DOI: 10.13009/EUCASS2025-730
11TH EUROPEAN CONFERENCE FOR AERONAUTICS AND SPACE SCIENCES (EUCASS)

MODEL BASED ENGINE CONTROL LAWS DESIGN PROCESS FOR SAFETY CRITICAL ENGINE CONTROL SYSTEMS

Aysegul Ceylan † *and Tugba Leblebici*
TUSAS Engine Industries Inc
Esentepe Blv. 356, 26210 Eskisehir, Turkiye
aysegul.ceylan@tei.com.tr† – tugba.leblebici@tei.com.tr

Abstract

This paper describes model-based engine control laws design process, including the explanation how SAE ARP4754A [1] and RTCA DO-331 [2] compliance are achieved throughout the model-based engine control laws development process. Behaviour Model is developed for model-based engine control laws design process, of which model includes controller and Plant Model. The contributions of the Behaviour Model for system requirement development and requirement validation process of ECS (Engine Control System) are discussed.
The transition process of model-based engine control laws design to Engine Control Law (ECL) software and the advantages of using a common library to adapt to RTCA DO-331 [2] are mentioned as well.

1. Introduction

Engine Control System (ECS) means any system or device which is part of the engine type design, which controls, limits or monitors engine operation and is necessary for continued airworthiness of the engine [3]. The ECS plays a pivotal role in ensuring the safe and efficient operation of an engine. Given its inherent complexity, encompassing numerous interconnected components, and functionalities, along with its safety-critical nature, the development of ECS demands strict adherence to rigorous safety, certification and guidelines. Notably, these guidelines are outlined in SAE ARP4754A [1], which governs system-level development processes for engine systems.

At the core of the ECS lies the Engine Control Laws (ECL), which dictate the fundamental operational logic of the engine. ECL constitute a fundamental part of complex and safety critical engine control system and must be developed in accordance with certification specifications (CS-E), SAE ARP4754A [1] and RTCA DO-178C [4] guidelines. The intricate process of designing these engine control laws algorithms and their subsequent software implementation presents numerous challenges. Control Engineers must contend with the nonlinear and time-varying dynamics of the engine, ensure real-time performance constraints are met, and build robustness against hardware limitations like fixed-point arithmetic. Furthermore, maintaining compliance with traceable and verifiable requirements throughout the development lifecycle is essential. These profound complexities and stringent requirements render traditional document-centric development approaches less suitable, highlighting the urgent need for more integrated, systematic, and simulation-driven development processes. To effectively navigate these challenges, meet demanding standards, and guarantee system safety and reliability, Model Based Design (MBD) has emerged as a central and indispensable methodology in the development of safety-critical systems, including Engine Control Systems and Flight Control Systems.

Model Based Design (MBD) is an engineering methodology that represents physical system behaviours through mathematical equations and visual diagrams. This method enables early modelling, validation, and verification of system behaviours, thereby reducing system development effort and enhancing system reliability. There are many benefits to a model-based development concept, including the use of common design artifacts and the facilitation of collaboration among participants of a development effort [7]. MATLAB/Simulink tools are widely employed in the control engineering domain to effectively model system dynamics, perform simulations, and facilitate model-based software development.

This paper presents a structured development approach for ECS, focusing on the creation of the Behavioural Model to define control laws strategies and transitions. The Prototype model is then refined according to software constraints, transformed into the Design Model compliant with discrete-time and fixed-point requirements, and verified through qualified analysis tools prior to automatic code generation. The objective is to demonstrate how MBD enables an efficient, traceable, and certifiable workflow for safety-critical control system development.

DOI: 10.13009/EUCASS2025-730

1.1 A Systematic Approach to Model-Based Design Workflow Selection

A model is a collection of one or more artifacts that represent a concept. When the concept is a system, the model is an abstraction whose form and content are chosen for the purpose of understanding, communicating, explaining, or designing aspects of interest of the system [6].

The scope, depth, and fidelity of a model must be chosen to fit the purpose. Models can be descriptive or analytical, and they can capture static properties of a system (e.g., hierarchical decomposition, interconnection) or dynamic properties (e.g., behaviour). Characteristics of successful system models include:

- providing a framework to attack a problem in a coherent and consistent manner;
- having the power to show that a solution satisfies the needs of the stakeholders;
- providing integrity and consistency to the system; and
- providing insight into the problem and comparative advantages of different possible approaches and solutions [7].

