apply. The following subsections provide a summary of the major new guidance contained in DO-331.

A. Model Requirements.

DO-331 makes a distinction between requirements for specification models and requirements for design models. Specification models use high-level software requirements to state model functional, performance, interface, and safety characteristics. Design models use primarily low-level requirements and software architecture specifications to represent internal data structures, data flow, and control flow. In either case, DO-331 requires that the models specify the configuration items, modeling techniques, model element libraries, interface descriptions, configuration items, and model development environment. Traceability between the design model, low-level requirements and the model code is required. There should be no model code that cannot be traced back to low-level requirements.

B. Model Coverage Analysis.

The objective of model coverage analysis is to discover requirements contained in the model that were not exercised by verification test cases. DO-331 recommends model coverage be shown for all state machine transitions, logic equation decisions, numeric data equivalence classes (and boundary data), and all derived requirements. Model coverage analysis should be performed using requirements-based verification test cases.

C. Model Simulation.

In order to obtain certification credit for simulation, DO-331 requires that the applicant clearly show what reviews and analyses are needed to satisfy the model verification objectives. The analyses must show that simulation cases exist for each model requirement and that these simulations address both normal range and robustness test inputs. Verification of the executable object code is encouraged to be primarily performed by testing in the target

computer environment. The objective of model simulation is to verify that the model satisfies the requirements used to create it and to gather evidence that the model is accurate, consistent, and compatible with all system-level, high-level, and low-level requirements.

D. Software Model Standards.

DO-331 recommends that software models be developed to standards that define the modeling techniques used to build the models. These software model standards specify the methods and tools used to develop the models, including the modeling language. The software model standards should include guidance on programming styles such as naming conventions, diagram layouts, allowable elements, and the number of nesting levels or architectural layers. The software model standards should state the level of traceability between requirements and other life cycle data. The software model standards should also specify any constraints on the use of modeling tools and model element libraries.

VIII. RTCA DO-332: Object Oriented Technology and Related Techniques Supplement

RTCA DO-332 was written to provide guidance on the use of object oriented programming languages that use concepts such as inheritance, polymorphism, overloading, type conversions, exception management, dynamic memory management, virtualization, and other concepts not universally in common usage at the time DO-178B was written. The DO-332 supplement is very well written and includes much explanatory text concerning the basic features of object-oriented programming.

DO-332 identifies the additions, modifications, and deletions to be made to the DO-178C (or DO-278A) objectives and activities when object-orientated techniques are used in airborne or ground-based software. Annex A of DO-332 contains modifications to the verification activities specified in DO-178C, while Annex C presents modifications to the verification activities specified in DO-278A. Annex D provides a discussion of vulnerabilities associated with the use of objected-oriented technologies. The highlights of the new guidance are presented below.

A. Software Development Process.

DO-332 recommends that the class hierarchy used should be based on high-level requirements and verified for consistency. A locally type-consistent class hierarchy should be developed with associated low-level requirements wherever substitution of types happens. Local type consistency is required to be verified. DO-332 also calls for the software development process to include a plan for dynamic memory management. A strategy for exception management should also be included. DO-332 states that bi-directional trace data should exist to show traceability between requirements and methods, since methods are used to implement all functionality in object-oriented programs. In addition, a requirement which traces to a method in a class should also trace to the method in its subclasses.

B. Software Verification Process.

DO-332 states that test cases should ensure that class constructors properly initialize the state of their objects. In cases where inheritance with method overriding and dynamic dispatch are used, verification activities should be done to ensure that all type substitutions are safe and that each class pass the same verification testing as their parent types.

DO-332 stresses the importance of having verification activities to verify that dynamic memory allocation is done correctly. Verification activities must show that the dynamic memory management is robust to reference ambiguity, fragmentation starvation, de-allocation starvation, memory exhaustion, premature de-allocation, lost updates, stale references, and unbounded allocation or de-allocation times. It should be verified that there is sufficient memory to accommodate the maximum storage required. The memory must be verified to successfully allocate memory for every request as long as there is sufficient free memory. The means of calculating the amount of free memory remaining should also be verified to be accurate and free from leakage problems whereby memory which is no longer needed fails to be de-allocated.

