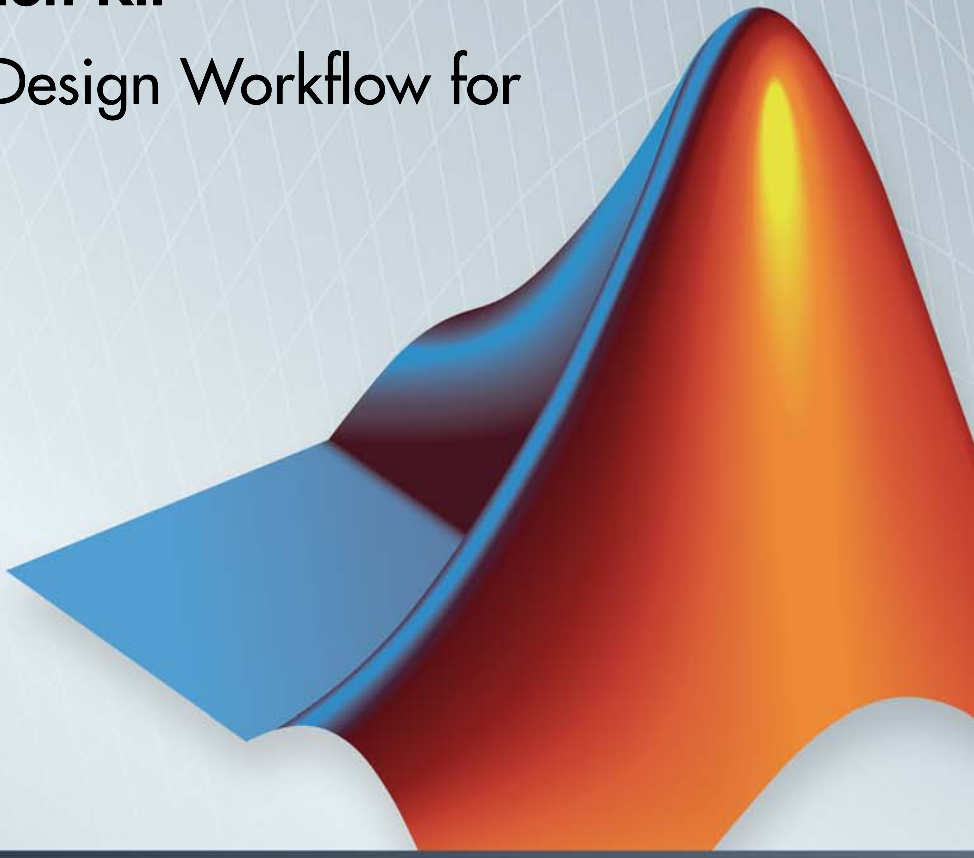
DO Qualification Kit

Model-Based Design Workflow for DO-178C

R2013a



MATLAB® & SIMULINK®

MathWorks®

Revision History

September 2010	Online only	New for Version 1.3 (Release 2010b)
April 2011	Online only	Revised for Version 1.4 (Release 2011a)
September 2011	Online only	Revised for Version 1.5 (Release 2011b)
March 2012	Online only	Revised for Version 1.6 (Release 2012a)
September 2012	Online only	Revised for Version 2.0 (Release 2012b)
March 2013	Online only	Revised for Version 2.1 (Release 2013a)

Contents

Tool Description

Overview of the Tools

The purpose of this section is to describe the high level architecture of the development and verification tools used in the DO-178C workflow with Model-Based Design. This section also describes the independence aspects of the various tools and how errors in the tools can be detected. There are two types of tools used in the workflow, development tools and verification tools.

Development tools are:

- Simulink®
- Stateflow®
- MATLAB® Coder™
- Simulink Coder
- Embedded Coder®

Verification tools are:

- MATLAB Report Generator™
- Simulink Report Generator
- Simulink Design Verifier™
- Simulink Code Inspector™
- Simulink Verification and Validation™ - Model Advisor
- Simulink Verification and Validation - Model Coverage
- SystemTest™
- Polyspace® Client™ for C/C++; Polyspace Server™ for C/C++

Independence of the Tools

Simulink and Stateflow are separate tools used for the development of models. Simulink may be used without Stateflow, but when Stateflow is used, Simulink is also required. Simulink and Stateflow are tightly integrated and are not independent of each other. There is not a requirement for Simulink and Stateflow to be independent since they are both used together as part of the development of the software design. The Simulink API, which is referenced throughout this document, provides an interface for other tools that cannot access the in memory data directly, to get the data from the model by using this interface. For example, a user can get data from a model using the get_param command in MATLAB or set a parameter in the model using the set_param command in MATLAB.

See the workflow section of this document, "Software Development Process" on page 2-10, which includes the following objectives for the use of Simulink and Stateflow:

- Software High-Level Requirements are Developed
- Derived Software High-Level Requirements are Developed
- Software Architecture is Developed
- Software Low-Level Requirements are Developed
- Derived Software Low-Level Requirements are Developed

MATLAB Coder, Simulink Coder and Embedded Coder are separate tools used for the development of source code. MATLAB Coder is a prerequisite for Simulink Coder and Embedded Coder. Simulink Coder is required when generating code from Simulink and Stateflow models. These three tools are tightly integrated and are not independent of each other. There is not a requirement for MATLAB Coder, Simulink Coder and Embedded Coder to be independent since they are used together as part of the development of the source code. In the following sections of this document, references to Embedded Coder are intended to include Simulink Coder and MATLAB Coder as the entire code generation tool set.

See the workflow section of this document, "Software Development Process" on page 2-10, which includes the following objectives for the use of MATLAB Coder, Simulink Coder and Embedded Coder:

- Source Code is Developed

The MATLAB and Simulink Report Generators are two separate tools, with the MATLAB Report Generator being a prerequisite for the Simulink Report Generator. The Simulink Report Generator provides components for reporting on Simulink and Stateflow models and is integrated with the MATLAB Report Generator. These components interrogate the model using the Simulink API to read data from the model loaded in memory. The report generator components used to generate the System Design Description document can only read data from the model, they do not have the capability to write or modify data in the model. The System Design Description includes requirements traceability links that may be inserted into the models using the Requirements Management Interface that is part of Simulink Verification and Validation.

See the workflow sections of this document, "Verification of Requirements Process" on page 2-17 and "Verification of Design Process" on page 2-24, which includes the following objectives for the use of MATLAB Report Generator and Simulink Report Generator:

- Verification of Requirements Process
  - Software High-Level Requirements Comply with System Requirements
  - High-Level Requirements are Accurate and Consistent
  - High-Level Requirements are Compatible with Target Computer
  - High-Level Requirements are Verifiable
  - High-Level Requirements Conform to Standards
  - High-Level Requirements are Traceable to System Requirements
  - Algorithms are Accurate
- Verification of Design Process
  - Low-Level Requirements Comply with High-Level Requirements
  - Low-Level Requirements are Accurate and Consistent
  - Low-Level Requirements are Compatible with Target Computer
  - Low-Level Requirements are Verifiable

- Low-Level Requirements Conform to Standards
- Low-Level Requirements are Traceable to System Requirements
- Algorithms are Accurate
- Software Architecture is Compatible with High-Level Requirements
- Software Architecture is Consistent
- Software Architecture is Compatible with Target Computer
- Software Architecture is Verifiable
- Software Architecture is Conforms to Standards

Simulink Design Verifier is a separate tool with three capabilities; Design Error Detection, Property Proving and Test Case Generation. Simulink Design Verifier contains formal analysis engines that operate on an internal representation derived from but in a different form than the Simulink model loaded in memory. Design Error Detection can find specific design errors in the model, such as divide-by-zero or numeric overflows, using formal methods. Property Proving, which also uses formal methods, can prove properties that are defined by the user in conjunction with assumptions that are also defined by the user. The formal analysis engines are separate and independent of Simulink and Stateflow, and do not involve simulation of the model. Simulink Design Verifier can automatically generate test cases based on the model that can be used to verify the executable object code complies with the model. The basis for the test cases can be a combination of user defined constraints, model coverage criteria for blocks in the model and user defined test objectives. The constraint blocks, model coverage criteria and test objective blocks are ignored by Embedded Coder and are therefore independent of the coding process. In order to verify the code using the generated test cases, the test cases must be run on the model in order to produce expected results for the code. The completeness of those test cases may be assessed using the model coverage tool and the expected results may be assessed via review of the results from simulation.

See the workflow sections of this document, “Verification of Requirements Process” on page 2-17, “Verification of Design Process” on page 2-24 and “Testing of Outputs of Integration Process” on page 2-40, which includes the following objectives for the use of Simulink Design Verifier:

- Verification of Requirements Process
  - Software High-Level Requirements Comply with System Requirements
  - High-Level Requirements are Verifiable
  - Algorithms are Accurate
- Verification of Design Process
  - Low-Level Requirements Comply with High-Level Requirements
  - Low-Level Requirements are Verifiable
  - Algorithms are Accurate
- Testing of Outputs of Integration Process
  - Executable Object Code Complies with Low-Level Requirements
  - Executable Object Code is Robust with Low-Level Requirements

Simulink Code Inspector is a separate tool that can be used to verify source code developed from Embedded Coder. This tool is implemented independent of Simulink, Stateflow and Embedded Coder. This tool interrogates the model using the Simulink API to read data from the model loaded in memory. All of the API commands used can only read data from the model, they do not have the capability to write or modify data in the model. The model is converted into a different intermediate representation for use in the code inspection process. The Simulink Code Inspector also uses the generated C code files as input and parses these into a different intermediate representation that can be compared to the model's intermediate representation. The requirements, design and source code for Simulink Code Inspector are developed separately and are independent of MATLAB Coder, Simulink Coder and Embedded Coder implementations.

See the workflow section of this document, "Verification of Coding and Integration Process" on page 2-35, which includes the following objectives for the use of Simulink Code Inspector:

- Source Code Complies with Low-Level Requirements
- Source Code Complies with Software Architecture
- Source Code is Verifiable

- Source Code is Traceable to Low-Level Requirements
- Source Code is Accurate and Consistent

The Model Advisor checks are provided in several different products; Simulink, Embedded Coder, Simulink Code Inspector, Simulink Verification and Validation and Simulink Control Design™. The basic core implementation of Model Advisor checks is done via an engine that uses MATLAB functions and is independent of Simulink, Stateflow and Embedded Coder. The Model Advisor uses the Simulink API to read data from the model loaded in memory. The Model Advisor does have the capability to automatically fix issues detected by checks, but the fixes must be initiated by the user and the model would have to be resaved. Then the checks can be re-run by the user in order to verify the fixes. For custom checks created by the user, it is the user's responsibility to not allow those checks to modify the model.

See the workflow sections of this document, "Verification of Requirements Process" on page 2-17 and "Verification of Design Process" on page 2-24, which includes the following objectives for the use of Model Advisor:

- Verification of Requirements Process
  - High-Level Requirements are Accurate and Consistent
  - High-Level Requirements are Compatible with Target Computer
  - High-Level Requirements Conform to Standards
  - High-Level Requirements are Traceable to System Requirements
  - Algorithms are Accurate
- Verification of Design Process
  - Low-Level Requirements are Accurate and Consistent
  - Low-Level Requirements are Compatible with Target Computer
  - Low-Level Requirements Conform to Standards
  - Low-Level Requirements are Traceable to System Requirements
  - Algorithms are Accurate
  - Software Architecture is Consistent
  - Software Architecture is Compatible with Target Computer

  - Software Architecture is Conforms to Standards

The Model Coverage capability is provided as part of Simulink Verification and Validation. Model Coverage instruments the model loaded into memory prior to simulation and evaluates the coverage criteria as the simulation progresses. Model Coverage also has the capability to merge multiple simulation runs into a combined coverage report. The user can run simulations with coverage enabled and disabled to insure there has been no effect on behavior of the model due to the instrumentation.

See the workflow sections of this document, "Verification of Requirements Process" on page 2-17 and "Verification of Design Process" on page 2-24, which includes the following objectives for the use of Model Coverage:

- Verification of Requirements Process
  - Software High-Level Requirements Comply with System Requirements
  - High-Level Requirements are Verifiable
- Verification of Design Process
  - Low-Level Requirements Comply with High-Level Requirements
  - Low-Level Requirements are Verifiable

SystemTest is a separate tool that can be used to execute simulations in a batch model and check actual results against expected results. It also provides the capability to author test cases manually or to import test cases in other formats, such as Excel® spreadsheets. Because the test cases and expected results are developed manually by the user, they are independent of the model and source code. The Limit Check element within SystemTest that is used to determine Pass/Fail of the model or code under test is implemented completely independent of Simulink, Stateflow and Embedded Coder.