RTCA DO-331 [2], which is supplement of RTCA DO-178C [4] for Model Based Development and Verification, classifies models into two categories based on their purpose and level of abstraction:

- **Specification Model**: represents the intended system behaviour; high abstraction; used for requirement specification and functional validation
- **Design Model:** captures implementation details; low abstraction; used for code generation and detailed simulation.

The distinction is fundamental, as it directly influences the role of the model within the development lifecycle and determines the appropriate verification strategies. Table 1 summarizes the key distinctions between Specification Model and Design Model as defined in RTCA DO-331 [2], including their level of abstraction and typical usage within model-based development process.

Table 1 Comparison of Specification Model and Design Model

Feature	**Specification Model**	**Design Model**
Purpose	Define the intended behaviour of the system	Specify implementation details of the software
Abstraction Level	High-Level	Low-Level
Content	Functional Requirements, performance, interfaces, safety characteristics	Algorithm, internal data structures, control and/or data flow
Usage	Simulation, analysis, verification	Code generation, detailed simulation, design-level verification
Code Generation	Not used	Used for automatic code generation

As outlined in RTCA DO-331 [2], five fundamental model-based development examples are defined (see Figure 1, Table MB.1-1 in RTCA DO-331 [2]). Selecting the most suitable example is critical activity to ensure both compliance and development efficiency.

DOI: 10.13009/EUCASS2025-730

<table>
<tr><th>Process that generates the Life Cycle Data</th><th>MB Example 1</th><th>MB Example 2</th><th>MB Example 3</th><th>MB Example 4</th><th>MB Example 5</th></tr>
<tr><td>System Requirement and System Design Processes</td><td>Requirements allocated to software</td><td>Requirements from which the Model is developed</td><td>Requirements from which the Model is developed</td><td>Requirements from which the Model is developed</td><td>Requirements from which the Model is developed</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td>Design Model</td></tr>
<tr><td rowspan="2">Software Requirement and Software Design Processes</td><td>Requirements from which the Model is developed</td><td>Specification Model</td><td>Specification Model</td><td>Design Model</td><td rowspan="2"></td></tr>
<tr><td>Design Model</td><td>Design Model</td><td>Textual Description</td><td></td></tr>
<tr><td>Software Coding Process</td><td>Source Code</td><td>Source Code</td><td>Source Code</td><td>Source Code</td><td>Source Code</td></tr>
</table>

Figure 1: Model Usage Examples from RTCA DO-331[2] Table MB.1-1

- MB Example 1 development workflow is closely mirrors traditional RTCA DO-178C [4] workflow. In this example High Level Requirements (HLRs) are derived from system requirements and Design Model defines software design and Low-Level Requirements (LLRs).
- In MB Example 2 and MB Example 3 development processes, Specification Model defines HLRs, the Design Model or textual LLRs are developed. MB Example 2 use both specification model and design model. In MB Example 3, auto code generation is not used and LLRs are written in textual format. These examples are less commonly adopted industry due to complexity and repetitions.
- In MB Example 4 development workflow, Design Model replaces both HLRs and software design; suitable when Design Model directly represents functional behaviour.
- In MB Example 5 development workflow, Design Model is developed during system development and reused in software development. This is ideal for control laws development, where models are created at system level to reflect dynamic behaviours (e.g. stability, transients) and reused as software development. This example supports the special use case of control laws in software development [8].

Textual HLRs are often represented through constructs such as state diagrams, truth tables etc., these requirements are very closed to models, so boundary between HLRs and models becomes blurred in model-based development process. In such cases, separate HLRs are just an artificially introduced level leading to “copy-and-paste development” [8], where redundant documentation replaces meaningful abstraction. Creating separate textual HLRs that merely replicate the behaviour already defined in the model adds little engineering value.

In MB Example 4 and MB Example 5, it is possible to use system requirements that drive the model design in place of separate textual HLRs. Thus the system requirements take over the role of HLRs for RTCA DO-331 [2]. These system requirements are developed in accordance with SAE ARP4754A [1] guideline and when meet the criteria of development of requirements according to requirement standard specified by RTCA DO-331 [2], are recognized by RTCA DO-331 [2] as valid HLRs. This approach eliminates repetition, maintains requirement integrity, and supports a seamless transition from system intent to model implementation within a model-based development process.

