Best practices for developing DO-178 compliant software using Model-Based Design

Raymond G. Estrada, Jr.[1]
The MathWorks, Torrance, CA

Eric Dillaber.[2]
The MathWorks, Natick, MA

Gen Sasaki[3]
The MathWorks, Torrance, CA

Model-Based Design with automatic code generation is an established approach for developing embedded control systems and is now commonly used on applications that must satisfy the commercial aviation software standard DO-178B. Model-Based Design enables engineers to create advanced embedded software systems using an executable model as the basis for design, simulation, and software verification and validation. The complexity of modern aerospace systems has led to an increased use of Model-Based Design across the full software development life cycle from early verification to implementation and system integration. Maximizing the benefits of Model-Based Design in the context of satisfying the objectives of DO-178B (and DO-178C upon acceptance by the FAA) requires a level of expertise that often takes years of hands-on experience to acquire.

I. Introduction

IN this paper we share a set of best practices that are critical to the success of organizations completing projects certified to DO-178B and DO-178C using Model-Based Design. The best practices describe key considerations, methods, and fundamental capabilities of Model-Based Design that span the software development process from modeling and simulation to code generation, verification and validation, and implementation. These best practices have been distilled from years of collaboration with customers in the aerospace industry in developing high-integrity software. Our intent is to help organizations avoid common pitfalls and reduce the time, effort, and costs required to develop high-integrity software that satisfies DO-178 objectives.

DO-178B and the newly-released DO-178C have clearly defined objectives for software life cycle process activities such as software requirements definition, software design, coding, integration, verification, and configuration management. Tables A-1 through A-10 in Annex A of the standard list the outputs required for each objective. The outputs are generated based on the software integrity level and require verification prior to use for certification. The tables and DO-178B [7] document, however, provide little in the way of guidance on the use of models, leaving aerospace companies to determine how to best apply Model-Based Design for developing DO-178 compliant software. The release of DO-178C brings a supplement, RTCA DO-331 "Model-Based Development and Verification Supplement to DO-178C and DO-278A" [6], containing modifications and additions to the objectives, activities, and software life cycle data that should be addressed when using Model-Based Design. DO-331 provides additional guidance on the artifacts generated using models and the corresponding verification evidence. While this guidance is valuable, engineering teams need an efficient workflow to identify errors earlier, quickly generate objective artifacts, eliminate manual processes using tools that can be qualified, and reduce the number of software errors during system integration. The following best practices can be adopted to implement such a workflow for developing software that satisfies DO-178 objectives using Model-Based Design:

[1] Embedded Code Generation and Certification.
[2] Embedded Code Generation and Certification.
[3] Embedded Code Generation and Certification.

- Establish a standardized Model-Based Design environment for development and verification
- Define a configuration management strategy that includes Model-Based Design artifacts
- Define an optimized Model-Based Design process for DO-178
- Determine whether models represent high-level or low-level requirements
- Use simulation and analysis to support DO-178 objectives
- Analyze requirements-based testing at a higher level of abstraction
- Employ Model-Based Design tools that can be qualified

II. Best Practices

A. Establish a standardized Model-Based Design environment for development and verification

DO-178 requires a standardized environment among developers that enables them to reproduce any particular version of the software [5]. An inconsistent software development environment can produce source and object code that that does not fully implement the requirements. The establishment of a standardized Model-Based Design environment helps to ensure that design models can be used to generate consistent, reproducible simulation and test results as well as source and object code that can be traced to requirements.

While most organizations with experience using traditional methods have already-established standardized environments for tooling, establishing an initial Model-Based Design environment can be a challenge due to the extensive tooling and customizations available to support the additional capabilities provided with Model-Based Design. Tooling is no longer confined primarily to points in the software development process but rather is used in a continuous fashion. As a result, the contents of the Model-Based Design environment must be well thought out to most effectively support the project and reduce maintenance.

Much like a compiler whose outputs are influenced by a separate linker file, the outputs of simulation and code generation are influenced by factors such as configuration settings, customization files, and other dependencies (Figure 1). These dependencies are often stored in multiple formats and are typically located at multiple hierarchical levels within a development environment.