See the workflow sections of this document, "Verification of Requirements Process" on page 2-17 and "Verification of Design Process" on page 2-24, which includes the following objectives for the use of SystemTest:

- Verification of Requirements Process
  - Software High-Level Requirements Comply with System Requirements

  - High-Level Requirements are Accurate and Consistent
  - High-Level Requirements are Verifiable
  - Algorithms are Accurate
- Verification of Design Process
  - Low-Level Requirements Comply with High-Level Requirements
  - Low-Level Requirements are Accurate and Consistent
  - Low-Level Requirements are Verifiable
  - Algorithms are Accurate
  - Software Architecture is Compatible with High-Level Requirements
  - Software Architecture is Consistent
  - Software Architecture is Verifiable

Polyspace is a separate tool that has two capabilities; coding standards checking (example MISRA C®) and run time error detection. The main input to Polyspace is the source code; however it can optionally read range specification data from the model using the Simulink API. When using the Polyspace Model Link™ SL product, it can trace defects found in the source code back to the source blocks in the model. Polyspace is completely independent of MATLAB Coder, Simulink Coder and Embedded Coder. The requirements, design and source code for Polyspace are developed separately and are independent of MATLAB Coder, Simulink Coder and Embedded Coder implementations. Polyspace also supports C code, whether it is automatically generated or manually developed. For run-time error detection, Polyspace uses Abstract Interpretation in its formal methods engine.

See the workflow section of this document, "Verification of Coding and Integration Process" on page 2-35, which includes the following objectives for the use of Polyspace:

- Source Code is Verifiable
- Source Code Conforms to Standards
- Source Code is Accurate and Consistent

Model and Source Code Development and Verification

In a workflow where code is generated from the Simulink and Stateflow models, the models are considered to be the low-level software requirements and architecture as defined in DO-178C. The actual low-level requirements are the compiled model in memory as interpreted by the Simulink engine based on input from the model file, as well as any data files, such as MATLAB or MAT files that load data into the MATLAB or model workspaces. See Figure 1: Model and Source Code Development and Verification on page 1-12. The model file itself does not represent the low-level requirements, because the model semantics are not fully included in that file. The model semantics are not complete until the model file has been loaded into memory and the Simulink engine has compiled the model. Some of the model semantics that are determined at compile time, but are not included in the model file, for the model consists of:

- Propagated Sample Times
- Propagated Data Types
- Propagated Signal Dimensions
- Propagated Signal Types
- Block Execution Order

The System Design Description, which is created using the Simulink Report Generator, provides a document that details the compiled for simulation in memory representation of the model. This provides documentation of the low-level software requirements, as defined in the DO-178C glossary:

Low-level requirements – Software requirements developed from high-level requirements, derived requirements, and design constraints from which Source Code can be directly implemented without further information.[1]

Compile for simulation and compile for code generation are two different compiles and result in two slightly different in-memory representations. SystemTest, Model Coverage, Simulink Code Inspector, Model Advisor and Report Generator only compile for simulation. Embedded Coder compiles for

1. "Software Considerations in Airborne Systems and Equipment Certification," Document No. RTCA DO-178C, December 13, 2011, Prepared by SC-205

code generation, which includes the entire compile for simulation information plus the following additional information.

- Model optimizations that are applied only for code generation
- Consistency checking for storage classes in the generated code

Since the model and code verification activities may take place at different times or on different computers, it is necessary to check the consistency of the in-memory representations of the model. An MD5 Checksum computation is used to check this consistency. The MD5 checksum is computed based on the in-memory representation and includes any data that has been loaded into the workspace from external files that are used by the model. The MD5 Checksum value is automatically inserted into the Model Advisor report, the System Design Description and the Simulink Code Inspector report. It is also possible to use the Simulink API to access the MD5 Checksum and insert it into a SystemTest report or for use in other reports that may be generated during simulations using other methods such as Report Generator or MATLAB scripts. A model version number and last saved date are also available in the reports, and this data is automatically updated each time that a model is saved. The model version number and last saved dates are not affected by externally loaded data, so that is why the MD5 Checksum is required to verify complete consistency of the in-memory representation. The System Design Description does document the workspace variables that are used by the model at the time the report is generated.

Note The model checksum computation is platform dependent.

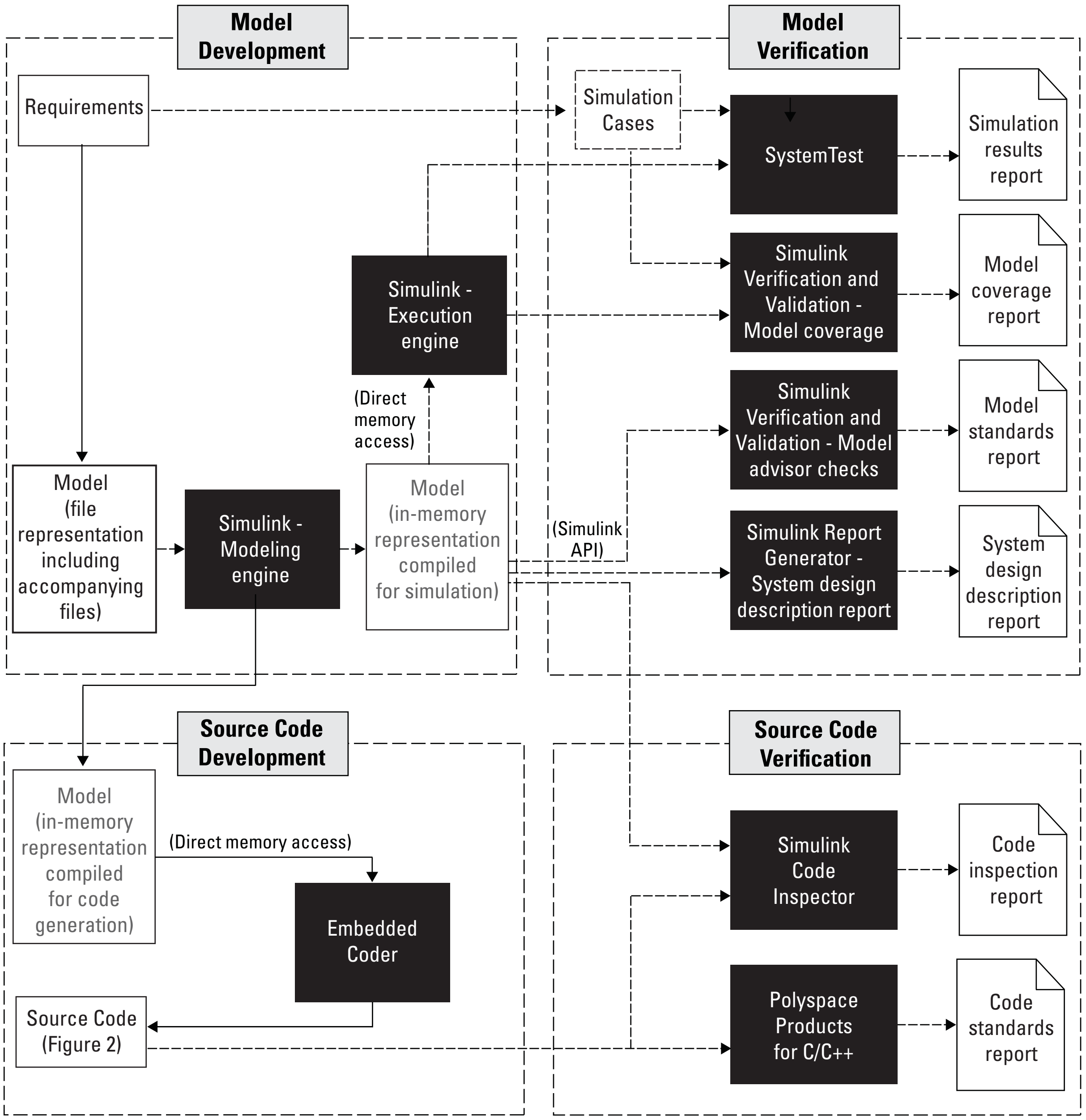

Model Development
Requirements
Model (file representation including accompanying files)
Simulink - Modeling engine
Model (in-memory representation compiled for simulation)
(Direct memory access)
Simulink - Execution engine
Model Verification
Simulation Cases
SystemTest
Simulation results report
Simulink Verification and Validation - Model coverage
Model coverage report
Simulink Verification and Validation - Model advisor checks
Model standards report
(Simulink API)
Simulink Report Generator - System design description report
System design description report
Source Code Development
Model (in-memory representation compiled for code generation)
(Direct memory access)
Embedded Coder
Source Code (Figure 2)
Source Code Verification
Simulink Code Inspector
Code inspection report
Polyspace Products for C/C++
Code standards report

Figure 1: Model and Source Code Development and Verification

Potential Tool Errors and Detection

The following table provides information regarding potential user and tool errors, the effects of those errors and how the errors are detected.

Error Source	Error Effect	Detected By	Mitigating Factors
User Input (model, MATLAB, or MAT file data)	Failure to comply with requirements	Simulation Cases and review of System Design Description	SystemTest and Report Generator Qualification
	Failure to conform to standards	Model Advisor and review of System design Description	Model Advisor and Report Generator Qualification
	Unintended function	Review of System Design Description, Simulation Cases and Model Coverage	SystemTest and Model Coverage Qualification
Simulink Engine	Failure to comply with requirements	Simulation Cases and review of System Design Description	SystemTest and Report Generator Qualification
	Failure to conform to standards	Model Advisor and review of System design Description	Model Advisor and Report Generator Qualification
	Unintended function	Review of System Design Description, Simulation Cases and Model Coverage	SystemTest and Model Coverage Qualification

Error Source	Error Effect	Detected By	Mitigating Factors
Simulink Execution Engine	Failure to comply with requirements	Simulation Cases	SystemTest Qualification
	Unintended function	Simulation Cases and Model Coverage	SystemTest and Model Coverage Qualification
Simulink API	Incorrect input to Model Advisor resulting in reported failure	Review of Model Advisor Report and resolution activity	Model Advisor Qualification
	Incorrect input to System Design Description	Review of System Design Description and resolution activity	Report Generator Qualification
	Incorrect input to Simulink Code Inspector resulting in reported failure	Review of Simulink Code Inspector Report and resolution activity	Simulink Code Inspector Qualification
SystemTest	Incorrect expected results evaluation resulting in reported failure	Review of simulation results report and resolution activity	SystemTest Qualification
Model Coverage	Incorrect model coverage reporting	Review of coverage report and additional requirement for code coverage assessment	Model Coverage Qualification

Error Source	Error Effect	Detected By	Mitigating Factors
Model Advisor	Incorrect model standards reporting	Model standards violation resulting in a corresponding code standards violation is detectable by Polyspace	Model Advisor and Polyspace Qualification
Report Generator	Incorrect System Design Description	Review of System Design Description and resolution activity	Report Generator Qualification
Embedded Coder	Incorrect source code	Review of Simulink Code Inspector Report	Simulink Code Inspector Qualification
Simulink Code Inspector	Incorrect reported failure of the source code	Review of Simulink Code Inspector Report and resolution activity	Simulink Code Inspector Qualification
Polyspace	Incorrect reported failure of the source code	Review of Polyspace Report and resolution activity	Polyspace Qualification

The only errors that can directly affect both the model and the source code are user input errors or Simulink Engine errors. In either of these cases the result is incorrect low-level software requirements. The incorrect low-level software requirements are detectable at the model level via a combination of design reviews, simulation, model coverage assessment and conformance to standards checking. Because these activities are being done on the compiled in memory model, the detection is effective whether the error is based on user input or the Simulink Engine. Additionally, if the software level is A or B, the simulation cases used to verify behavior, must be developed by a person other than the model developer in order to achieve independence requirements.

Once the model has been verified, the source code can be generated by Embedded Coder and verified by Simulink Code Inspector and Polyspace. The three tools are developed by independent groups with MathWorks and have independent requirements and code. The one exception is that Simulink Code Inspector and Polyspace do share a common parser function for the C code, but Embedded Coder does not contain this functionality. The Simulink Code Inspector uses the Simulink API as input source for the model information. This is the same API used by Model Advisor and the Simulink Report Generator. This API is verified during the tool qualification testing process for each of these tools. The Simulink Code Inspector input from the model is based on the compiled for simulation in memory representation and does not have access to the compiled for code generation additional information. Both Simulink Code Inspector and Polyspace read the code and header files that are output from Embedded Coder directly as ASCII text files.

The following model verification tools may be qualified, per DO-178C guidelines, using the DO Qualification Kit:

- Simulink Report Generator – System Design Description (SDD) Report
- Simulink Verification and Validation – Model Advisor DO-178C/DO-331 Checks
- Simulink Verification and Validation – Model Coverage
- SystemTest – Limit Check Element

Additionally, the following code verification tools may be qualified, per DO-178C guidelines, using the DO Qualification Kit:

- Simulink Code Inspector – Code inspection report
- Polyspace Client for C/C++; Polyspace Server for C/C++

To summarize, all tool errors in the workflow are detectable by one or more verification activities. Additionally, the tool qualification process for the verification tools provides a level of confidence in the tools that is equivalent to manual verification activities that are automated by the tools.

Object Code Development and Verification

Figure 2: Executable Object Code Development and Verification on page 1-18 shows the Executable Object Code development and verification activities, including the use of Processor In-The-Loop (PIL) mode and target integration testing. These activities are downstream of the model and source code development and verification activities. The compiler is a third party tool that is not provided by MathWorks and therefore is independent. Errors injected by the compiler are detectable by the testing process. The code coverage tool is also provided by a third party, rather than MathWorks®, and this tool is normally qualified.