MB Example 5 is well-suited to control laws development. Engine control laws must be designed at the engine level, as they require a holistic understanding of the entire engine system and are derived from engine-level handling qualities such as stability and transient behaviour. Control laws models are already developed at system level using model based design tools for simulation and optimize laws purposes. However these models may not initially be structured for software development.

MB Example 5 supports reusing these system-level models within the software development workflow, creating continuity between system-level design and software-level implementation. This reuse must be planned carefully, ensuring clarity in roles, responsibilities, verification strategy and adherence to both SAE ARP4754A [1] and RTCA DO-331 [2] guidelines.

DOI: 10.13009/EUCASS2025-730

2. Model Based Design Process

Figure 2 illustrates the comprehensive model based development lifecycle, as RTCA DO-331 [2]. The life cycle is meticulously structured into three interconnected main processes:

- Planning Process: Establishes the foundation framework for software development and verification through documents such as PSAC, SDP and SVP.
- Development Process: Initiates with the precise definition of requirements, progressing through design, coding, and integration phases.
- Integral Process: Comprising continuous activities like verification, configuration management, quality assurance, and certification liaisons, these are applied consistently throughout the entire development life cycle to ensure safety, traceability, verifiability, and certifiability.

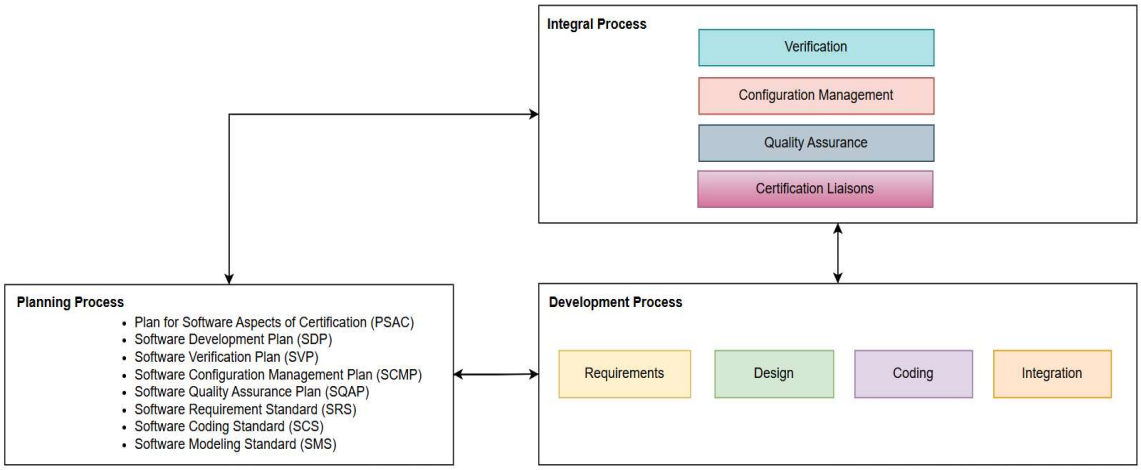

Figure 2: Model Based Development Lifecycle Process

Engine Control System is developed based on system-level requirements derived from the analysis of high-level engine specifications. Among these system requirement, those allocated to software components are defined as SRATS (system requirements allocated to software). While system development process is conducted in accordance with SAE ARP4754A [1] guideline, the software development process is supported by RTCA DO-178C [4] and its supplement RTCA DO-331 [2], which addresses model-based development and verification.

Engine Control Laws (ECL) requirements, as SRATS, are developed and managed at the system level using Requirement Management Tool to ensure their traceability and control. Based on these detailed requirements, Prototype Model is developed using Simulink/Stateflow. This model represents the intended behavioural logic of the system.

Prototype Model is then integrated with Plant Model, which reflects the dynamic behaviour of the engine system, to form Behavioural Model. This integrated modelling environment, model-in-the loop (MIL) simulation, is used for early stage evaluation and validation of ECL requirements. This allows engineers to evaluate the control strategy in a virtual setting before entering implementation phase. This early validation is a key advantage of MBD, significantly reducing potential rework and improving overall system quality.