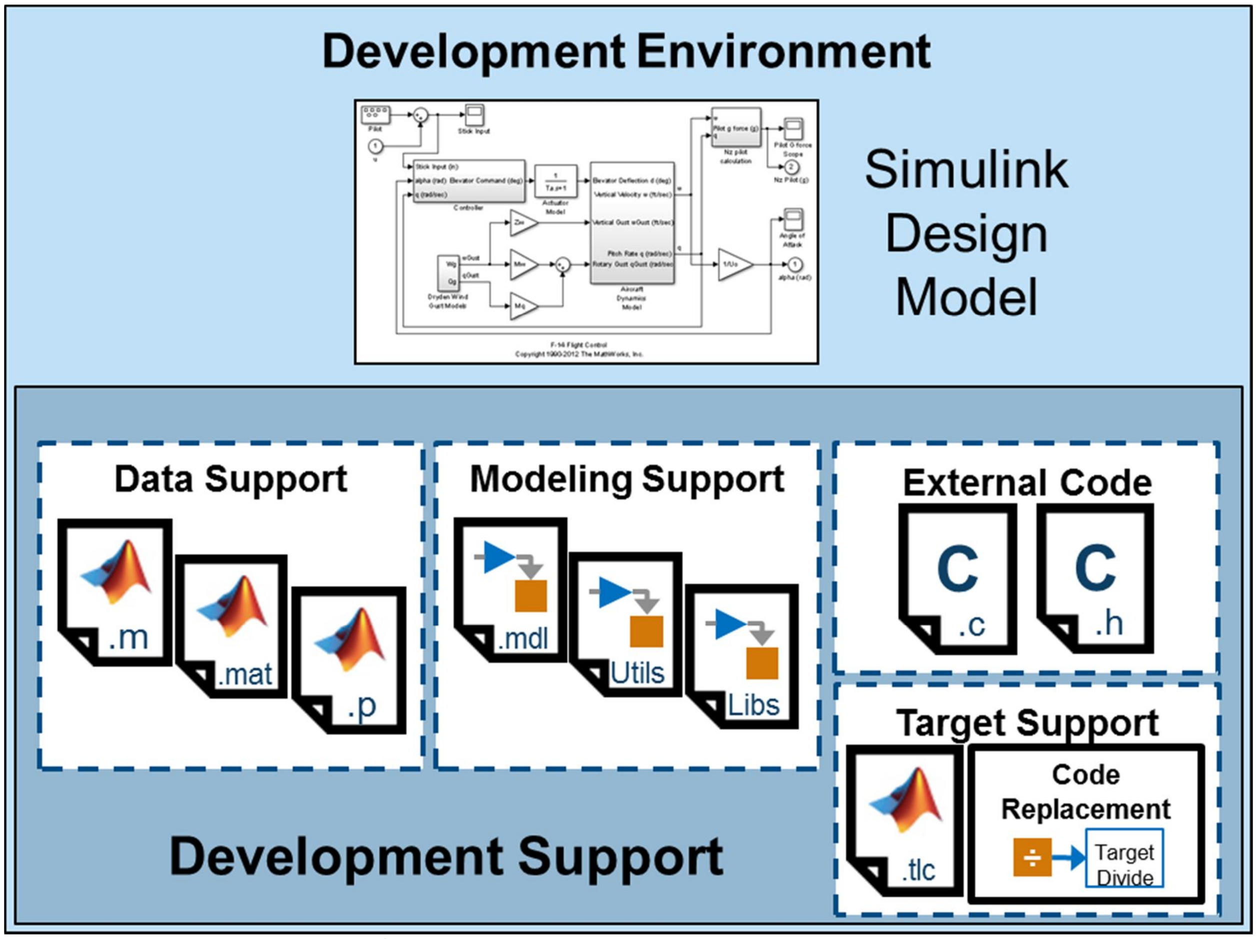

Figure 1 Software Development Environment with Model-Based Design

A best practice is to automate the installation and setup of the development environment for all developers. For new users or for a fresh installation of the development environment, a batch script can be used for automatic installation of:

1. Correct versions of development tools
2. Correct compiler version
3. Model development libraries
4. Data files and scripts

In addition, initialization scripts that execute automatically upon startup can help to ensure a consistent development environment by automatically performing tasks such as:

1. Verification of correct tool versions, libraries, patches, and so on, as specified in the Software Development Plan (SDP)
2. Addition of paths to all design model dependencies
3. Execution of additional setup and support scripts
4. Definition and loading of data

An example of this is a Windows desktop shortcut configured to execute a top level startup script that initializes the development environment. Automatic installation, set up, and configuration helps to establish the correct set of design model dependencies to be used.

A standardized software development environment includes a common, shared library from which the design models are developed. Primitive blocks (for example, gain and sum blocks) often have multiple configurable options available to developers, which can lead to inconsistent use of block options across design models. The Simulink gain block, for example, has multiple configuration options available such as sample time, signal, and parameter attributes. A library containing a subset of the available primitives should be created for common use. A custom Model Advisor check can be used to verify that the correct block parameters are used consistently throughout the design model.

B. Define a configuration management strategy that includes Model-Based Design artifacts

A standardized Model-Based Design environment alone is not sufficient to reproduce consistent software life cycle data. Typical source model dependencies include support libraries, data files, scripts, and more. The dependencies are required for simulation, testing, verification, and code generation. DO-178 states that software life cycle data must be placed under configuration management (CM) [5], as described in the plan for software aspects of certification (PSAC). Establishing a CM strategy that supports Model-Based Design and is consistent with the PSAC can, in conjunction with a standardized design environment, allow for consistent, reproducible simulation and test results, source, and object code.

The "golden rule" of configuration management with Model-Based Design artifacts is that the model is the source of design information. Parameter data applied to the model may also be considered as part of the model in this context, but should be managed using separate data files or scripts. Derived artifacts such as C code and executable files should always be generated from the source model. Representative types of Model-Based Design artifacts are shown in Figure 2.