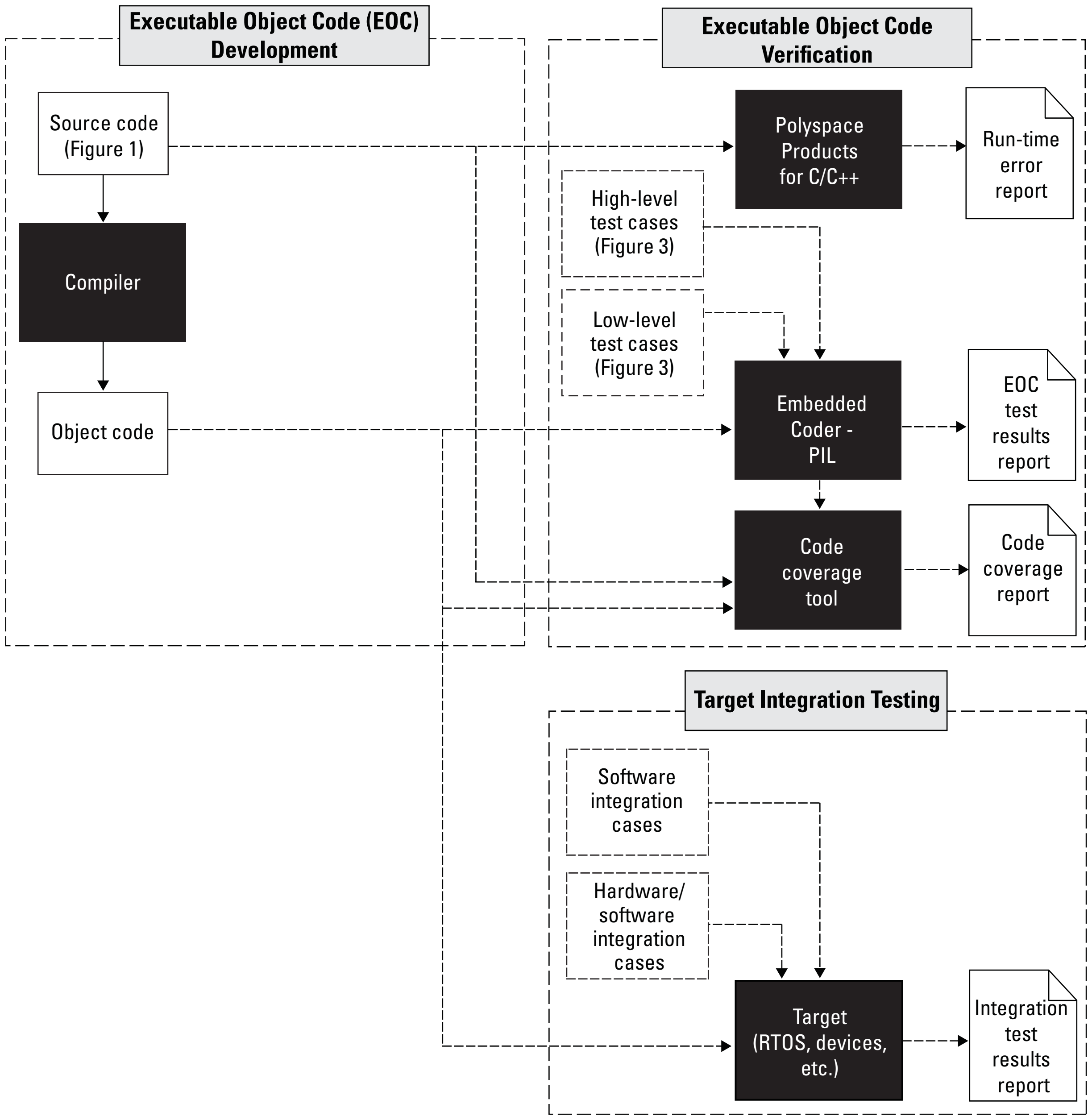

Executable Object Code (EOC) Development
Source code (Figure 1)
Compiler
Object code
Executable Object Code Verification
Polyspace Products for C/C++
Run-time error report
High-level test cases (Figure 3)
Low-level test cases (Figure 3)
Embedded Coder - PIL
EOC test results report
Code coverage tool
Code coverage report
Target Integration Testing
Software integration cases
Hardware/ software integration cases
Target (RTOS, devices, etc.)
Integration test results report

Figure 2: Executable Object Code Development and Verification

Test Case Development

The DO-178C standard calls out three types of testing, all of which are based on the software requirements:

- Hardware/Software Integration Tests
- Software Integration Tests
- Low-Level Tests

Additionally, for DO-178C, test cases should include:

- Normal range test cases
- Robustness test cases

For the executable object code developed from models, the high level test cases and expected results can be the same as the simulation cases and expected results (see Figure 3: Test Case Development on page 1-20). These are developed from the high level requirements document and are completely independent of Simulink, Embedded Coder and the compiler used for the project. The test cases and expected result should also include robustness cases. These test cases can be executed using processor in-the-loop (PIL) capability in conjunction with the Simulink environment used as a test harness, or on a completely separate software test harness.

The low-level test cases and expected results are based on the models, which represent the low-level requirements. Simulink Design Verifier may be used to develop these test cases (see Figure 3). Simulink Design Verifier uses the model as its primary input and also has the capability to input model coverage data. DO-178C calls out that if it can be shown that high level tests cover low-level requirements, then those low-level requirements do not need to be covered by specific low level tests. Model coverage can be used as evidence that high level tests cover low-level requirements, in particular for logical decisions within the models, but also for lookup table data and signal range data within the models. Simulink Design Verifier can then be used to generate tests for the remaining low-level requirements that are not covered by high level testing, for example derived requirements within the model. The user can also insert signal constraints and user defined test objectives within the models or in model test harnesses to complete the testing. The use of test objectives on the inputs to a model to insert test data beyond normal ranges is a good way to verify robustness, for example.

The Hardware/Software Integration cases and the Software Integration cases (see Figure 3) are typically developed manually based on the high-level software requirements. These test cases are executed on the final target in an environment independent of the modeling environment. The final target would include an RTOS or scheduler and the device drivers that interface to the target hardware.

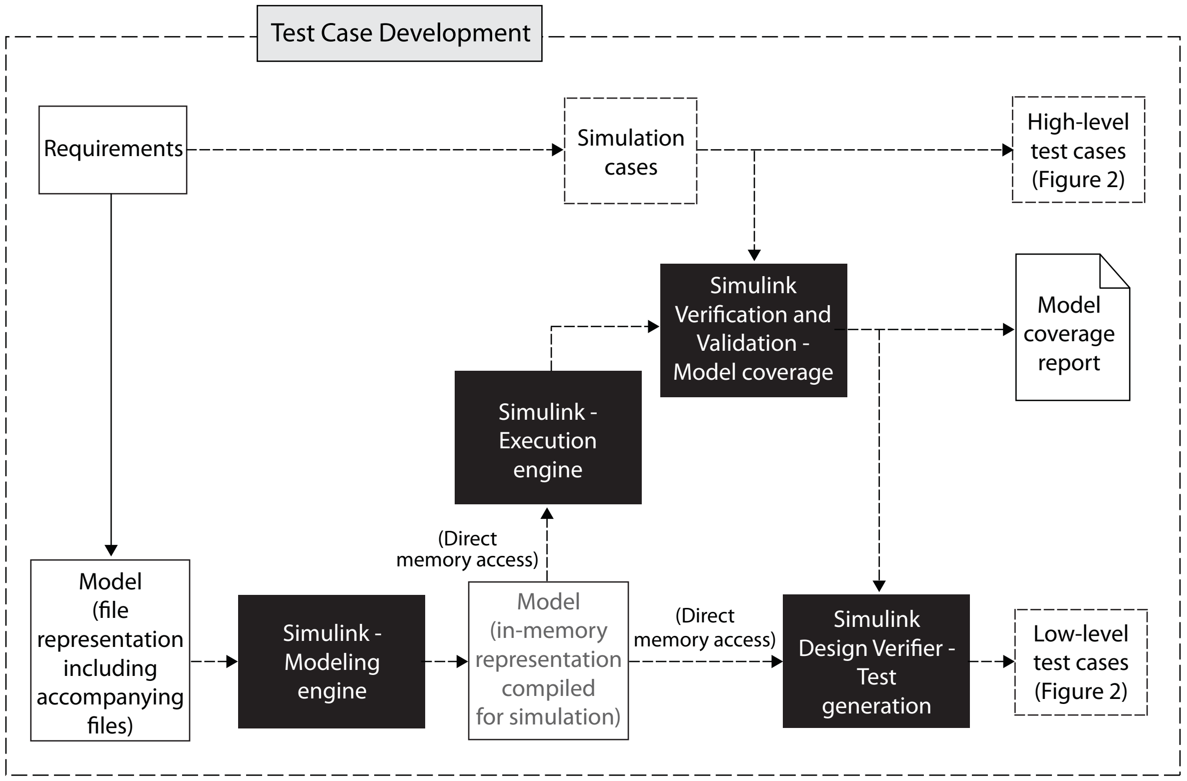

Test Case Development
Requirements
Simulation cases
High-level test cases (Figure 2)
Simulink Verification and Validation - Model coverage
Model coverage report
Simulink - Execution engine
(Direct memory access)
Model (file representation including accompanying files)
Simulink - Modeling engine
Model (in-memory representation compiled for simulation)
(Direct memory access)
Simulink Design Verifier - Test generation
Low-level test cases (Figure 2)

Figure 3: Test Case Development

DO-178C Software Life Cycle

DO-178C Software Life Cycle Overview

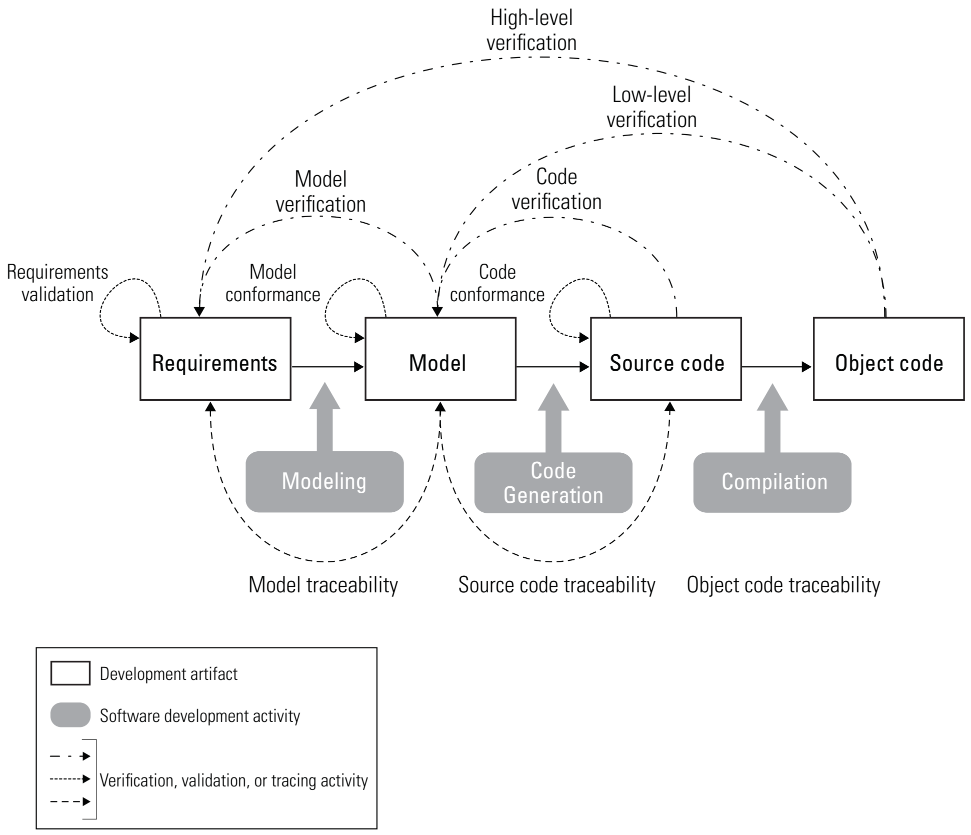
The DO-178C software life cycle consists of the following processes:

- Planning
- Software development
- Verification of requirements
- Verification of design
- Verification of coding and integration
- Testing of outputs of integration
- Verification of verification results
- Software configuration management
- Software quality assurance
- Certification liaison process

There are objectives that must be met for each of the life cycle stages in DO-178C. In Annex A of DO-178C, these objectives are summarized in tables. This document summarizes those tables and provides recommendations on meeting the objectives using a Model-Based Design process. Available Model-Based Design tools that can be used in achieving the objectives are also included.

Model-Based Design Workflow in DO-178C

The following diagram shows a Model-Based Design workflow that addresses the development and verification activities in a DO-178C software life cycle.



The following table lists the MathWorks products and capabilities that can be used in each activity of the workflow as Model-Based Design tools.