A key feature of this process is that Design Model used in software development lifecycle is derived directly from Prototype Model without the need for a separate modelling effort. Thus, Design Model is developed from system-level requirements, without the need for separate set of textual High-Level Software Requirements (HLRs). This approach align with Model Usage Example 5 as defined in RTCA DO-331 [2]. In this way, the artificial separating between model and requirements documentation is eliminated. By treating model as the primary expression of requirements, strong traceability is maintained from system intent to implemented behaviour. This enables efficient verification and validation directly at the model level, without duplication effort.

DOI: 10.13009/EUCASS2025-730

Following the creation of Design Model, Matlab Coder/Simulink Coder/Embedded Coder tools are used to automatically generate source code. This code is then compiled into Executable Object Code (EOC) using a target compiler as part of the integration process.

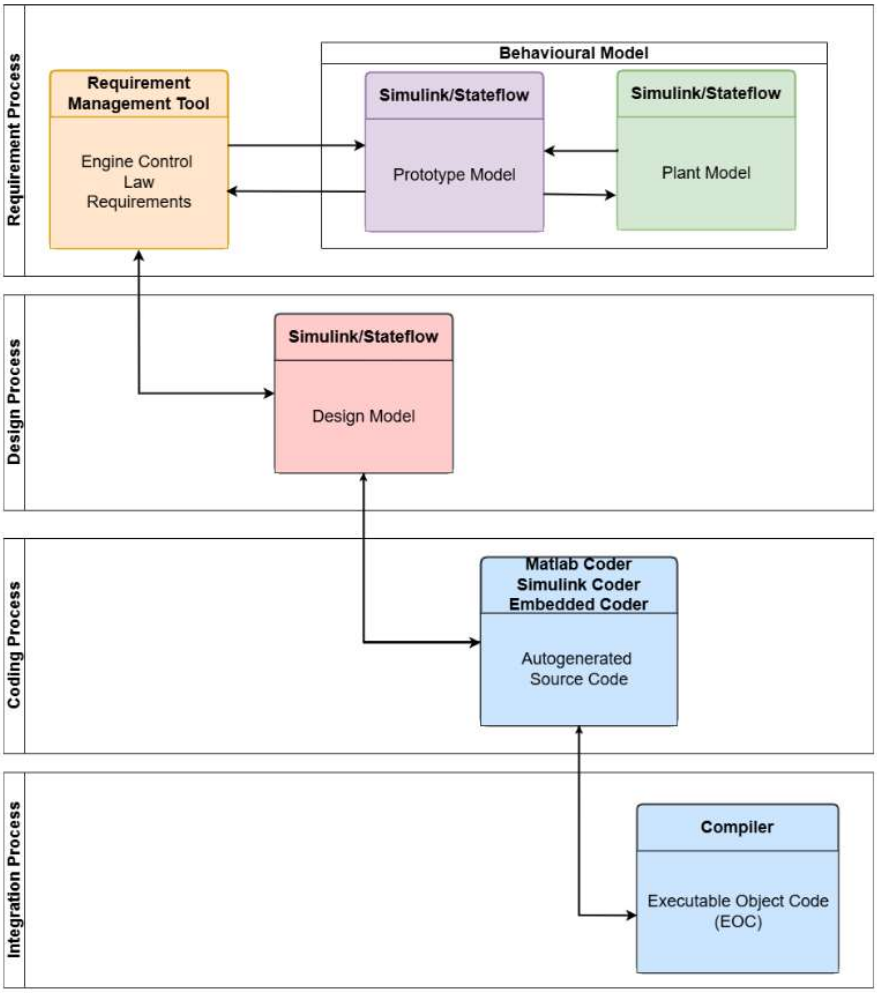

Figure 3: Model Based Design Development Process

Figure 3 visually reinforces this process flow, highlighting how the synergy between Prototype Model and Plant Model results in the central Behavioural Model, pivotal for ECL requirements development and validation. It further underscores the seamless integration of SAE ARP4754A [1] system processes with RTCA DO-331 [2] model based software development processes.

2.1 Model-Based Development Environment

Model-Based Development (MBD) is a systematic approach that integrates systems engineering with software engineering to develop control algorithm in a traceable and verifiable manner, enabling automatic code generation. This approach is especially critical in safety-critical systems and is carried out in alignment with software lifecycle defined at RTCA DO-331 [2].