C. Vulnerability Analysis.

DO-332 presents a vulnerability analysis discussion in Annex D. The purpose is to describe the complications that may arise with the use of object-oriented technologies. These special problems are associated with inheritance, parametric polymorphism, overloading, type conversion, exception management, dynamic memory management, and virtualization. Examples of the extensive verification guidance provided by the annex include:

- With regard to inheritance, a demonstration of type consistency by verifying that each subclass is substitutable for its superclass is recommended. Every verification activity performed on the superclass should also be performed on the subclass.
- For software having parametric polymorphism, verification activities should show that operations acting on substituted parameters implement the intended semantic behavior. Each unique instantiation of a parameterized type or combination of types should be verified.
- To minimize the problems associated with overloading, the use of explicit type conversion should be used to reduce overloading ambiguity. Verification activities should ensure that type conversions (implicit and explicit) are safe and that all implications are understood.
- It is recommended that a strategy be developed to handle all exceptions such as range checks, bounds checks, divide-by-zero checks, or checks on post-conditions. It is desired that all code modules handle exceptions in the same way.
- It recommended that an automatic method of memory reclamation be provided instead of relying on the correct use of malloc() and free() for dynamic memory management.
- It is advised that worst-case execution timing be performed considering all in-code dynamic memory operations. Separate threads used for memory management (e.g., garbage collection) and should be considered as part of the task load.

Annex D also discusses activities for verification of traceability, structural coverage, component-based development, memory management, and timing for object-oriented programs. Though procedural programming techniques require verification of these as well, object-oriented programming requires additional analyses. DO-332 recommends verifying traceability between the requirements of a subclass and the requirements of all of its superclasses to ensure type substitutability. Traceability should be shown between object code and the source code if multiple dynamic dispatches are possible through a call point. Detailed discussion of these points and many others are presented in the annex of DO-332.

IX. RTCA DO-333: Formal Methods Supplement to DO-178C and DO-278A

RTCA DO-333 states that formal methods are mathematically based techniques for the specification, development, and verification of software aspects of digital systems. The objective of the supplement is to provide additions to and modification of the DO-178C objectives, activities, explanatory text, and software life cycle data that apply when formal methods are used as part of the software life cycle. The supplement makes clear that formal methods may be used for all or just a small part of the verification process and may supplement other verification methods. In addition to the modifications of DO-178C, the supplement also provides clarifications on the use of formal methods through discussion papers contained in Appendix B.

The supplement requires that if formal methods are used to verify some aspect of the software development process, then the software plans must explain the intended usage. The Plan for Software Aspects of Certification (PSAC) should provide an overview of how formal methods will be used and what evidence those methods will provide. The Software Development Plan should provide details on the specific use of formal methods. Additions required to the Software Verification Plan (SVP) are especially important. All assumptions related to the use of formal analysis to detect errors and to verify functionality should be justified therein. The SVP should show that there are no verification gaps produced by the combination of formal and procedural analyses.

A. Formal Models.

DO-333 considers a formal model to be an abstract representation of certain aspects of a system (or code) for which the model notation uses precise, unambiguous, and mathematically defined syntax and semantics. The models may be graphical (e.g., state machine diagrams), differential equations, or computer languages. Because formal notations are precise and unambiguous, they can be used to assist verification by helping show accuracy and consistency in the representation of requirements and life cycle data. DO-333 does not require all of the requirements of a formal model to be formally expressed. However, if the high-level requirements and low-level requirements are both formally modeled, then formal analysis can be used to show compliance. DO-333 defines formal analysis as the use of mathematical reasoning to guarantee that properties are always satisfied by a formal model.

B. Automated Reviews and Analyses.

A substantial amount of the guidance provided by the Formal Methods Supplement consists of defining what DO-178C-required reviews and analyses can be augmented or replaced by formal analysis. DO-333 allows the use

of formal analysis to show satisfaction of the objectives for software output review and analysis. Formal methods may be used to optionally demonstrate the compliance of outputs to inputs, the accuracy of software representation, the consistency of requirements, the conformance to standards, the traceability of requirements to code, and/or the algorithmic correctness. Similarly, for both high-level and low-level requirements, formal analysis is an acceptable means of showing accuracy, consistency, verifiability, traceability, and conformance to standards. DO-333 also presents the specific aspects of software architecture and source code review and analysis that may be satisfied by formal analysis. The precise and unambiguous language of formal models helps reviews and analyses to also be precise and unambiguous.

C. Verification of Source Code and Executable Code.

DO-333 states that formal methods can be used in the verification of the source code or object code (or both) if the requirements for each code are formally expressed, if formal models of the codes exist, and if formal evidence demonstrates that the formal models of the codes satisfy the requirements. If formal models exist for both the source and the object codes, the verification of property preservation between source code and object code is allowed using formal analysis.

D. Coverage Analysis.

DO-333 discusses the ways in which formal analysis may be used to satisfy the coverage requirements of DO-178C and DO-278A. This guidance states that when any low-level testing is used to verify that low-level requirements for a software component are satisfied, then the DO-178C guidance for structural coverage analysis should be followed. When only formal methods are used to verify that low-level requirements are satisfied, then the guidance in DO-333 (section 6) applies. The supplement states that although it is possible to use a mixture of testing and formal analysis to show that verification evidence exists for all high-level and low-level requirements, no known test cases exist. In this case, the supplement permits certification authorities to approve software coverage if it can be demonstrated by a combination of methods that structural and requirements-based coverage have been achieved.