Workflow Activity	Available Products and Capabilities for Model-Based Design
Requirements validation	Manual review
Modeling	Simulink, Stateflow

Workflow Activity	Available Products and Capabilities for Model-Based Design
Model traceability	Simulink Verification and Validation — Requirements Management Interface (RMI), Simulink Report Generator — System Design Description report*
Model conformance	Simulink Verification and Validation — DO-178C/DO-331 checks*
Model verification	SystemTest — Limit Check element*, Simulink Design Verifier — Property Proving (optional), Simulink Design Verifier — Design Error Detection (optional), Simulink Verification and Validation — Model Coverage*, Simulink Report Generator — System Design Description report*
Code generation	Embedded Coder
Source code traceability	Simulink Code Inspector — Traceability Report*
Code conformance	Polyspace Products for C/C++ — MISRA AC AGC checks*
Code verification	Simulink Code Inspector — Code Verification Report*, Polyspace Products for C/C++*
Compilation	Third-party IDE or compiler
Low-level verification	SystemTest — Limit Check element*, Simulink Design Verifier — Test Generation, Embedded Coder — PIL test, Embedded Coder — Code coverage tool link (requires third-party code coverage tool), Polyspace Products for C/C++*
High-level verification	SystemTest — Limit Check element*, Embedded Coder — PIL test, Embedded Coder — Code coverage tool link (requires third-party code coverage tool), Polyspace Products for C/C++*
Object code traceability (Level A only)	Embedded Coder — Code generation report, Third-party IDE or compiler — Object code listing
*The DO Qualification Kit product may be used to support DO-178C tool qualification.	

Software Planning Process

The following table contains a summary of the planning process objectives from DO-178C, including the objective, applicable DO-178C reference sections, and software levels applicable to the objective. The table also describes the potential impact to the process when using Model-Based Design.

Table A-1: Software Planning Process

	Objective	**Ref Sections**	**Activity Ref Sectons**	**Software Levels**	**Model-Based Design Process Impact**
1	The activities of the software lifecycle processes are defined.	4.1.a	4.2.a 4.2.c 4.2.d 4.2.e 4.2.g 4.2.i 4.2.l 4.3.c	A, B, C, D	Must include Model-Based Design as part of the development process.
2	The software life cycle(s), including the inter-relationships between the processes, their sequencing, feedback mechanisms, and transition criteria, is defined.	4.1.b	4.2.i 4.3.b	A, B, C	Must include Model-Based Design transition and sequencing relationships.
3	Software life cycle environment is selected and defined.	4.1.c	4.4.1 4.4.2.a 4.4.2.b 4.4.2.c 4.4.3	A, B, C	Must include Model-Based Design tools in the life cycle processes.

Table A-1: Software Planning Process (Continued)

	Objective	Ref Sections	Activity Ref Sectons	Software Levels	Model-Based Design Process Impact
4	Additional considerations are addressed.	4.1.d	4.2.f 4.2.h 4.2.i 4.2.j 4.2.k	A, B, C, D	If applicable to the project and tool qualification, must address any EASA Certification Review Items and FAA Issue Papers. DO Qualification Kit product available for tool qualification.
5	Software development standards are defined.	4.1.e	4.2.b 4.2.g 4.5	A, B, C	As part of the development standards, must include modeling standards.
6	Software plans comply with DO-178C.	4.1.f	4.3.a 4.6	A, B, C	No impact
7	Development and revision of software plans are coordinated.	4.1.g	4.2.g 4.6	A, B, C	No impact

The following sections describe in more detail the potential impacts for each planning process objective when using Model-Based Design, if applicable, as compared to traditional development.

Activities of the Software Lifecycle Processes are Defined

Model-Based Design must be defined as one of the activities in the software development process. This means the DO-331, *Model-Based Development and Verification Supplement to DO-178C and DO-278A*, becomes applicable. Models might be defined as Specification Models or Design Models, as described in DO-331 Section MB.1.6.2. The model definition should be addressed in the planning process. DO-331 Section MB.1.6.3 gives some examples of how Specification and Design Models might be used. As shown in the examples, the responsibility for developing the model requirements, and the models themselves, might be in the systems or software domains. In either case, both the model requirements, and the models themselves, fall under the guidance of DO-331. The responsibilities should be identified in the planning process. The planning process must also identify how to use model simulation and model analysis as part of the verification processes.

If formal methods tools, for example Polyspace, are to be used for certification credit, then DO-333, *Formal Methods Supplement to DO-178C and DO-278A* becomes applicable. The planning process must identify how to use formal analysis as part of the verification processes.

With one possible exception, Model-Based Design does not use object-oriented technology. The possible exception allows Embedded Coder to optionally generate C++ encapsulated code. In this case, the code interface uses classes and methods. Therefore, DO-332, *Object-Oriented Technology and Related Techniques Supplement to DO-178C and DO-278A*, becomes applicable. In this case, the planning process must identify how to develop and verify this object-oriented technology.

Address change control and configuration management of the models during the planning process.

Software Life Cycle is Defined

The software life cycle(s), including the inter-relationships between the processes, their sequencing, feedback mechanisms, and transition criteria, must be defined. When Model-Based Design begins, it must also be defined. This stage is when the higher-level requirements (either system requirements or high-level software requirements) are developed, configured, and approved.

When code is generated, the code must be defined. This stage is when the models have been developed, configured, and approved. The steps to approve the models as complete and correct must be defined and may include model:

- Reviews
- Simulation testing
- Static analysis

- Dynamic analysis

Software Life-Cycle Environment Is Selected and Defined

Model-Based Design tools used in the development and verification processes must be defined. The tools may include the MATLAB, Simulink, Stateflow, MATLAB Coder, Simulink Coder, Embedded Coder, Simulink Code Inspector,Polyspace products for C/C++, Simulink Verification and Validation, Simulink Design Verifier, SystemTest, and Simulink Report Generator products.

Additional Considerations are Addressed

If any Model-Based Design tools are qualified as Criteria 1, Criteria 2, or Criteria 3 tools, as described in DO-178C Section 12.2.2, each of the tools to be qualified must be identified and the tool qualification activities must be defined as described in DO-330, *Software Tool Qualification Considerations*. The DO Qualification Kit product may be used in the qualification of MathWorks verification tools.

Software Development Standards are Defined

Whether Specifications or Design Modes are used (see "Activities of the Software Lifecycle Processes are Defined" on page 2-6), modeling standards must be in place to satisfy the requirements standards objectives. The Simulink documentation contains Modeling Guidelines that can be used as a starting point for developing project specific modeling standards. Compliance to the standards must be verified through the use of tools and/or human reviews. There are Model Advisor checks to verify compliance with the Modeling Guidelines provided in the Simulink documentation.

For the Embedded Coder tool, MISRA® AC AGC[2] coding standards can be used. Some constructs in the generated code, such as naming conventions, can be controlled by users to meet specific customer coding standards. Compliance to the standards must be verified through tools and/or human reviews. Polyspace provides capability to check the MISRA AC AGC rules.

2. The Motor Industry Software Reliability Association. *MISRA AC AGC: Guidelines for the application of MISRA-C:2004 in the context of automatic code generation, ISBN 978-906400-02-6 (PDF), November 2007.* MIRA Limited, 2004.

Software Plans Comply with DO-178C

A Plan for Software Aspects of Certification (PSAC) must be developed, the same as for traditional development programs.

Development and Revision of Software Plans are Coordinated

The Plan for Software Aspects of Certification (PSAC) must be configured under change control and approved by the applicable certification authorities as part of the program, as in a traditional development process.

Software Development Process

The following table contains a summary of the software development process objectives from DO-178C and DO-331, including the objective, applicable DO-178C and DO-331 reference sections, and software levels applicable to the objective. The table also describes the available Model-Based Design tools for satisfying the objectives.

Table A-2: Software Development Process

	Objective	Ref Sections	Activity Ref Sections	Software Levels	Available Products for Model-Based Design
1	High-level requirements are developed.	MB.5.1.1.a	MB.5.1.2.a MB.5.1.2.b MB.5.1.2.c MB.5.1.2.d MB.5.1.2.e MB.5.1.2.f MB.5.1.2.g MB.5.1.2.j MB.5.1.2.k MB.5.1.2.l 5.5a	A, B, C, D	Simulink, Stateflow
2	Derived high-level requirements are defined and provided to the system processes, including the system safety assessment process.	MB.5.1.1.b	MB.5.1.2.h MB.5.1.2.i MB.5.1.2.k	A, B, C, D	Simulink, Stateflow
3	Software architecture is developed.	MB.5.2.1.a	MB.5.2.2.a MB.5.2.2.d MB.5.2.2.h	A, B, C, D	Simulink, Stateflow

Table A-2: Software Development Process (Continued)

	Objective	Ref Sections	Activity Ref Sections	Software Levels	Available Products for Model-Based Design
4	Low-level requirements are developed.	MB.5.2.1.a	MB.5.2.2.a MB.5.2.2.e MB.5.2.2.f MB.5.2.2.g MB.5.2.2.h MB.5.2.3.a MB.5.2.3.b 5.2.4.a 5.2.4.b 5.2.4.c MB.5.5 5.5b	A, B, C	Simulink, Stateflow
5	Derived low-level requirements are defined and provided to the system proceses, including the system safety assessment process.	MB.5.2.1.b	MB.5.2.2.b MB.5.2.2.c MB.5.2.2.h	A, B, C	Simulink, Stateflow
6	Source code is developed.	5.3.1.a	5.3.2.a 5.3.2.b 5.3.2.c 5.3.2.d MB.5.5 5.5.c	A, B, C	Simulink Coder, Embedded Coder

Table A-2: Software Development Process (Continued)

	Objective	Ref Sections	Activity Ref Sections	Software Levels	Available Products for Model-Based Design
7	Executable Object Code and Parameter Data Item Files, if any, are produced and loaded in the target computer.	5.4.1.a	5.4.2.a 5.4.2.b 5.4.2.c 5.4.2.d 5.4.3.e 5.4.3.f	A, B, C, D	Embedded Coder — IDE Link
8	Specification Model elements that do not contribute to implementation or realization of any high-level requirements are identified.	MB.5.1.1.c	MB.5.1.2.k	A, B, C, D	Simulink, Stateflow
9	Design Model elements that do not contribute to implementation or realization of any software architecture are identified.	MB.5.2.1.c	MB.5.2.2.h	A, B, C, D	Simulink, Stateflow
10	Design Model elements that do not contribute to implementation or realization of any low-level requirements are identified.	MB.5.2.1.c	MB.5.2.2.h	A, B, C	Simulink, Stateflow

The following sections describe in more detail the potential impacts for each software development process objective when using Model-Based Design, if applicable, as compared to traditional development.

High-Level Requirements are Developed

If models are defined as Specification Models, as described in DO-331 Sepction MB.1.6.2, then the Simulink and Stateflow products may be used to develop the high-level software requirements. The components within these models, such as Simulink blocks or Stateflow objects, would then trace to the applicable system-level requirements, which are developed in accordance with ARP4754A[3]. The models should be developed in accordance with the modeling standards defined during the planning process.

Derived High-Level Requirements are Defined and Provided to System Processes

If models are defined as Specification Models, as described in DO-331 Section MB.1.6.2, , any Simulink or Stateflow components that do not trace to the system requirements would be identified as derived requirements. These derived requirements would be provided to the safety assessment process.

Software Architecture Is Developed

When models are derived as Design Models, as described in DO-331 Section MB.1.6.2, architecture of individual software modules may be defined by the Simulink and Stateflow models, including sequencing and interfacing of the various elements within the models. If model reference capability is used, then the model dependency viewer may be used to document the architecture of the software modules that are integrated using this capability.

The higher-level architecture of how the Model-Based Design generated code interfaces to other code within the system must be defined separately. This may include an interface to the real-time operating system (RTOS), calling sequence for the code generated from the Model-Based Design, and data interface to other code modules.

3. SAE International. *Guidelines for Development of Civil Aircraft and Systems*, 2010.

Low-Level Requirements are Developed

If models are defined as Design Models, as described in DO-331 Section MB.1.6.2, then the Simulink and Stateflow products may be used to develop the low-level software requirements. The components within these models would then trace to the applicable high-level software requirements. The models should be developed in accordance with the modeling standards defined during the planning process.

Derived Low-Level Requirements are Defined and Provided to the System Processes

If models are defined as Design Models, as described in DO-331 Section MB.1.6.2, then any Simulink or Stateflow components that do not trace to the high-level software requirements would be identified as derived requirements. These derived requirements would be provided to the safety assessment process.

Source Code Is Developed

Embedded Coder and Simulink Coder products may be used to generate the source code from the model. The source code can trace to the model components by using commenting options. The source code can be generated in accordance with MISRA AC AGC standards, with some exceptions, by adhering to modeling standards.

Executable Object Code and Parameter Data Item Files are Produced and Loaded in the Target Computer

The generated source code may be compiled, linked, and the executable object code automatically downloaded to a target processor or DSP using the IDE Link capability of the Embedded Coder product. If parameter data items are used, the parameter data item files should be generated and downloaded to the target processor DSP.

Alternatively, the generated source code may be compiled and linked using standard compilers and linkers. The make file that the compiler uses may be generated by the Embedded Coder product or developed manually. The executable object code is then loaded onto the target computer.

Specification Model Elements That Do Not Contribute to Implementation or Realization of Any High-Level Requirements are Identified

Simulink contains both virtual blocks and non-virtual blocks, as described in the documentation. In general, implementation is defined by non-virtual blocks, not virtual blocks. Examples of virtual blocks include DOC blocks, subsystems used to group blocks together in a model, and some connections inside virtual subsystems, such as inports or outports. The modeling standards for the project should define these types of virtual blocks as not contributing to the implementation.