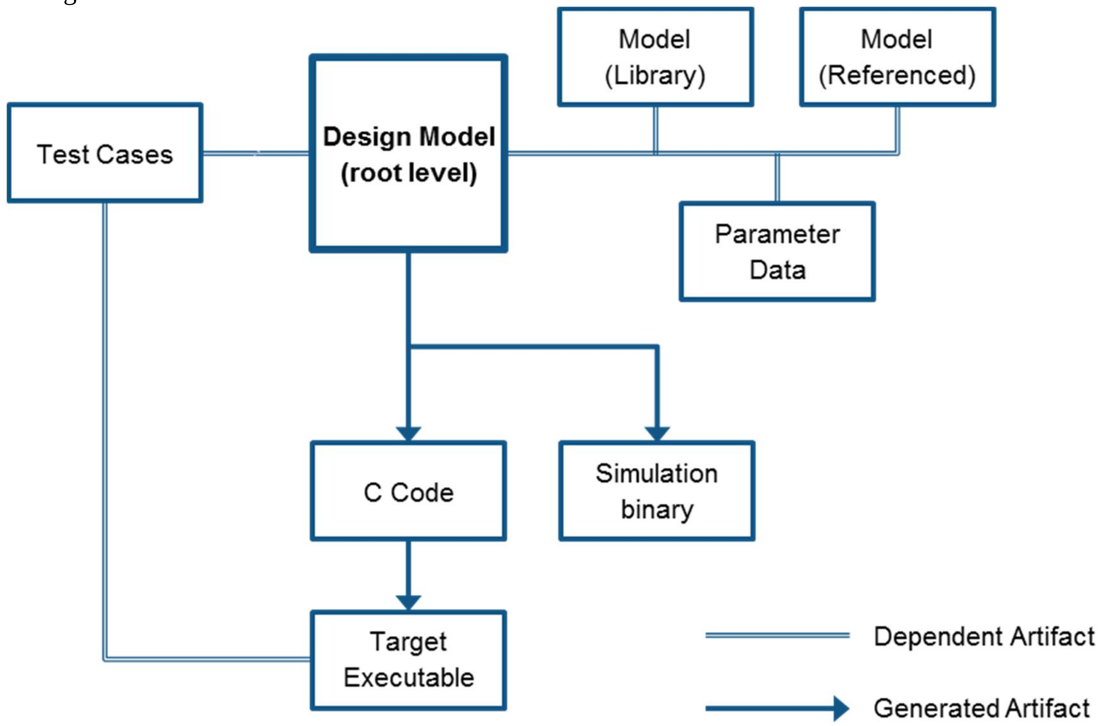

Figure 2 Representative Model-Based Design Artifacts

With respect to DO-178, the following considerations are particularly important in a CM process:

- Design artifacts archived in CM repository must be synchronized
- Design artifacts should be checked against standards before being checked-in to the CM repository

The following paragraphs highlight common issues faced by engineers in defining a CM process, and some best practices to address them.

Synchronize artifacts

With Model-Based Design, many artifacts, such as C code and executable files, can be automatically generated from the model and associated parameter data. In a process that must comply with DO-178, these artifacts should be archived so that they may be recalled in the future. This is useful to identify changes in derived artifacts as the sources from which they are generated are updated in subsequent iterations of the design process.

To help ensure this is done in a consistent and reproducible manner, a process to verify that all artifacts are synchronized with their source files before being checked-in to the CM repository should be established. The process should, for example, prevent modified Model-Based Design source files, such as models and data files, from being checked in without the corresponding auto-generated code. In addition, the process should prevent the check-in of modified auto-generated artifacts, such as C source and header files, so that the design model and its dependencies are the only source of all auto-generated files. If non-synchronized artifacts do get checked-in, it may hinder or prevent debugging efforts by not allowing engineers to replicate past designs. In addition, the purpose of each generated artifact stored in the CM repository should be understood and documented. For example, it should be documented that a System Design Description (SDD) Report, generated from the design model and its dependencies, is used for model verification and traceability activities.

Perform standards checking of artifacts prior to check-in

While not mandated by DO-178, it is a best practice to check development artifacts for compliance with established modeling and coding standards prior to check-in to the CM repository. Examples of established Simulink modeling standards are those published by NASA (Orion GN&C) and the MathWorks Automotive Advisory Board (Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow). For C code, the MISRA C standard is an available option. In addition, teams should verify that models can be simulated and that code can be generated from design models, if possible, prior to check-in.

Formal verification activities must be performed on the configured model and its derived artifacts. This best practice helps to facilitate and streamline verification activities by helping to ensure review of the correct artifacts. A checked-in model that does not comply with standards, for example, may fail to simulate or generate code. This prolongs formal verification activities because the model must be updated and the artifacts regenerated prior to resuming formal verification activities. Employing Model-Based Design tools, such as Model Advisor, can help to reduce or eliminate such delays through automatic checking of artifacts against standards prior to check-in.

C. Define an optimized Model-Based Design process for DO-178

Models should be used for multiple purposes across the complete development life cycle to maximize process efficiency gains, increase the capacity to manage complexity, and improve ROI. An optimized Model-Based Design approach incorporates simulation and analysis, code generation, requirement linking and traceability analysis, automated code reviews, automated document generation, model coverage, and formal methods for verification.

The level of software requirements represented by models should be determined prior to the implementation of the project-specific software design process that uses Model-Based Design. Additional information and best practices can be found in section [**Determine whether models represent high-level or low-level requirements**]. One approach used in the aerospace industry is to develop models derived from textual high-level requirements (HLR). These models represent the low-level requirements (LLR) as indicated in “MB Example 1” in Table MB.1-1 (Model Usage Examples) in DO-331 [6] and are design models.

The software design process using Model-Based Design is ideally implemented upon establishing the initial set of software HLR. The proceeding sections describe several key areas requiring a high level of effort in which Model-Based Design offers advantages over traditional methods

Earlier verification of design

Software development with Model-Based Design facilitates early verification of the design. Verification at the model level helps to reduce the number of requirements and design errors found during hardware and software testing and beyond. Desktop simulation of the design models is used to perform non real-time simulation using requirements-based test cases. Model Coverage, required for software levels A, B, and C of DO-178C, is used in conjunction with simulation to identify any requirements expressed by the design models that were not exercised.

The verification of LLR compliance with the HLR and conformance to standards is typically accomplished through manual design reviews. When using Model-Based Design, some of activities performed during traditional design reviews can be reduced or automated. The Model Advisor tool, for example, performs an automated review of the model to assess conformance to modeling standards. Manual inspection of the items not covered by the Model Advisor checks is necessary. Model compliance to high level requirements can be assessed using simulation, an approach that can be more effective than traditional design reviews in finding real issues in the design.