In the system development process, Plant Model, which represents the behaviour of the physical system, serves as a fundamental building block for design and verification of control system. Particularly in domains such as aerospace and defence, where high reliability is essential, Plant Model mathematically represents physical processes- such as engine dynamics, mechanical actuators, sensor feedbacks, and environment effects- through differential equations, transfer functions, or state-space representations. The goal of this model is to provide a virtual testing environment

DOI: 10.13009/EUCASS2025-730

where the behaviour of control algorithms under various operating conditions can be assessed without deploying them on actual hardware.

SAE ARP6538 [5] standard systematically defines the roles, types, and usage purposes of models like Plant Model within the system development life cycle. According to this standard, Plant Model is not only a simulation tool but also a reference model used for verification of system-level requirements. By modelling dynamic system responses, disturbances, and sensor/actuator limitations, Plant Model becomes a reliable reference to verify both the design validity and performance accuracy of control algorithms.

In accordance with RTCA DO-331 [2], model-based development begins with system-level requirements and ensures that the model represents design representation. These models are not just design representations but they must also serve as verifiable specifications. The process start with development of Prototype Model, which defines the fundamental logic of the control laws, including its input-output relationships and behavioural characteristics.

Engine Control Laws (ECL) requirements serve as the foundational reference for developing Prototype Model. Derived from high-level engine specifications, ECL requirements articulate essential control system properties such as functionality, stability, observability, and acceptability. These requirements also provide formal bridge between system-level development and software design-level implementation.

Following the construction of Prototype Model, it is integrated with Plant Model to create Behavioural Model. This integration enables model-in-the-loop (MIL) simulations, which are used to evaluate control logic under various operating conditions. These simulations validate system's closed loop behaviour, stability margins, timing sensitivities, command responses, and compliance with functional requirements before deployment to hardware platform. ECL requirements are further refined and validated at this stage.

Requirements are systematically classified and evaluated requirements using seven verification domains proposed by Arnold et al. [9], originally developed Flight Control Laws (FCL):

1. Control law structure
2. Criteria for control law design
3. Input filtering and command shaping
4. Open-Loop behaviour
5. Logic circuits
6. Output transformations
7. Filter design

This classification provides a unified framework to determine which modelling techniques – such as transfer functions, and lookup tables – are most appropriate for each requirement type. It enables both modelling and verification activities to be systematically aligned, improving traceability, automation potential. Similarly, this classification is applicable to ECL requirements, which can be systematically modelled and verified using these methods.

Importantly, RTCA DO-331 [2] distinguished between system-level requirements and control law design-level specifications. For instance, stating specific PID gains (e.g., "Kp= 3, Ki=1.3") is considered a design detail, not a valid system requirement. Instead, acceptable system-level requirements might specify behaviours such as: "The system shall respond to pitch reference change with a settling time less than 2 seconds and a maximum overshoot of 5%." This principle reinforces the ECL's role in translating abstract system-level performance goals into structured model behaviour.

Once validated, Prototype Model evolves into Design Model, configured for real-time embedded execution. Design Model must be developed as a fixed-step, discrete model to synchronize with deterministic timing of embedded processors. In this phase, Design Model must conform to RTCA DO-331 [2] design objectives, which govern model architecture, data integrity and flow, execution determinism, and compatibility with automatic code generation. These rules are usually formalized in Software Modeling Standard (SMS) and Software Coding Standard (SCS) documents which defines modelling practises and constraints that are both RTCA DO-331 [2] compliant and suitable for embedded software development.

Following the development of Design Model, a series of verification and analysis activities must be performed to ensure that the Design Model is correct, complete and compliant with Software Model Standard. These activities are

DOI: 10.13009/EUCASS2025-730

essential to demonstrate that Design Model is suitable for further stages of development life cycle. As a first step, static analysis is conducted to examine the model structure without executing it. This analysis involves checking for:

- Model Advisor checks model structure and style against RTCA DO-331 [2] - conformant modelling guidelines and standards, also it analyses complexity, modularity, readability, and other quality metrics to assess verification readiness within Simulink Check tool.

- Simulink Design Verifier performs formal analyses to identify logical inconsistencies, unreachable states such as integer overflow, dead logic and division by zero.

The behavioural correctness of Design Model is validated not only MIL simulation but also by ensuring traceability to ECL requirements. Structural correctness, code generation readiness, and semantic alignment with requirements are critical success factors in this stage.