DO-333 requires that all assumptions made during the formal analysis are verified. It should be demonstrated that for all input conditions, the required output has been specified; and likewise, for all outputs, the required input conditions have been specified. Analysis test cases should provide evidence that the formal analysis achieves the required coverage level. All code structures must be shown to be covered by either formal or procedural analyses. DO-333 states that when a combination of formal methods and testing are used to assess coverage, functional (requirements-based) tests executed on the target hardware should always be done to ensure that the software in the target computer will satisfy the high-level requirements.

X. Conclusion

This paper provided an overview of the new certification guidance contained in RTCA DO-178C, DO-278A, DO-330, and the related supplemental documents created by RTCA SC-205 for the development of safety-critical airborne and CNS/ATM ground-based software. The objective of this paper was to help those not familiar with the new DO-178C documentation set to gain an appreciation for the scope of the information contained in the nearly 1000 pages of new guidance material from RTCA. A review of the DO-178B software verification guidance was presented prior to discussing the new material introduced in DO-178C and DO-278A. Following this, an overview of the new content contained in DO-178C for airborne software verification and in DO-278A for ground-based CNS/ATM software verification was discussed. Then the highlights of new guidance contained in the other documents supporting DO-178C and DO-278A were presented in subsequent sections. These other documents are:

- RTCA DO-248C[4]: Supporting Information for DO-178C and DO-278A
- RTCA DO-330[5]: Software Tool Qualification Considerations
- RTCA DO-331[6]: Model-Based Development and Verification Supplement to DO-178C and DO-278A
- RTCA DO-332[7]: Object-Oriented Technology and Related Techniques Supplement to DO-178C and DO-278A, and
- RTCA DO-333[8]: Formal Methods Supplement to DO-178C and DO-278A.

Although within the scope of this paper it was not possible to present every detail of the new guidance, it is hoped that the summary information contained herein will stimulate interest in these publications. The reader requiring specific information must download these documents from RTCA[9] in order to fully appreciate and apply the new guidance.

Acknowledgments

The new DO-178C, DO-278A, and companion documents are the work of RTCA Special Committee 205 (SC-205). Although the first credit goes to the RTCA staff, the organizing work of the SC-205 leadership team also deserves special recognition. The author especially wishes to acknowledge the effort and devotion of SC-205 chairs, Jim Krodel (Pratt & Whitney) and Gerard Ladier (Airbus), and executive committee members Barbara Lingberg (FAA-CAST Chair), Mike DeWalt (FAA), Leslie Alford (Boeing), Ross Hannan (Sigma Associates), Jean-Luc Delamaide (EASA), John Coleman (Dawson Consulting), Matt Jaffe (ERAU), and Todd White (L-3 Communications/Qualtech). Also deserving special recognition are the leads and co-leads of the subgroups: Ron Ashpole (Silver Atena), Ross Hannan (Sigma Associates), Frederic Pothon (ACG Solutions), Leanna Rierson (Digital Safety Consulting), Pierre Lionne (EADS APSYS), Mark Lillis (Goodrich GPECS), Herve Delseny (Airbus), Jim Chelinni (Verocel), Duncan Brown (Rollys-Royce), Kelly Hayhurst (NASA), David Hawkens (NATS), and Don Heck (Boeing). Several others who chaired these committees prior to publication of DO-178C and the SC-205 committee members themselves whose names are far too numerous to mention are cited in Appendix A of DO-178C (including the author of this paper).

The author's support of RTCA SC-205 was provided by the NASA Aviation Safety Program, the NASA Intelligent Resilient Aircraft Control (IRAC) Project, and the NASA System-Wide Safety and Assurance Technologies (SSAT) Project.

References

[1]RTCA DO-178B, "Software Considerations in Airborne Systems and Equipment Certification," December 1992.

[2]RTCA DO-178C, "Software Considerations in Airborne Systems and Equipment Certification," December 2011.

[3]RTCA DO-278A, "Software Integrity Assurance Considerations for Communication, Navigation, Surveillance and Air Traffic Management (CNS/ATM) Systems," December 2011.

[4]RTCA DO-248C, "Supporting Information for DO-178C and DO-278A," December 2011.

[5]RTCA DO-330, "Software Tool Qualification Considerations," December 2011.

[6]RTCA DO-331, "Model-Based Development and Verification Supplement to DO-178C and DO-278A," December 2011.

[7]RTCA DO-332, "Object-Oriented Technology and Related Techniques Supplement to DO-178C and DO-278A," December 2011.

[8]RTCA DO-333, "Formal Methods Supplement to DO-178C and DO-278A," December 2011.

[9]RTCA, www.rtca.org