Reduction or elimination of manual code reviews

Simulink Code Inspector can be used to satisfy objectives in table MB.A-5 in DO-331 that are typically satisfied through manual code reviews. This tool uses formal methods to systematically examine blocks, parameters, and settings in a model to determine whether they are structurally equivalent to operations, operators, and data in the generated code. The detailed bidirectional traceability analysis between the model and code can identify issues missed in manual code reviews. The resulting structural equivalence and traceability reports are archived and made available for audit by the certification authorities as proof of compliance with the objectives in table MB.A-5 in DO-331. To maximize the effectiveness of Simulink Code Inspector, design models should be developed using the subset of supported primitive blocks.

Verification of executable object code

Processor-In-the-Loop (PIL) tests are used to verify high-level, low-level, and derived requirements through execution of requirements-based test cases on target hardware (Figure 3). The existing set of requirements-based test cases is reused for this activity. This can eliminate the need to develop and review additional test cases and code often required for object code verification using traditional methods. A code coverage tool is used in conjunction with PIL to assess coverage and produce a code coverage report. Polyspace static code analysis tools are used to prove the absence of certain run-time errors in the source code and produce the required documentation. The test results, code coverage, and run-time error reports are archived and made available to the certification authorities as proof of compliance with the objectives in table MB.A-6 and MB.A-7 in DO-331.

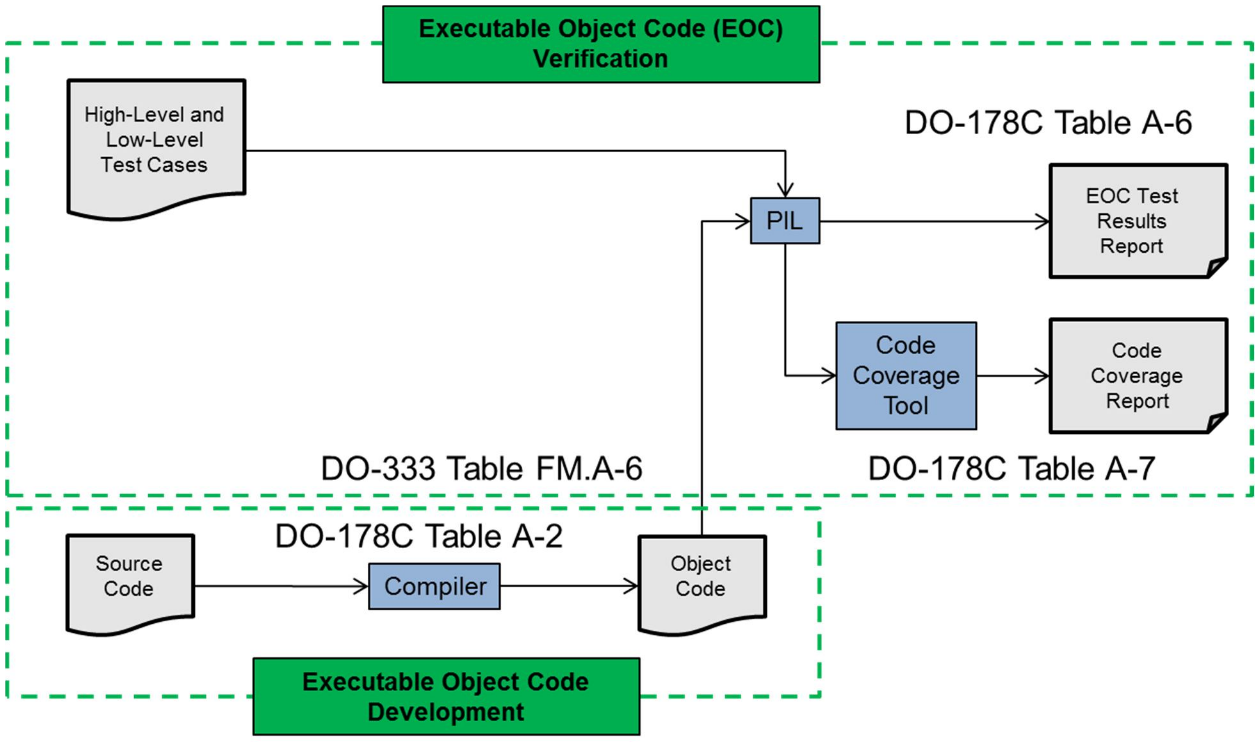

Figure 3 Executable Object Code (EOC) Verification

Generation of proof of verification of DO-178C objectives

Proof of compliance must be generated and made available to certification authorities for objectives in Tables A1-A10 of Annex A in DO-178C. Under traditional methods the required artifacts are often developed manually, an activity that can be both time-consuming and error-prone. Under an optimized software development process using Model-Based Design, a number of the required artifacts are created through automatic report generation. Table 1 lists documents generated for various workflow activities.

Table 1: Generated Artifacts

Workflow Activity	Document
Model Traceability	System Design Description Report
Model Conformance	Model Standards Report
Model Verification	Simulation Test Report System Design Description Report
Source Code Traceability	Code Inspection Traceability Report
Code Conformance	Code Standards Report
Code Verification	Code Inspection Report
Low-Level Verification	Executable Object Code Test Results Report
High-Level Verification	Executable Object Code Test Results Report
Object Code Traceability	Code Generation Report Object code listing (Third-party IDE or compiler)

Automatic generation of the required artifacts refocuses effort from developing to reviewing them. Tools that can be qualified, such as Simulink Code Inspector, should be employed because review of the generated artifacts can be significantly reduced.