Design Model Elements That Do Not Contribute to Implementation or Realization of Any Software Architecture are Identified

Simulink contains both virtual blocks and non-virtual blocks, as described in the documentation. In general, implementation is defined by non-virtual blocks, not virtual blocks. Examples of virtual blocks include DOC blocks, subsystems used to group blocks together in a model, and some connections inside virtual subsystems, such as inports or outports. The modeling standards for the project should define these types of non-virtual blocks as not contributing to the implementation. It should be noted that some virtual blocks, such as Mux/Demux or Goto/From, do define data flows for the software architecture. Note that some virtual blocks, such as Mux/Demux or Goto/From blocks, do define data flows for the software architecture.

Design Model Elements That Do Not Contribute to Implementation or Realization of Any Low-Level Requirements are Identified

Simulink contains both virtual blocks and non-virtual blocks, as described in the documentation. In general, implementation is defined by non-virtual blocks, not virtual blocks. Examples of virtual blocks include DOC blocks, subsystems used to group blocks together in a model, and some connections inside virtual subsystems, such as inports or outports. The modeling standards for the project should define these types of non-virtual blocks as not contributing to the implementation. It should be noted that some virtual blocks, such as Mux/Demux or Goto/From, do define data flows for the software architecture.

Verification of Requirements Process

The following table contains a summary of the verification of requirements process objectives from DO-178C and DO-331, including the objective, applicable DO-178C and DO-331 reference sections, and software levels applicable to the objective. The table also provides the available Model-Based Design tools that may be used in satisfying the objectives.

Table A-3: Verification of Requirements Process

	Objective	Ref Sections	Activity Ref Sectons	Software Levels	Available Products for Model-Based Design
1	High-level requirements comply with system requirements.	MB.6.3.1.a	MB.6.3.1 MB.6.8.1	A, B, C, D	Simulink Verification and Validation, Simulink Design Verifier, SystemTest, Simulink Report Generator, DO Qualification Kit
2	High-level requirements are accurate and consistent.	MB.6.3.1.b	MB.6.3.1 MB.6.8.1	A, B, C, D	Simulink Verification and Validation, SystemTest, Simulink Report Generator, DO Qualification Kit
3	High-level requirements are compatible with target computer.	MB.6.3.1.c	MB.6.3.1	A, B	Simulink Verification and Validation, Simulink Report Generator, DO Qualification Kit
4	High-level requirements are verifiable.	MB.6.3.1.d	MB.6.3.1 MB.6.8.1	A, B, C	Simulink Verification and Validation, Simulink Design Verifier, SystemTest, Simulink Report Generator, DO Qualification Kit

Table A-3: Verification of Requirements Process (Continued)

	Objective	Ref Sections	Activity Ref Sectons	Software Levels	Available Products for Model-Based Design
5	High-level requirements conform to standards.	MB.6.3.1.e	MB.6.3.1	A, B, C	Simulink Verification and Validation, Simulink Report Generator, DO Qualification Kit
6	High-level requirements are traceable to system requirements.	MB.6.3.1.f	MB.6.3.1	A, B, C, D	Simulink Verification and Validation, Simulink Report Generator, DO Qualification Kit
7	Algorithms are accurate.	MB.6.3.1.g	MB.6.3.1 MB.6.8.1	A, B, C	Simulink Verification and Validation, Simulink Design Verifier, SystemTest, Simulink Report Generator, DO Qualification Kit
8	Simulation cases are correct	MB.6.8.3.2.a	MB.6.8.1 MB.6.8.3.2	A, B, C, D	SystemTest, Simulink Report Generator
9	Simulation procedures are correct	MB.6.8.3.2.b	MB.6.8.1 MB.6.8.3.2	A, B, C, D	SystemTest, Simulink Report Generator
10	Simulation results are correct and discrepancies explained	MB.6.8.3.2.c	MB.6.8.1 MB.6.8.3.2	A, B, C, D	SystemTest, Simulink Report Generator

The following sections describe in more detail the potential impacts for each of the verification of requirements process objectives when using Model-Based Design, if applicable, as compared to traditional development.

High-Level Requirements Comply with System Requirements

If models are defined as Specification Models, as described in DO-331 Section MB.1.6.2, compliance with system requirements may be accomplished using a combination of model reviews, model analysis, and simulation. The Simulink Report Generator product may be used to generate a System Design Description report that includes a trace report to the system requirements. The SystemTest and Simulink Verification and Validation products may be used to develop test cases based on the system requirements, and execute those test cases on the model to assist in verifying that the system requirements are satisfied. The Simulink Design Verifier product may be used to prove properties of the model to assist in verifying certain system requirements are satisfied.

The following capabilities may be qualified as a verification tool using the DO Qualification Kit product:

- When used for pass and fail determination, the Limit Check element in the SystemTest product.
- System Design Description report in the Simulink Report Generator product.

High-Level Requirements Are Accurate and Consistent

If models are defined as Specification Models, as described in DO-331 Section MB.1.6.2, accuracy and consistency may be verified using a combination of model reviews and simulation. The Simulink Report Generator product may be used to generate a System Design Description report that includes a trace report to the higher-level requirements. The SystemTest and Simulink Verification and Validation products may be used to develop and execute test cases based on the system requirements to assist in verifying the accuracy and consistency. The Model Advisor may be used to assist in verifying the diagnostic settings used during Simulink simulations, and also to check the usage of certain Simulink blocks.

The following capabilities may be qualified as a verification tool using the DO Qualification Kit product:

- When used for pass and fail determination, the Limit Check element in the SystemTest product.
- DO-178C/DO-331 checks in the Simulink Verification and Validation product.
- System Design Description report in the Simulink Report Generator product.

High-Level Requirements Are Compatible withTarget Computer

If models are defined as Specification Models, as described in DO-331 Section MB.1.6.2, compatibility with target hardware may be accomplished using a combination of model reviews and Model Advisor checks. The Simulink Report Generator product may be used to generate a System Design Description report that includes a trace report to the higher-level requirements. The Model Advisor may be used to assist in verifying that the hardware interface settings used by the Embedded Coder product are compatible with the target processor.

The following capabilities may be qualified as a verification tool using the DO Qualification Kit product:

- DO-178C/DO-331 checks in the Simulink Verification and Validation product.
- System Design Description report in the Simulink Report Generator product.

High-Level Requirements Are Verifiable

If models are defined as Specification Models, as described in DO-331 Section MB.1.6.2, verification may be accomplished using a combination of model reviews and simulation. The Simulink Report Generator product may be used to generate a System Design Description report that includes a trace report to the higher-level requirements. The SystemTest and Simulink Verification and Validation products may be used to develop test cases from the system requirements and execute those test cases on the model. During execution of these test cases, a Simulink Verification and Validation model coverage report may be generated to assist in verifying that all requirements are fully verified. The coverage report may assist in finding conditions and decisions in the model that cannot be reached, indicating that the requirements may not be fully verifiable. The Simulink Design Verifier product may be used to identify untestable or unreachable model conditions and decisions using test case generation, indicating that the high-level requirements may not be fully verifiable. The Model Advisor may be used to assist in checking the usage of certain Simulink blocks and data types.

The following capabilities may be qualified as a verification tool using the DO Qualification Kit product:

- When used for pass and fail determination, the Limit Check element in the SystemTest product.
- DO-178C/DO-331 checks in the Simulink Verification and Validation product.
- Model coverage in the Simulink Verification and Validation product.
- System Design Description report in the Simulink Report Generator product.

High-Level Requirements Conform to Standards

If models are defined as Specification Models, as described in DO-331 Section MB.1.6.2, conformance to standards may be accomplished using a combination of model reviews and Model Advisor checks. The Simulink Report Generator product may be used to generate a System Design Description report that includes a trace report to the higher-level requirements. The Model Advisor may verify predefined model standards, and may be customized using an API to perform checks defined by the user that may be unique to their application.

The following capabilities may be qualified as a verification tool using the DO Qualification Kit product:

- DO-178C/DO-331 checks in the Simulink Verification and Validation product.
- Custom checks added by the user, but the user is responsible for defining the Tool Operational Requirements, Test Cases, Procedures, and Expected Results for those custom checks.
- System Design Description report in the Simulink Report Generator product.

High-Level Requirements Are Traceable to System Requirements

If models are defined as Specification Models, as described in DO-331 Section MB.1.6.2, traceability to system requirements may be accomplished by model reviews that include a report generated by the Requirements Management Interface (RMI), a capability of the Simulink Verification and Validation product. The Simulink Report Generator product may be used to generate a System Design Description report that includes a trace report to the system requirements. The Model Advisor may be used to assist in verifying that requirements links are consistent, and can identify model components that do not trace to requirements.

The following capabilities may be qualified as a verification tool using the DO Qualification Kit product:

- DO-178C/DO-331 checks in the Simulink Verification and Validation product.
- System Design Description report in the Simulink Report Generator product.

Algorithms Are Accurate

If models are defined as Specification Models, as described in DO-331 Section MB.1.6.2, accuracy of the algorithms may be verified using a combination of model reviews and simulation. The Simulink Report Generator product may be used to generate a System Design Description report that includes a trace report to the higher-level requirements. The SystemTest and Simulink Verification and Validation products may be used to develop test cases from the system requirements and execute those test cases on the model, assisting in verifying the accuracy of the algorithms within the model. The Model Advisor may be used to assist in checking the usage of certain Simulink blocks and data types. The Simulink Design Verifier design error detection capability may be used to assist in finding potential divide by zero or numeric overflow computations that could lead to incorrect behavior.

The following capabilities may be qualified as a verification tool using the DO Qualification Kit product:

- When used for pass and fail determination, the Limit Check element in the SystemTest product.
- DO-178C/DO-331 checks in the Simulink Verification and Validation product.
- System Design Description report in the Simulink Report Generator product.

Simulation Cases Are Correct

Simulation cases may be developed using SystemTest or Simulink Report Generator. These test cases need to be reviewed against the system requirements.

Simulation Procedures Are Correct

Simulation procedures may be developed using SystemTest or Simulink Report Generator. These test procedures need to be reviewed against the system requirements and test cases.

Simulation Results Are Correct and Discrepancies Explained

Simulations may be executed using SystemTest or Simulink Report Generator. In the case of SystemTest, the Limit Check element can be used to compare the expected results to actual results. The simulation results need to be reviewed and failures need to be corrected or explained.

The following capabilities may be qualified as a verification tool using the DO Qualification Kit product:

- When used for pass and fail determination, the Limit Check element in the SystemTest product.

Verification of Design Process

The following table contains a summary of the verification of design process objectives from DO-178C and DO-331, including the objective, applicable DO-178C and DO-331 reference sections, and software levels applicable to the objective. The table also describes the available Model-Based Design tools for satisfying the objectives.

Table A-4: Verification of Design Process

	Objective	Ref Sections	Activiity Ref Sections	Software Levels	Available Products for Model-Based Design
1	Low-level requirements comply with high-level requirements.	MB.6.3.2.a	MB.6.3.2 MB.6.8.1	A, B, C	Simulink Verification and Validation, Simulink Design Verifier, SystemTest, Simulink Report Generator, DO Qualification Kit
2	Low-level requirements are accurate and consistent.	MB.6.3.2.b	MB.6.3.2 MB.6.8.1	A, B, C	Simulink Verification and Validation, SystemTest, Simulink Report Generator, DO Qualification Kit
3	Low-level requirements are compatible with the target computer.	MB.6.3.2.c	MB.6.3.2	A, B	Simulink Verification and Validation, Simulink Report Generator, DO Qualification Kit
4	Low-level requirements are verifiable.	MB.6.3.2.d	MB.6.3.2 MB.6.8.1	A, B	Simulink Verification and Validation, Simulink Design Verifier, SystemTest, Simulink Report Generator, DO Qualification Kit
5	Low-level requirements conform to standards.	MB.6.3.2.e	MB.6.3.2	A, B, C	Simulink Verification and Validation, Simulink Report Generator, DO Qualification Kit

Table A-4: Verification of Design Process (Continued)

	Objective	Ref Sections	Activiity Ref Sections	Software Levels	Available Products for Model-Based Design
6	Low-level requirements are traceable to high-level requirements.	MB.6.3.2.f	MB.6.3.2	A, B, C	Simulink Verification and Validation, Simulink Report Generator, DO Qualification Kit
7	Algorithms are accurate.	MB.6.3.2.g	MB.6.3.2 MB.6.8.1	A, B, C	Simulink Verification and Validation, Simulink Design Verifier, SystemTest, Simulink Report Generator, DO Qualification Kit
8	Software architecture is compatible with high-level requirements.	MB.6.3.3.a	MB.6.3.3 MB.6.8.1	A, B, C	Simulink Report Generator
9	Software architecture is consistent.	MB.6.3.3.b	MB.6.3.3 MB.6.8.1	A, B, C	Simulink Verification and Validation, Simulink Report Generator, DO Qualification Kit
10	Software architecture is compatible with target computer.	MB.6.3.3.c	MB.6.3.3	A, B	Simulink Verification and Validation, Simulink Report Generator, DO Qualification Kit
11	Software architecture is verifiable.	MB.6.3.3.d	MB.6.3.3 MB.6.8.1	A, B	Simulink Verification and Validation, Simulink Report Generator, DO Qualification Kit
12	Software architecture conforms to standards.	MB.6.3.3.e	MB.6.3.3	A, B, C	Simulink Verification and Validation, Simulink Report Generator, DO Qualification Kit

Table A-4: Verification of Design Process (Continued)

	Objective	Ref Sections	Activiity Ref Sections	Software Levels	Available Products for Model-Based Design
13	Software partitioning integrity is confirmed.	MB.6.3.3.f	MB.6.3.3	A, B, C, D	Not applicable
14	Simulation cases are correct	MB.6.8.3.2.a	MB.6.8.1 MB.6.8.3.2	A, B, C	SystemTest, Simulink Report Generator
15	Simulation procedures are correct	MB.6.8.3.2.b	MB.6.8.1 MB.6.8.3.2	A, B, C	SystemTest, Simulink Report Generator
16	Simulation cases are correct and discrepancies explained	MB.6.8.3.2.c	MB.6.8.1 MB.6.8.3.2	A, B, C	SystemTest, Simulink Report Generator, DO Qualification Kit

The following sections describe in more detail the potential impacts for each of the verification of design process objectives when using Model-Based Design, if applicable, as compared to traditional development.