Once Design Model passes verification, automatic C/C++ code generation suitable for production can be performed using Matlab Coder, Simulink Coder and Embedded Coder tools. Prior to code generation, the model is converted to a fixed-point representation to ensure compatibility with the target microcontroller's processing capabilities. The generated code is tested in Hardware-in-the-Loop (HIL) environments to assess both functional correctness and real-time execution feasibility.

According to RTCA DO-178C [4], if a tool is used in way that its output is not independently verified, for example automatic code generation, the tool must be qualified to ensure its reliability and correctness. RTCA DO-330 [10] which is supplement of RTCA DO-178C [4] defines tool qualification process details. It is crucial to note that all tools used in the model-to-code process- such as Simulink Design Verifier or Embedded Coder- must be qualified according to RTCA DO-178C [4] For this, Math Works Tool Qualification Kit supports tool certification and documentation required for safety-critical projects.

This end-to-end process aligns directly with MB Example 5 defined in RTCA DO-331 [2]. In this scenario, system-level requirements are directly translated into a model, all verification activities are performed on model, and production code is automatically generated from it. This approach transforms the model into not a design tool but an integrated artifact for behavioural specification, requirement validation, and software implementation.

3. Conclusions

The development of ECS demands a rigorous, traceable, and certifiable engineering approach due to their complexity and safety-critical role in aerospace systems. Traditional document-centric methods often fall short in managing the dynamic, nonlinear, and timing-sensitive characteristics of engine control laws. In contrast, Model-Based Development (MBD), particularly as defined by RTCA DO-331 [2] MB Example 5, provides a robust framework for addressing these challenges.

This paper demonstrated how system-level requirements can be effectively translated into executable control software through a systematic MBD workflow. The use of Prototype Model and Plant Models to construct Behavioural Models ensures early-stage validation and verification, while the transformation into Design Model enables compliance with embedded software constraints and automatic code generation. The reuse of system-level model as software artifacts bridges the gap between system design and implementation, significantly reducing repetition and ensuring consistency.

By eliminating the artificial separation between textual HLRs and functional model, MBD allows models themselves to serve as verifiable and traceable representations of system intent. This integrated approach not only streamlines requirement validation and verification but also supports certification efforts by aligning with industry standards, guidelines such as SAE ARP 4754A [1] and RTCA DO-331 [2].

In conclusion, adopting an MB Example 5 based development strategy enables the creation of highly reliable, and maintain ECS architectures. It fosters collaboration across system and software domains, supports automation, reduces manual error, an ultimately contributes to safer and more efficient control systems.

DOI: 10.13009/EUCASS2025-730

References

[1] ARP4754A.2010. Guidelines for Development of Civil Aircraft and Systems. S-18 Aircraft and Sys Dev and Safety Assessment Committee, SAE International.
[2] RTCA DO-331. Dec. 2011. Model-Based Development and Verification Supplement to DO-178C and DO-278. Radio Technical Commission for Aeronautics, RTCA.
[3] CS-Definitions Amendment 2. Definitions and abbreviations used in Certification Specifications for products, parts and appliances, EASA.
[4] RTCA DO-178C. Dec. 2011. Software Considerations in Airborne Systems and Equipment Certification. Radio Technical Commission for Aeronautics, RTCA.
[5] ARP6538.2018. Dynamic Modeling of Aerospace Systems (DyMAS), SAE International.
[6] Guide to the Systems Engineering Body of Knowledge (SEBok). http://www.sebokwiki.org/wiki/Guide_to_the_Systems_Engineering_Body_of_Knowledge_%28SEBoK%29. Accessed March 16, 2014.
[7] Torres-Pomales,W. Dec. 2014. "Is Model-Based Development a Favorable Approach for Complex and Safety-Critical Computer Systems on Commercial Aircraft? NASA, Langley Research Center.
[8] Markus Tobias Hoschstrasser, Nov. 2020. “Modular model-based development of safety critical flight control software”. Munchen Technical University
[9] A. Arnold, R. Luckner, 2023 G. Weber, “Classification of Flight Control Law Requirements in the Context of Model Based Software Development and Verification”. Liebherr-Aerospace Lindenberg GmbH
[10] RTCA DO-330. Dec. 2011. Software Tool Qualification Considerations. Radio Technical Commision for Aeronaustics, RTCA.