D. Determine whether models represent high-level or low-level requirements

Models can be used to express both high-level requirements (HLR) and low-level requirements (LLR). The representation and flow down of requirements needs to be carefully defined to determine which DO-178 objectives in Annex tables must be satisfied. This is essential to establish clear and concise traceability between HLR, LLR, and generated source and object code. While DO-178C and the attached supplements offer some guidance, aerospace companies are left to choose a requirements definition and implementation strategy. They may, for example, opt to use text or a modeling language to describe requirements. DO-178B, DO-178C and its supplements allow these strategies, given that the appropriate objectives in the Annex tables are satisfied. This section covers the different approaches of using models for each type of requirement, as well as issues that should be taken into consideration when choosing an approach.

According to Position Paper CAST-15 (Certification Authorities Software Team) [15], the HLR generally represent "what" is to be designed and LLR represent "how" to carry out the design. Models used to define HLR should contain the data as listed in section 11.9 of DO-178C. This data focuses on the definition of high-level functionality, as well as high-level interfaces. On the other hand, models used to define LLR should contain data as listed in section 11.10. These data define detailed design aspects, including software architecture, data and control flow, and scheduling.

There are several approaches to partitioning the HLR and LLR between one or two models, and each approach has advantages and disadvantages. Table 2 provides a summary of five potential choices of using the model to define requirements.

Table 2 Choices for Model Definition of Requirements

No.	Approach	Advantages	Disadvantages
1	LLR are defined using models, and HLR are defined textually	• Clear separation of data representing LLR and HLR • Can employ Model-Based Design aspects such as code generation and coverage analysis	• Verification may be more difficult due to having to compare textual requirements to simulation and on-target test results

2	LLR and HLR are defined using separate models	• Clear separation of data representing LLR and HLR • Can employ Model-Based Design aspects such as code generation and coverage analysis • The specification model may be used to verify the functionality of the design model through simulation	• Two models must be maintained and verified • Two separate modeling standards needed • Unlikely that all HLR will be able to be represented in the model
3	LLR are defined textually, and HLR are defined using models	• Clear separation of LLR and HLR • Can employ Model-Based Design aspects such as test coverage analysis	• Verification may be more difficult due to having to compare textual requirements to simulation and on-target test results • Automatic code generation cannot be used • Unlikely that all HLR will be able to be represented in the model
4	A single set of models are used to define LLR and HLR, but not used in the System Requirement and the System Design Process	• Minimal number of modeling artifacts • Can employ Model-Based Design aspects such as code generation and coverage analysis	• System requirements must be sufficiently detailed for tracing to the design model
5	A single set of models are used to define LLR and HLR, and also used in the System Requirement and the System Design Process	• Minimal number of modeling artifacts • Can employ Model-Based Design aspects such as code generation and coverage analysis	• System requirements must be sufficiently detailed for tracing to Design Model

Refer to appendix MB.B.17 and the examples that it contains for a more detailed discussion of how models can be used to represent different levels of requirements, and in different parts of the development life cycle.

When using a design model in software development, it should be determined whether the model will represent the LLR, software architecture, or both. It is typical that the models will include some, but not all, aspects of the software architecture. For example, a model reference hierarchy provides a definition of software architecture, but the model-based code may also be integrated with I/O drivers using a real-time operating system (RTOS). That portion of the RTOS software architecture will be defined independent of the models.

The use of models in the entire system and software development process should be examined to find opportunities for model reuse. The optimal case of reusability would be example 5 in Table 2 (also in table MB.1-1 in DO-331) in which a model is used in the development of system requirements and design. While this approach does not reduce the number of DO-178C objectives to satisfy, it provides an opportunity to develop the design model at an earlier stage in development.

In the case where models represent HLR and LLR, the HLR model must be handled separately from the LLR model. Specifically, development aspects such as modeling standards, test environment, and verification activities must be defined separately for the HLR model and the LLR model. However, keeping the HLR model and LLR model in the same modeling environment will provide opportunities for efficiency gains through tool reuse and verification through simulation of both models in a single environment.

Merging the HLR and LLR into a single model or data item is not recommended [15]. This is because of the different purposes of each level of requirements; HLR state what the system software is to do, while the LLR state how it is done and what components are used to do it. Verification must be performed on the data for both HLR and LLR separately, and merging the two may make this impossible. Note that although Section 5 of DO-178C allows a single level of requirements, it is typically not done because it requires the system level requirements to contain additional detail to allow traceability to the combined HLR/LLR data item.

E. Use simulation and analysis to support DO-178 objectives

Simulation is a verification and validation technique presented in DO-331. With Model-Based Design, simulation can replace traditional reviews and analysis to provide reproducible evidence of compliance with a subset of objectives for high and low-level requirements. Simulation enables engineers to predict the dynamic behavior of a system, which can be more effective than traditional reviews and analysis in the verification and validation of the design.

In the case of a design model that contains low-level requirements, simulation can be used to demonstrate compliance with high-level software requirements (DO-331, section MB.6.3.2.a). Model-Based Design tools, such as Simulink Report Generator, are used to execute the suite of requirements-based test cases using the design models, simulate the dynamic response of the system and automatically capture results in a report (Figure 4) for manual review. Support scripts to compare simulation outputs to expected results should be developed and qualified as verification tools (criterion 2, per DO-330) [8].