Low-Level Requirements Comply with High-Level Requirements

If models are defined as Design Models, as described in DO-331 Section MB.1.6.2, compliance with high-level software requirements may be accomplished using a combination of model reviews, model analysis, and simulation. The Simulink Report Generator product may be used to generate a System Design Description report that includes a trace report to the system requirements. The SystemTest and Simulink Verification and Validation products may be used to develop test cases from the high-level requirements and execute those test cases on the model to assist in verifying that the high-level requirements are satisfied. The model coverage report from Simulink Verification and Validation product may be used to assist in identifying unintended functionality in a Design Model and also to assess the completeness of the simulation cases. The Simulink Design Verifier product may be used to prove properties of the model in order to assist in verifying certain high-level requirements are satisfied. If a Specification Model is also used on a project with a Design Model derived from it, then the Simulink Design Verifier product may be used to generate test cases from the Specification Model. The test cases can then be used to verify the Design Model.

The following capabilities may be qualified as a verification tool using the DO Qualification Kit product:

- When used for pass and fail determination, the Limit Check element in the SystemTest product.
- System Design Description report in the Simulink Report Generator product.
- Model coverage in the Simulink Verification and Validation product.

Low-Level Requirements Are Accurate and Consistent

If models are defined as Design Models, as described in DO-331 Section MB.1.6.2, accuracy and consistency may be verified using a combination of model reviews and simulation. The Simulink Report Generator product may be used to generate a System Design Description report that includes a trace report to the higher-level requirements. The SystemTest and Simulink Verification and Validation products may be used to develop test cases from the high-level requirements, and execute those test cases on the model to assist in verifying the accuracy and consistency. The Model Advisor may be used to assist in verifying the diagnostic settings used during Simulink simulations, and also to check the usage of certain Simulink blocks.

The following capabilities may be qualified as a verification tool using the DO Qualification Kit product:

- When used for pass and fail determination, the Limit Check element in the SystemTest product.
- DO-178C/DO-331 checks in the Simulink Verification and Validation product.

- System Design Description report in the Simulink Report Generator product.

Low-Level Requirements Are Compatible with Target Computer

If models are defined as Design Models, as described in DO-331 Section MB.1.6.2, compatibility with target hardware may be accomplished using a combination of model reviews and Model Advisor checks. The Simulink Report Generator product may be used to generate a System Design Description report that includes a trace report to the higher-level requirements. The Model Advisor may be used to assist in verifying that the hardware interface settings used by the Embedded Coder product are compatible with the target processor.

The following capabilities may be qualified as a verification tool using the DO Qualification Kit product:

- DO-178C/DO-331 checks in the Simulink Verification and Validation product.
- System Design Description report in the Simulink Report Generator product.

Low-Level Requirements Are Verifiable

If models are defined as Design Models, as described in DO-331 Section MB.1.6.2, verifiability may be accomplished using a combination of model reviews and simulation. The Simulink Report Generator product may be used to generate a System Design Description report that includes a trace report to the higher-level requirements. The SystemTest and Simulink Verification and Validation products may be used to develop test cases from the high-level requirements, and execute those test cases on the model. During execution of these test cases, a Simulink Verification and Validation model coverage report may be generated to assist in verifying that all requirements are fully verified. The coverage report may assist in finding conditions and decisions in the model that cannot be reached, indicating that the design may not be fully verifiable. The Simulink Design Verifier product may be used to identify untestable or unreachable model conditions and decisions using test case generation, indicating that the low-level requirements may not be fully verifiable. The Model Advisor may be used to assist in checking the usage of certain Simulink blocks and data types.

The following capabilities may be qualified as a verification tool using the DO Qualification Kit product:

- When used for pass and fail determination, the Limit Check element in the SystemTest product.
- DO-178C/DO-331 checks in the Simulink Verification and Validation product.
- Model coverage in the Simulink Verification and Validation product.
- System Design Description report in the Simulink Report Generator product.

Low-Level Requirements Conform to Standards

If models are defined as Design Models, as described in DO-331 Section MB.1.6.2, conformance to standards may be accomplished using a combination of model reviews and Model Advisor checks. The Simulink Report Generator product may be used to generate a System Design Description report that includes a trace report to the higher-level requirements. The Model Advisor may be used to verify predefined model standards and may also be customized to perform checks defined by the user that are unique for their application. The model coverage report from Simulink Verification and Validation product may be used to determine Cyclomatic complexity of the model. Model complexity criteria is typically a part of the modeling standards.

The following capabilities may be qualified as a verification tool using the DO Qualification Kit product:

- DO-178C/DO-331 checks in the Simulink Verification and Validation product.
- Custom checks added by the user, but the user is responsible for defining the Tool Operational Requirements, Test Cases, Procedures, and Expected Results for those custom checks.
- System Design Description report in the Simulink Report Generator product.

- Model coverage in the Simulink Verification and Validation product.

Low-Level Requirements Are Traceable to High-Level Requirements

If models are defined as Design Models, as described in DO-331 Section MB.1.6.2, traceability to high-level software requirements may be accomplished using a combination of model reviews and the Requirements Management Interface (RMI). The Simulink Report Generator product may be used to generate a System Design Description report that includes a trace report to the high-level software requirements. The Model Advisor may be used to assist in verifying that requirements links are consistent.

The following capabilities may be qualified as a verification tool using the DO Qualification Kit product:

- DO-178C/DO-331 checks in the Simulink Verification and Validation product.
- System Design Description report in the Simulink Report Generator product.

Algorithms Are Accurate

If models are defined as Design Models, as described in DO-331 Section MB.1.6.2, accuracy of the algorithms may be verified using a combination of model reviews and simulation. The Simulink Report Generator product may be used to generate a System Design Description report that includes a trace report to the higher-level requirements. The SystemTest and Simulink Verification and Validation products may be used to develop test cases from the high-level requirements, and execute those test cases on the model, assisting in verifying the accuracy of the algorithms within the model. The Model Advisor may be used to assist in checking the usage of certain Simulink blocks and data types. The Simulink Design Verifier design error detection capability may be used to assist in finding potential divide by zero or numeric overflow computations that could lead to incorrect behavior.

The following capabilities may be qualified as a verification tool using the DO Qualification Kit product:

- When used for pass and fail determination, the Limit Check element in the SystemTest product.
- System Design Description report in the Simulink Report Generator product.

Software Architecture Is Compatible with High-Level Requirements

Compatibility of the software architecture within the models may be verified using model reviews. The Simulink Report Generator product may be used to generate a System Design Description report. The Model Dependency Viewer in the Simulink product can show the architecture of reference models and library blocks.

The System Design Description report capability in the Simulink Report Generator product may be qualified as a verification tool using the DO Qualification Kit product.

The higher-level software architecture, which includes the real-time operating system (RTOS) and other code, may be verified using traditional methods.

Software Architecture Is Consistent

Consistency of the software architecture within the models may be verified using model reviews. The Simulink Report Generator product may be used to generate a System Design Description report. The Model Dependency Viewer in the Simulink product can show the architecture of reference models and library blocks. The Model Advisor may be used to assist in verifying the diagnostic settings used during Simulink simulations, and also to check the usage of certain Simulink blocks.

The following capabilities may be qualified as a verification tool using the DO Qualification Kit product:

- DO-178C/DO-331 checks in the Simulink Verification and Validation product.
- System Design Description report in the Simulink Report Generator product.

The higher-level software architecture, which includes the RTOS and other code, may be verified using traditional methods.

Software Architecture Is Compatible with Target Computer

Target compatibility of the software architecture within the models may be verified using model reviews. The Simulink Report Generator product may be used to generate a System Design Description report. The Model Advisor may be used to verify that the hardware interface settings used by the Embedded Coder product are compatible with the target processor.

The following capabilities may be qualified as a verification tool using the DO Qualification Kit product:

- DO-178C/DO-331 checks in the Simulink Verification and Validation product.
- System Design Description report in the Simulink Report Generator product.

The higher-level software architecture, which includes the RTOS and other code, may be verified using traditional methods.

Software Architecture Is Verifiable

Verification of the software architecture may be accomplished using a combination of model reviews and simulation. The Simulink Report Generator product may be used to generate a System Design Description report. The SystemTest and Simulink Verification and Validation products may be used to develop test cases from the high-level requirements, and execute those test cases on the model. During execution of these test cases, a model coverage report may be generated to assist in verifying that all requirements are fully verified. The coverage report may assist in finding conditions and decisions in the model architecture that cannot be reached, indicating that the software architecture may not be fully verifiable.

The following capabilities may be qualified as a verification tool using the DO Qualification Kit product:

- When used for pass and fail determination, the Limit Check element in the SystemTest product.
- Model coverage in the Simulink Verification and Validation product.
- System Design Description report in the Simulink Report Generator product.

The higher-level software architecture, which includes the RTOS and other code, may be verified using traditional methods.

Software Architecture Conforms to Standards

Conformance to standards may be accomplished using a combination of model reviews and Model Advisor checks. The Simulink Report Generator product may be used to generate a System Design Description report. The Model Advisor may be used to verify predefined model standards, and may also be customized to perform checks defined by the user that are unique for their application.

The following capabilities may be qualified as a verification tool using the DO Qualification Kit product:

- DO-178C/DO-331 checks in the Simulink Verification and Validation product.
- Custom checks added by the user, but the user is responsible for defining the Tool Operational Requirements, Test Cases, Procedures, and Expected Results for those custom checks.
- System Design Description report in the Simulink Report Generator product.

The higher-level software architecture, which includes the RTOS and other code, may be verified using traditional methods.

Software Partitioning Integrity Is Confirmed

Because partitioning is outside of the scope of Model-Based Design, partitioning may be verified using traditional methods.

Simulation Case Are Correct

Simulation cases may be developed using SystemTest or Simulink Report Generator. These test cases need to be reviewed against the high-level requirements.

Simulation Procedures Are Correct

Simulation procedures may be developed using SystemTest or Simulink Report Generator. These test procedures need to be reviewed against the high-level requirements and test cases.

Simulation Results Are Correct and Discrepancies Explained

Simulations may be executed using SystemTest or Simulink Report Generator. Within these tools the actual results can be compared to expected results within the test report. In the case of SystemTest, the Limit Check element can be used to compare the expected results to actual results. The simulation results need to be reviewed and failures need to be corrected or explained.

The following capabilities may be qualified as a verification tool using the DO Qualification Kit product:

- When used for pass and fail determination, the Limit Check element in the SystemTest product.

Verification of Coding and Integration Process

The following table contains a summary of the verification of coding and integration process objectives from DO-178C, DO-331, DO-332 and DO-333, including the objective, applicable DO-178C, DO-331, DO-332 and DO-333 reference sections, and software levels applicable to the objective. The table also describes the available Model-Based Design tools for satisfying the objectives.