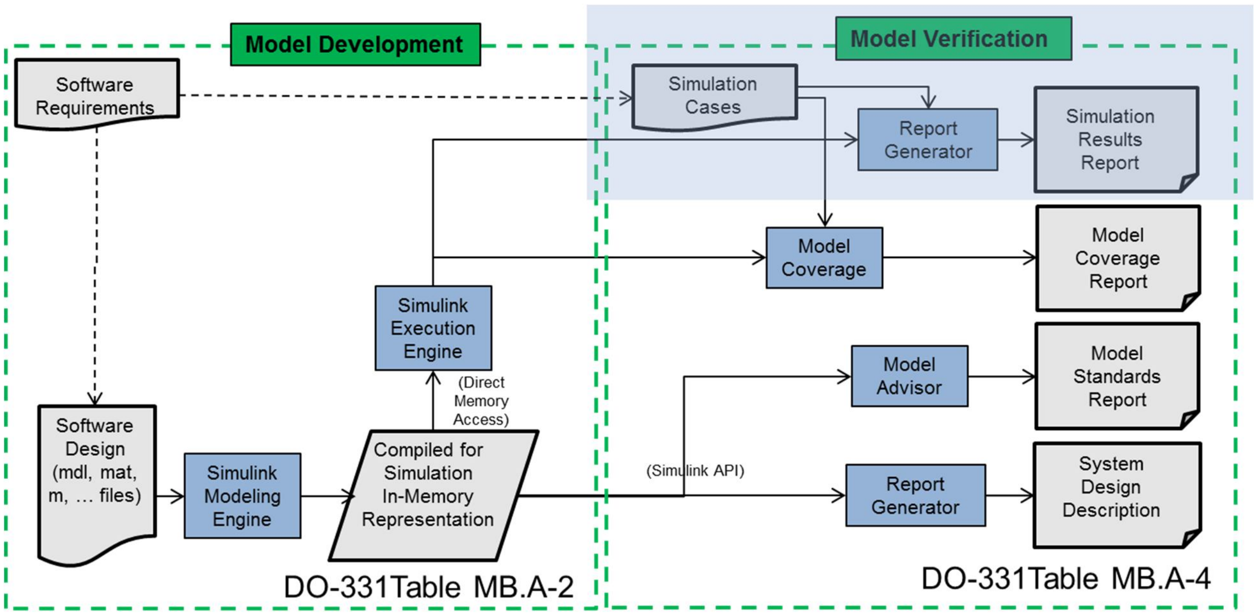

Figure 4 Simulation for Model Verification

Per section 6.4.1 in DO-178C [5], a processor in-the-loop (PIL) testing platform may be used to verify individual software components, even if the testing platform does not include the final target hardware. For each design model, a test harness should be developed and configured to interface with a development board with a processor identical to the target. Each test harness should contain a model block at the top level that references the corresponding design model. Model-Based Design tools, such as Simulink Report Generator, are used to execute the suite of requirements-based test cases on the target and automatically capture results in a report (Figure 5). The report is manually reviewed to verify that the executable object code satisfies the high-level software requirements (DO-331, Table MB.A-6).

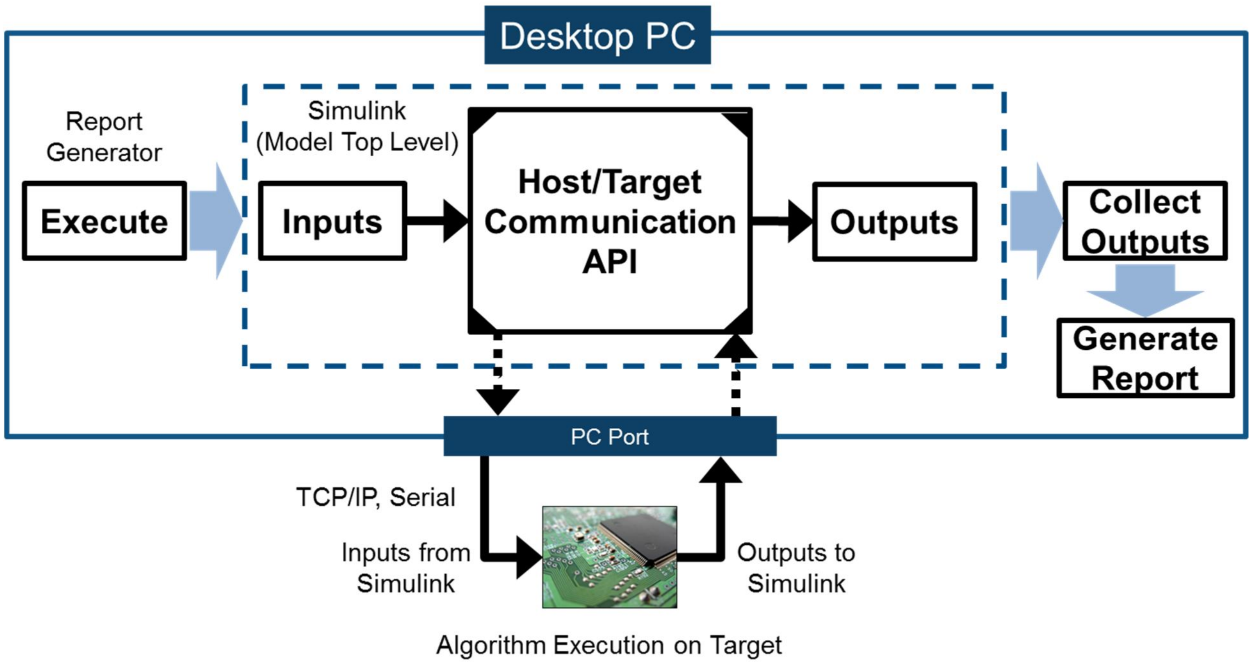

Figure 5 Testing on the Target with Processor-In-The-Loop (PIL)

F. Analyze requirements-based testing at a higher level of abstraction

DO-178C allows the use of simulation to verify functionality in place of, or alongside, manual reviews of design data. In addition, the use of simulation in testing yields other benefits such as model coverage, which provides important feedback on how thoroughly the design was tested.

Without Model-Based Design, software requirements are traditionally defined using textual documents or static graphical specifications such as flowcharts. Such higher-level design data cannot be used for verification on their own. They needed to be first transformed to source code, and then exercised in either a simulator or the target computer. Test cases also must be created from the requirements. With Model-Based Design, this type of testing can be done at a higher level of abstraction, meaning that the models representing the design, the requirements, or both can be used directly in verification activities.

There are several aspects to this best practice. First, process and methods must be established to apply and analyze traces from requirements to the model and from requirements to test cases. Second, a process to develop test cases in the modeling and simulation environment must be established. Third, techniques such as model coverage analysis should be applied to measure the adequacy of the tests developed from requirements and to detect any design defects such as dead logic. Lastly, formal methods tools may be used to automatically create requirements-based tests in certain cases for certain modeling tool chains or to detect design defects such as inter overflow and division by zero.

Requirements traceability is one of the fundamental pillars of DO-178. Establishing clear and concise traceability between high level requirements, models, source code, object code, and test cases can be challenging. A best practice is to assess model coverage using requirements-based test cases, and use this analysis to uncover missing requirements, derived requirements, and other issues during the design phase. This helps prevent wasted effort on generating software with known defects and can reduce costs associated with fixing issues due to missing or incorrect requirements further into the software development process [9]. Many modeling tools offer features to trace external requirements to the model, or to test cases used in the simulation environment. Those features often allow traceability analysis in addition to establishing traces.

Just as with the development of software architecture and design, test cases and procedures must be developed from the HLR and LLR. These test cases and procedures can be developed within the modeling environment and used in simulation in support of formal verification activities. DO-331, section MB.6.8.1 outlines activities that should be completed in order to do this. Many of these activities involve determining if simulation is appropriate for the verification of the requirements and if simulation completely satisfies the specific review or analysis objectives. When using simulation for verification activities it is recommended that model coverage be used to determine which requirements expressed by the model were not exercised by verification based on the requirements from which the model was developed [6]. Note that model coverage is separate from structural coverage per DO-178C section 6.4.4.2.

Formal methods tools such as Simulink Design Verifier can be used to identify "dead" logic in a model, and model coverage data can be used to create test cases for any requirements that have not been tested. Although certification credit cannot be attained by using these tools at the model level, identifying such design issues before the generation of source code reduces downstream effort.

G. Employ Model-Based Design tools that can be qualified

DO-178C allows the use of tools to eliminate, reduce, or automate DO-178C processes. Organizations can reduce or eliminate code reviews, for example, by using a tool that automates the verification of DO-178C objectives for source code against low-level requirements. These tools must be qualified as stated in RTCA DO-330 "Software Tool Qualification Considerations" [9]. A best practice is to use and qualify as many Model-Based Design verification tools as possible throughout the software development life cycle. Development tool qualification is generally not performed (for example, there are no commercially available qualified compilers) because it offers limited ROI versus an automated approach that verifies the output of a development tool using a qualified verification tool.

Tool qualification kits provide artifacts and guidance to streamline the tool qualification process. The qualification kits can be extended to meet specific company standards and requirements. To reduce effort, however, extension of Model-Based Design tools and their respective qualification kits should be minimized. In addition, tool qualification artifacts should be generated within the same software development environment used for the primary application. Traditionally, verifying code against requirements is a time-consuming and error-prone process that requires manually reviewing code line-by-line against a project checklist to satisfy source verification objectives in table A-5 in DO-178C. The automated code inspection Simulink Code Inspector provides can significantly reduce the time required for satisfying these objectives. The design models should be developed using the subset of blocks supported by Simulink Code Inspector (Figure 6).

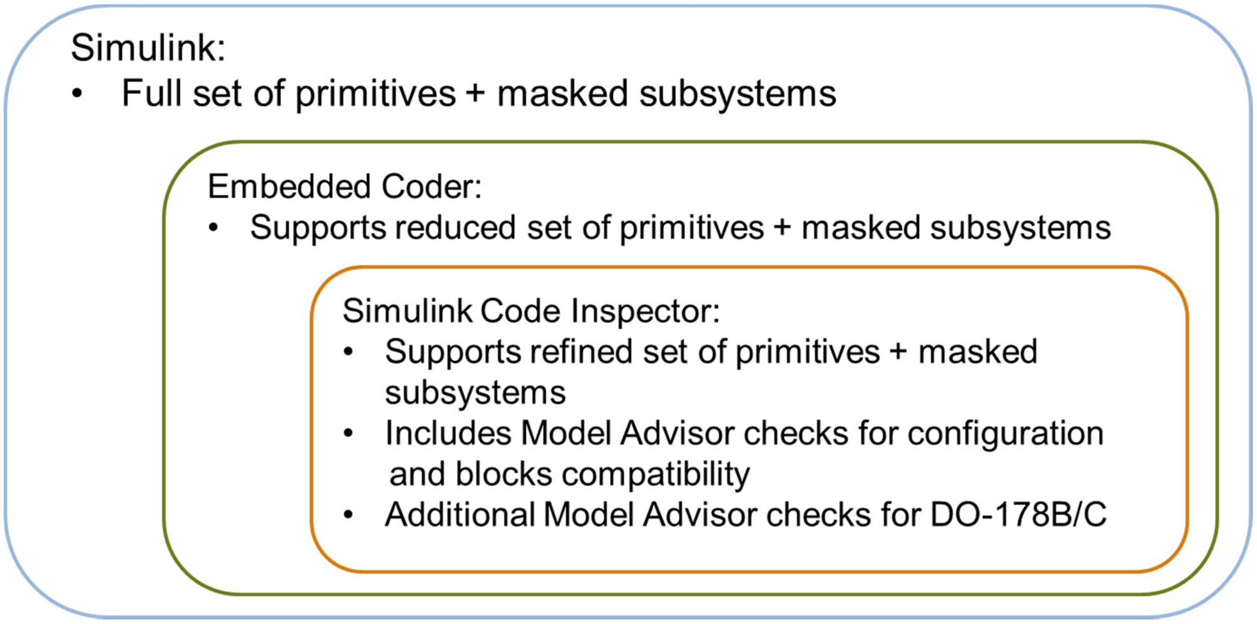
Simulink:
• Full set of primitives + masked subsystems
Embedded Coder:
• Supports reduced set of primitives + masked subsystems
Simulink Code Inspector:
• Supports refined set of primitives + masked subsystems
• Includes Model Advisor checks for configuration and blocks compatibility
• Additional Model Advisor checks for DO-178B/C