Table A-5: Verification of Coding and Integration Process

	Objective	Ref Sections	Activity Ref Sections	Software Levels	Available Products for Model-Based Design
1	Source code complies with low-level requirements.	MB.6.3.4.a	MB.6.3.4	A, B, C	Simulink Code Inspector, DO Qualification Kit
2	Source code complies with software architecture.	MB.6.3.4.b OO.6.3.4.b FM.6.3.4.b FM.6.3.a	MB.6.3.4 OO.6.3.4 FM.6.3.4	A, B, C	Simulink Code Inspector, Polyspace, DO Qualification Kit
3	Source code is verifiable.	MB.6.3.4.c OO.6.3.4.c FM.6.3.4.c FM.6.3.e	MB.6.3.4 OO.6.3.4 FM.6.3.4	A, B	Simulink Code Inspector, Polyspace, DO Qualification Kit
4	Source code conforms to standards.	MB.6.3.4.d OO.6.3.4.d FM.6.3.4.d FM.6.3.f	MB.6.3.4 OO.6.3.4 FM.6.3.4	A, B, C	Polyspace, DO Qualification Kit
5	Source code is traceable to low-level requirements.	MB.6.3.4.e	MB.6.3.4	A, B, C	Simulink Code Inspector, DO Qualification Kit

Table A-5: Verification of Coding and Integration Process (Continued)

	Objective	Ref Sections	Activity Ref Sections	Software Levels	Available Products for Model-Based Design
6	Source code is accurate and consistent.	6.3.4.f OO.6.3.4.f FM.6.3.4.f FM.6.3.b FM.6.3.c	MB.6.3.4 OO.6.3.4 FM.6.3.4	A, B, C	Simulink Code Inspector, Polyspace, DO Qualification Kit
7	Output of software integration process is complete and correct.	6.3.5.a	6.3.5	A, B, C	Not applicable
8	Parameter Data Item File is correct and complete	6.6.a	6.6	A,B,C,D	Not applicable
9	Verification of parameter Data Item File is achieved.	6.6.b	6.6	A,B,C	Not applicable
10	Formal analysis cases and procedures are correct.	FM.6.3.6.a FM.6.3.6.b	FM.6.3.6	A, B, C	Polyspace, DO Qualification Kit
11	Formal analysis results are correct and discrepancies explained.	FM.6.3.6.c	FM.6.3.6	A, B, C	Polyspace, DO Qualification Kit
12	Requirements formalization is correct.	FM.6.3.i	FM.6.3.i	A, B, C	Polyspace, DO Qualification Kit
13	Formal method is correctly justified and appropriate.	FM.6.2.1	FM.6.2.1.a FM.6.2.1.b FM.6.2.1.c	A, B, C, D	Polyspace, DO Qualification Kit

The following sections describe in more detail the potential impacts for each of the verification of coding and integration process objectives when using Model-Based Design, if applicable, as compared to traditional development.

Source Code Complies with Low-Level Requirements

Compliance to low-level requirements may be verified using Simulink Code Inspector, which verifies that the source code complies with the requirements in the model.

Simulink Code Inspector may be qualified using the DO Qualification Kit product.

Source Code Complies with Software Architecture

Compliance to software architecture may be verified using Simulink Code Inspector, which verifies that the source code complies with the architecture defined in the model.

For hand-written code, Polyspace is able to prove adherence to software, because it automatically builds global data dictionary and identification of shared data reading and writing accesses. Polyspace is also able to prove adherence to software architecture, because it automatically builds the application call tree.

Simulink Code Inspector and the Polyspace products for C/C++ may be qualified using the DO Qualification Kit product.

Source Code Is Verifiable

Verifiability of the code may be verified using Simulink Code Inspector, which verifies compliance with the model, and since the model is verifiable, the code is also verifiable. The Polyspace products for C/C++ can assist in the identification of unreachable, and therefore nonverifiable, code. Polyspace is also able to find unreachable code, either hand-written or generated from a model.

Simulink Code Inspector and the Polyspace products for C/C++ may be qualified using the DO Qualification Kit product.

Source Code Conforms to Standards

Standards compliance of source code may be verified using the MISRA C, MISRA C++, or JSF++ rules checker in the Polyspace products for C/C++. The MISRA C checker works with the Simulink product. Polyspace is also able to determine the cyclomatic complexity of the code, which is typically also included in the coding standard.

The Polyspace products for C/C++ may be qualified using the DO Qualification Kit product.

Source Code Is Traceable to Low-Level Requirements

Traceability of source code to low-level requirements may be verified using Simulink Code Inspector, which verifies the traceability between the model and code and provides a traceability report.

Simulink Code Inspector may be qualified tool using the DO Qualification Kit product.

Source Code Is Accurate and Consistent

Accuracy and consistency of source code may be verified using Simulink Code Inspector, which verifies the accuracy and consistency with respect to the model.

The Polyspace products for C/C++ have the capability to identify run-time errors, such as potential underflow, overflow, divide by zero, etc. The Polyspace products for C/C++ also have the capability to detect uninitialized variables and constants.

Simulink Code Inspector and the Polyspace products for C/C++ may be qualified using the DO Qualification Kit product.

Output of Software Integration Process Is Complete and Correct

Because the integration process is outside of the scope of Model-Based Design, the integration process may be verified using traditional methods.

Parameter Data Item File Is Correct and Complete

Because the Parameter Data Item File verification process is outside the scope of Model-Based Design, verify the Parameter Data Item File through traditional methods.

Verification of Parameter Data Item File Is Achieved

Because the Parameter Data Item File verification process is outside the scope of Model-Based Design, verify the Parameter Data Item File through traditional methods.

Formal Analysis Cases and Procedures Are Correct

This is shown through the qualification of Polyspace and the justification of Abstract Interpretation.

Formal Analysis Results Are Correct and Discrepancies Explained

This is accomplished through the review of the Polyspace Run Time Error results report. Any discrepancies must be explained and justified.

Requirements Formalization Is Correct

This is shown through the qualification of Polyspace and the justification of Abstract Interpretation.

Formal Method Is Correctly Justified and Appropriate

The *DO Qualification Kit: Polyspace Client/Server for C/C++ Theoretical Foundation* justifies the soundness of the Abstract Interpretation method used by Polyspace.

Testing of Outputs of Integration Process

The following table contains a summary of the testing of outputs of integration process objectives from DO-178C and DO-333, including the objective, applicable DO-178C and DO-333 reference sections, and software levels applicable to the objective. The table also describes the available Model-Based Design tools for satisfying the objectives.

Table A-6: Testing of Outputs of Integration Process

	Objective	Ref Sections	Activity Ref Sections	Software Levels	Available Products for Model-Based Design
1	Executable Object Code complies with high-level requirements.	6.4.a FM.6.7.a FM.6.7.c	6.4.2 6.4.2.1 6.4.3 6.5 FM.6.7 FM.6.5	A, B, C, D	SystemTest, Simulink Design Verifier, Embedded Coder — IDE Link, Polyspace, DO Qualification Kit
2	Executable Object Code is robust with high-level requirements.	6.4.b FM.6.7.b FM.6.7.c	6.4.2 6.4.2.2 6.4.3 6.5 FM.6.7 FM.6.5	A, B, C, D	SystemTest, Simulink Design Verifier, Embedded Coder — IDE Link, Polyspace, DO Qualification Kit
3	Executable Object Code complies with low-level requirements.	6.4.c FM.6.7.d FM.6.7.c	6.4.2 6.4.2.1 6.4.3 6.5 FM.6.7 FM.6.5	A, B, C	SystemTest, Simulink Design Verifier, Embedded Coder — IDE Link, Polyspace, DO Qualification Kit

Table A-6: Testing of Outputs of Integration Process (Continued)

	Objective	Ref Sections	Activity Ref Sections	Software Levels	Available Products for Model-Based Design
4	Executable Object Code is robust with low-level requirements.	6.4.d FM.6.7.b FM.6.7.c	6.4.2 6.4.2.2 6.4.3 6.5 FM.6.7 FM.6.5	A, B, C	SystemTest, Simulink Design Verifier, Embedded Coder — IDE Link, Polyspace, DO Qualification Kit
5	Executable Object Code is compatible with target computer.	6.4.e	6.4.1.a 6.4.3.a	A, B, C, D	Embedded Coder — IDE Link

The following sections describe in more detail the potential impacts for each testing of outputs of integration process objective when using Model-Based Design, if applicable, as compared to traditional development.

Executable Object Code Complies with High-Level Requirements

The executable object code may be verified by reusing the same test cases that are used to verify the models. During execution of the model verification tests, using the SystemTest product, the inputs and outputs of each model under test can be logged and saved for use in verifying the executable object code.

In the case where a Specification Model exists, the Simulink Design Verifier product may be used to generate high-level tests from the Specification Model. These test cases can be run on the Design Model and the executable object code, and the results compared to the expected results from the Specification Model. The comparison is used to demonstrate that the executable object code complies with the high-level requirements.

The executable object code may be tested on a target processor or DSP using the IDE Link capability of the Embedded Coder product. The SystemTest product may be used to execute these tests and compare the test results to expected results.

When used for pass and fail determination, the Limit Check element capability within the SystemTest product may be qualified as a verification tool using the DO Qualification Kit product.

The Polyspace products for C/C++ may also be used to satisfy this objective by verifying the source code using abstract interpretation. Some errors detected by the Polyspace products for C/C++ may not be detected during traditional dynamic testing.

The Polyspace products for C/C++ help to exhaustively identify:

- Uninitialized variables
- Parameter passing errors
- Data corruption, especially global data
- Inadequate, end-to-end numerical resolution
- Detection of arithmetic faults
- Detection of violation of array limits

The Polyspace products for C/C++ help to partially identify:

- Incorrect initialization of variables and constants
- Incorrect initialization of variables and constants leading to an underflow or overflow
- Global data corruption of shared variables without protection mechanism
- Incorrect sequencing of events and operations
- Detection of loops leading to run-time error
- Detection of incorrect logic decision leading to unreachable code or run-time errors

The Polyspace products for C/C++ may be qualified as a verification tool using the DO Qualification Kit product.

Executable Object Code Is Robust with High-Level Requirements

Robustness tests should be developed against the models and may be done using the SystemTest product. The robustness of the executable object code may be verified by reusing the same test cases that are used to verify robustness of the models. During execution of the model verification tests, using the SystemTest product, the inputs and outputs of each model under test can be logged and saved for use in verifying the executable object code.

In the case where a Specification Model exists, the Simulink Design Verifier product may be used to generate robustness tests from the Specification Model. These test cases can be run on the Design Model and the executable object code, and the results compared to the expected results from the Specification Model. The comparison demonstrates that the executable object code is robust with the high-level requirements. For robustness test cases, Test Condition and Test Objective blocks may be used to assist in the definition of test cases that exercise the object code outside of normal boundary conditions.

The executable object code may be tested on a target processor or DSP using the IDE Link capability of the Embedded Coder product. The SystemTest product may be used to execute these tests and compare the test results to expected results.

When used for pass and fail determination, the Limit Check element capability within the SystemTest product may be qualified as a verification tool using the DO Qualification Kit product.

The Polyspace products for C/C++ may also be used to satisfy this objective by verifying the source code using abstract interpretation. Some of the errors detected by the Polyspace products for C/C++ may not be detected during traditional dynamic testing.

The Polyspace products for C/C++ help to partially identify:

- Incorrect initialization of variables and constants
- Incorrect initialization of variables and constants leading to an underflow or overflow

- Detection of loops leading to run-time error
- Detection of overflows
- Detection of certain run-time errors

The Polyspace products for C/C++ may be qualified as a verification tool using the DO Qualification Kit product.

Executable Object Code Complies with Low-Level Requirements

The Simulink Design Verifier product may be used to generate low-level tests from the model. These test cases can be run on the model and the executable object code, and the results compared. The comparison is used to demonstrate that the executable object code complies with the low-level requirements.

The executable object code may be tested on a target processor or DSP using the IDE Link capability of the Embedded Coder product. The SystemTest product may be used to execute these tests and compare the test results to expected results.

When used for pass and fail determination, the Limit Check element capability within the SystemTest product may be qualified as a verification tool using the DO Qualification Kit product.

The Polyspace products for C/C++ may also be used to satisfy this objective by verifying the source code using abstract interpretation. Some of the errors detected by the Polyspace products for C/C++ may not be detected during traditional dynamic testing.

The Polyspace products for C/C++ help to exhaustively identify:

- Uninitialized variables
- Parameter passing errors
- Data corruption, especially global data
- Inadequate, end-to-end numerical resolution
- Detection of arithmetic faults

- Detection of violation of array limits

The Polyspace products for C/C++ help to partially identify:

- Incorrect initialization of variables and constants
- Incorrect initialization of variables and constants leading to an underflow or overflow
- Global data corruption of shared variables without protection mechanism
- Incorrect sequencing of events and operations
- Detection of loops leading to run-time error
- Detection of incorrect logic decision leading to unreachable code or run-time errors

The Polyspace products for C/C++ may be qualified as a verification tool using the DO Qualification Kit product.

Alternatively, verification against the low-level requirements may be eliminated, if requirements based coverage and structural coverage are achieved using the high-level requirements based tests (for example, software integration tests). The following guidance is provided in section 6.4 of DO-178C:

> *If a test case and its corresponding test procedure are developed and executed for hardware/software integration testing or software integration testing and satisfy the requirements-based coverage and structural coverage, it is not necessary to duplicate the test for low-level testing. Substituting nominally equivalent low-level tests for high-level tests may be less effective due to the reduced amount of overall functionality tested.*

Executable Object Code Is Robust with Low-Level Requirements

The Simulink Design Verifier product may be used to generate robustness tests from the model. These test cases can be run on the model and the executable object code, and the results compared. The comparison demonstrates that the executable object code is robust with the low-level requirements. For robustness test cases, Test Condition and Test Objective blocks may be used to assist in the definition of test cases that exercise the object code outside of normal boundary conditions.

The executable object code may be tested on a target processor or DSP using the IDE Link capability of the Embedded Coder product. The SystemTest product may be used to execute these tests and compare the test results to expected results.

When used for pass and fail determination, the Limit Check element capability within the SystemTest product may be qualified as a verification tool using the DO Qualification Kit product.

The Polyspace products for C/C++ may also be used to satisfy this objective by verifying the source code using abstract interpretation. Some of the errors detected by the Polyspace products for C/C++ may not be detected during traditional dynamic testing.