Figure 6 Simulink Code Inspector Supports a Subset of Available Blocks

Traditionally, manually derived artifacts are developed and reviewed to satisfy objectives in tables A-3 and A-4 in DO-178C [5]. Review checklists are developed and used for activities such as verification of conformance to standards and traceability to requirements. With Model-Based Design, the objectives in tables MB.A-3 and MB.A-4 in DO-331 [6] can be satisfied by a combination of the following:

- Review of a detailed report describing the system design
- Review and simulation of the design model
- Model Advisor checks

Simulink Report Generator should be used to automatically generate a System Design Description report (SDD)—a report describing the system design in detail—from the design models. The SDD generation process can be customized to include or remove specific content. The final report, however, should contain key details that characterize the system design, including:

- Root-level and subsystem interfaces
- Links to high-level requirements
- Design variables
- Block parameters
- Design model configuration parameters
- External dependencies

The SDD capability of the Simulink Report Generator tool must be qualified in order to claim credit for satisfying the objectives in tables MB.A-3 and MB.A-4 in DO-331 [6]. The qualification kit should be used to streamline the tool qualification process.

Model-Based Design tools, such as Simulink Report Generator, should also be used to execute requirements-based test cases using the design models. Though review of the numerical outputs from simulation and PIL against expected results can be performed manually, the best practice is to develop and qualify custom scripts that perform the comparison. The scripts should be able to support multiple data types (single, double, integers) as well as implement best practices for comparing floating point numbers.

II. Conclusion

The DO-178B document provides little guidance on the use of models. DO-331, a supplement to the newly released DO-178C, provides additional guidance on artifact and verification evidence when using models. The documents focus on process and verification objectives leaving aerospace companies to determine how to most efficiently apply Model-Based Design across the full software development life cycle. Distilled from knowledge gained through years of collaboration with customers in the aerospace industry in developing high-integrity software, the best practices presented in this paper can be adopted to implement an efficient workflow using Model-Based Design to develop software certified to DO-178.

References

1. "Design Times at Honeywell Cut by 60 Percent," W. King, Honeywell Commercial Aviation Systems, Nov. 1999, http://www.mathworks.com/company/user_stories
2. "Flight Control Law Development for F-35 Joint Strike Fighter," D. Nixon, Lockheed-Martin Aeronautics, Oct. 2004, http://www.mathworks.com/programs/techkits/pcg_tech_kits.html
3. "ESA's First-Ever Lunar Mission Satellite Orbits Moon with Automatic Code," P. Bodin, Swedish Space, Oct. 2005, http://www.mathworks.com/programs/techkits/pcg_tech_kits.html

4. “Automatic Code Generation at Northrop Grumman,” R. Miller, Northrop Grumman Corporation, June 2007, http://www.mathworks.com/programs/techkits/pcg_tech_kits.html
5. “Software Considerations in Airborne Systems and Equipment Certification,” RTCA/DO-178C, RTCA Inc. Dec. 2011
6. “Model-Based Development and Verification Supplement to DO-178C and DO-278A,” RTCA/DO-331, RTCA Inc. Dec. 2013
7. “Software Considerations in Airborne Systems and Equipment Certification,” RTCA/DO-178B, RTCA Inc. Dec. 1992
8. “Software Tool Qualification Considerations,” RTCA/DO-330, RTCA Inc. Dec. 2011
9. J.B. Dabney, "Return on Investment of Independent Verification and Validation Study Preliminary Phase 2B Report." Fairmont, W.V.: NASA IV&V Facility, 2003
10. “Model-Based Design for DO-178B with Qualified Tools,” Tom Erkkinen, Bill Potter, The MathWorks, Inc., AIAA Modeling and Simulation Technologies Conference and Exhibit 2009, AIAA Paper 2009-6233
11. “Complying with DO-178C and DO-331 using Model-Based Design,” Bill Potter, The MathWorks, Inc., SAE Paper 12AEAS-0090
12. “Best Practices for Establishing a Model-Based Design Culture,” Paul F. Smith, Sameer M. Prabhu, Jonathan H. Friedman, The MathWorks, Inc., SAE Paper 2007-01-0777
13. “Pragmatic Strategies for Adopting Model-Based Design for Embedded Applications,” Eric Dillaber, Larry Kendrick, Wensi Jin and Vinod Reddy, The MathWorks, Inc., SAE Paper 2010-01-0935
14. “Large Scale Modeling for Embedded Applications,” Kerry Grand, Vinod Reddy, Gen Sasaki, Eric Dillaber, The MathWorks, Inc., SAE Paper 2010-01-0938
15. “Merging High-Level and Low-Level Requirements,” Certification Authorities Software Team (CAST), Position Paper CAST-15 Feb. 2003