The Polyspace products for C/C++ help to partially identify:

- Incorrect initialization of variables and constants
- Incorrect initialization of variables and constants leading to an underflow or overflow
- Detection of loops leading to run-time error
- Detection of overflows
- Detection of certain run-time errors

The Polyspace products for C/C++ may be qualified as a verification tool using the DO Qualification Kit products.

Executable Object Code Is Compatible with Target Computer

The executable object code may be evaluated for stack usage, memory usage, and execution time on a target processor or DSP using the IDE Link capability of the Embedded Coder product.

Other aspects of hardware compatibility such as interrupt handling, resource contention, hardware interfaces, partitioning, etc., must be verified using traditional methods.

Verification of Verification Process Results

The following table contains a summary of the verification of verification process results objectives from DO-178C, DO-331, and DO-333 including the objective, applicable DO-178C, DO-331, and DO-333 reference sections, and software levels applicable to the objective. The table also describes the available Model-Based Design tools that may be used in satisfying the objectives.

Table A-7: Verification of Verification Process Results

	Objective	Ref Sections	Activity Ref Sections	Software Levels	Available Products for Model-Based
1	Test procedures are correct.	6.5.4.b	6.4.5	A, B, C	Simulink Verification and Validation, DO Qualification Kit
2	Test results are correct and discrepancies explained.	6.5.4.c	6.4.5	A, B, C	SystemTest, DO Qualification Kit
3	Test coverage of high-level requirements is achieved.	6.4.4.a	6.4.4.1 MB.6.8.2.a	A, B, C, D	Simulink Verification and Validation, DO Qualification Kit
4	Test coverage of low-level requirements is achieved.	6.4.4.b	6.4.4.1 MB.6.7	A, B, C	Simulink Verification and Validation, DO Qualification Kit
5	Test coverage of software structure (modified condition/decision) is achieved.	6.4.4.c	6.4.4.2.a 6.4.4.2.b 6.4.4.2.d 6.4.4.3 MB.6.8.2.a	A	Not applicable (Simulation credit for testing EOC is not taken)

Table A-7: Verification of Verification Process Results (Continued)

	Objective	Ref Sections	Activity Ref Sections	Software Levels	Available Products for Model-Based
6	Test coverage of software structure (decision coverage) is achieved.	6.4.4.c	6.4.4.2.a 6.4.4.2.b 6.4.4.2.d 6.4.4.3 MB.6.8.2.a	A, B	Not applicable (Simulation credit for testing EOC is not taken)
7	Test coverage of software structure (statement coverage) is achieved.	6.4.4.c	6.4.4.2.a 6.4.4.2.b 6.4.4.2.d 6.4.4.3 MB.6.8.2.a	A, B, C	Not applicable (Simulation credit for testing EOC is not taken)
8	Test coverage of software structure (data coupling and control coupling) is achieved.	6.4.4.d	6.4.4.2.c 6.4.4.2.d 6.4.4.3 MB.6.8.2.a	A, B, C	Not applicable (Simulation credit for testing EOC is not taken)
9	Verification of additional code that cannot be traced to Source Code is achieved.	6.4.4.c	6.4.4.2.b	A	Not applicable (Simulation credit for testing EOC is not taken)
MB 10	Simulation cases are correct	MB.6.8.3.2.a	MB.6.8.3.2	A, B, C	Not applicable (Simulation credit for testing EOC is not taken)
MB 11	Simulation procedures are correct	MB.6.8.3.2.b	MB.6.8.3.2	A, B, C	Not applicable (Simulation credit for testing EOC is not taken)

Table A-7: Verification of Verification Process Results (Continued)

	Objective	Ref Sections	Activity Ref Sections	Software Levels	Available Products for Model-Based
MB 12	Simulation results are correct and discrepancies explained	MB.6.8.3.2.c	MB.6.8.3.2	A, B, C	Not applicable (Simulation credit for testing EOC is not
FM 1	Formal analysis cases and procedures are correct	FM.6.7.2.a FM.6.7.2.b	FM.6.7.2	C	Polyspace, DO Qualification Kit
FM 2	Formal analysis results are correct and discrepancies explained	FM.6.7.2.c	FM.6.7.2	A, B, C	Polyspace, DO Qualification Kit
FM 3	Coverage of high-level requirements is achieved	FM.6.7.1.a	FM.6.7.1.1	A, B, C, D	Not Applicable
FM 4	Coverage of low-level requirements is achieved	FM.6.7.1.a	FM.6.7.1.1	A, B, C	Not Applicable
FM 5-8	Verification coverage of software structure is achieved	FM.6.7.1.c	FM.6.7.1.2 FM.6.7.1.3 FM.6.7.1.4 FM.6.7.1.5		Polyspace, DO Qualification Kit

Table A-7: Verification of Verification Process Results (Continued)

	Objective	Ref Sections	Activity Ref Sections	Software Levels	Available Products for Model-Based
FM 9	Verification of property preservation between source and object code	FM.6.7.f	FM.6.7	A, B, C, D	Polyspace, DO Qualification Kit
FM 10	Formal method is correctly defined, justified, and appropriate	FM.6.2.1	FM.6.2.1.a FM.6.2.1.b FM.6.2.1.c	A, B, C, D	Polyspace, DO Qualification Kit

The following sections describe in more detail the potential impacts for each of the verification of verification process results objective when using Model-Based Design, if applicable, as compared to traditional development.

Test Procedures Are Correct

Correctness of the test procedures from the higher-level requirements may be verified by reviewing the test procedures. The Simulink Verification and Validation product may assist in test procedure reviews by providing traceability from the test cases to the requirements, including hyperlinks to the requirements in the higher-level requirements document.

Completeness of the test cases generated by the Simulink Design Verifier product may be verified by executing the test cases on the Simulink model while measuring model coverage during simulation. The expected results produced by Simulink may be verified by reviewing the results.

The model coverage capability in the Simulink Verification and Validation product may be qualified as a verification tool using the DO Qualification Kit product.

Test Results Are Correct and Discrepancies Explained

Correctness of the test results may be verified by reviewing the test results. If using SystemTest in conjunction with PIL mode (Processor in-the-loop) in Simulink, then the Limit Check Element within SystemTestcan be used for Pass/Fail verification of the results. As an alternative, develop a processor-in-the-loop test platform for the executable object code that could be qualified as a verification tool in order to determine pass and fail status of the results.

Test Coverage of High-Level Requirements Is Achieved

Test coverage of high-level software requirements may be verified by reviewing the test cases and traceability to the high-level requirements. The Simulink Verification and Validation product can be used to trace the test cases to the high-level requirements, providing the capability to assist in verifying that each requirement has associated test cases.

Test Coverage of Low-Level Requirements Is Achieved

Test coverage of low-level software requirements may be verified using the Simulink Verification and Validation model coverage report during execution of the high-level requirements based tests. The model coverage report provides data to assist in proving that low-level requirements are fully covered during high-level testing.

The model coverage capability in the Simulink Verification and Validation product may be qualified as a verification tool using the DO Qualification Kit product.

Test Coverage of Software Structure (Modified Condition/Decision) Is Achieved

Modified condition and decision coverage of the software structure may be verified using a commercial, off-the-shelf structural coverage analysis tool. This analysis is accomplished during the execution of the requirements based tests described in “Executable Object Code Complies with High-Level Requirements” on page 2-41.

If requirements-based test cases are developed at the model level and reused for testing of the executable object code, the model coverage capability of the Simulink Verification and Validation product may be used during development of the requirements based test cases. Using the capability helps predict the effectiveness of the test cases in providing structural coverage for the generated code.

Test Coverage of Software Structure (Decision Coverage) Is Achieved

Decision coverage of the software structure may be verified using a commercial, off-the-shelf structural coverage analysis tool. This analysis is accomplished during the execution of the requirements based tests described in “Executable Object Code Complies with High-Level Requirements” on page 2-41.

If requirements-based test cases are developed at the model level and reused for testing of the executable object code, the model coverage capability may be used during development of the requirements based test cases. Using the tool helps predict the effectiveness of the test cases in providing structural coverage for the generated code.

Test Coverage of Software Structure (Statement Coverage) Is Achieved

Statement coverage of the software structure may be verified using a commercial, off-the-shelf structural coverage analysis tool. This analysis is accomplished during the execution of the requirements based tests described in “Executable Object Code Complies with High-Level Requirements” on page 2-41.

If requirements-based test cases are developed at the model level and reused for testing of the executable object code, then the model coverage capability may be used during development of the requirements based test cases. Using the tool helps predict the effectiveness of the test cases in providing structural coverage for the generated code.

Test Coverage of Software Structure (Data Coupling and Control Coupling) Is Achieved

Because the data coupling and control is outside of the scope of code generated using Model-Based Design, data coupling and control may be verified using traditional methods. The test coverage for data coupling and control involves verification of the data interfaces to and from the automatically generated code and the calling sequence of the automatically generated code in relation to other code modules.

Verification of Additional Code That Cannot Be Traced to Source Code Is Achieved

Because the additional object code verification process is outside the scope of Model-Based Design, verify additional object code through traditional methods. However, a sample model of Simulink and Stateflow elements used by a project may be used to generate code that provides a sample of source code for evaluating the traceability to the object code.

Simulation Cases Are Correct

This object is only applicable when credit is taken for simulation in place of executable object code testing. This is not a recommended workflow due to the difficulty of demonstrating equivalence between a host based simulation and the target code.

Simulation Procedures Are Correct

This object is only applicable when credit is taken for simulation in place of executable object code testing. This is not a recommended workflow due to the difficulty of demonstrating equivalence between a host based simulation and the target code.

Simulation Results Are Correct and Discrepancies Explained

This object is only applicable when credit is taken for simulation in place of executable object code testing. This is not a recommended workflow due to the difficulty of demonstrating equivalence between a host based simulation and the target code.

Formal Analysis Cases and Procedures Are Correct

This is shown through the qualification of Polyspace and the justification of Abstract Interpretation.

Formal Analysis Results Are Correct and Discrepancies Explained

This is accomplished through the review of the Polyspace Run Time Error results report. Any discrepancies must be explained and justified.

Coverage of High-Level Requirements Is Achieved

Not applicable, Polyspace does not support this objective.

Coverage of Low-Level Requirements Is Achieved

Not applicable, Polyspace does not support this objective.

Verification Coverage of Software Structure Is Achieved

Polyspace may be used to find unreachable code, whether it is hand written or generated from a model. The structural coverage and data coupling and control coverage objectives must still be achieved during testing, as described in previous sections.

Verification of Property Preservation Between Source And Object Code

Polyspace analysis is only used to take credit for detection of specific error types. Testing of the executable object code against the high- and low-level requirements is still required to fully satisfy the objectives for executable object code. In order to demonstrate preservation of properties between the source and object code, a traceability analysis between the source and object code needs to be accomplished to demonstrate that additional code, not directly traceable to source code, is not inserted. Additionally, tests can be used to show that properties are preserved between low-level requirements, source code, and executable object code.

Formal Method Is Correctly Justified And Appropriate

The *DO Qualification Kit: Polyspace Client/Server for C/C++ Theoretical Foundation* justifies the soundness of the Abstract Interpretation method used by Polyspace.

Software Configuration Management Process

The following table contains a summary of the configuration management process objectives from DO-178C, including the objective, applicable DO-178C reference sections, and software levels applicable to the objective. The table also describes the potential impact to the process when using Model-Based Design.

Table A-8: Software Configuration Management Process

	Objective	Ref Sections	Activity Ref Sections	Software Levels	Model-Based Design Process Impact
1	Configuration items are identified.	7.2.a	7.2.1	A, B, C, D	No impact
2	Baselines and traceability are established.	7.1.b	7.2.2	A, B, C, D	Use of Requirements Management Interface (RMI) and traditional baseline establishment
3	Problem reporting, change control, change review, and configuration status accounting are established.	7.1.c 7.1.d 7.1.e 7.1.f	7.2.3 7.2.4 7.2.5 7.2.6	A, B, C, D	No impact
4	Archive, retrieval, and release are established.	7.1.g	7.2.7	A, B, C, D	No impact
5	Software load control is established.	7.1.h	7.4	A, B, C, D	No impact
6	Software life cycle environment control is established.	7.1.i	7.5	A, B, C, D	No impact

The following sections describe in more detail the potential impacts for each configuration management process objective when using Model-Based Design, if applicable, as compared to traditional development.

Configuration Items Are Identified

For projects using Model-Based Design, throughout the project, the following artifacts may have to be configured and identified:

- High-level requirements (level above the models)
- Models
- System Design Description and trace reports
- Model Advisor reports
- Automatically generated code
- Code Inspection reports
- Polyspace Code Standards reports
- Polyspace Run-Time Error reports
- Model test harnesses
- Model test scripts
- SystemTest files
- Model test results reports
- Model coverage reports
- Object code structural coverage reports

These artifacts are in addition to, or substitute for, traditional configured items.

Baselines and Traceability Are Established

Establishing baselines and traceability is the same as for traditional projects. Part of the traceability may be covered by the Requirements Management Interface (RMI).

Problem Reporting, Change Control, Change Review, and Configuration Status Accounting Are Established

Establishing problem reporting, change control, change review, and configuration status accounting is the same as for traditional projects.

Archive, Retrieval, and Release Are Established

Establishing archive, retrieval, and release is the same as for traditional projects. The version of the Model-Based Design tools used on the project may have to be archived.

Software Load Control Is Established

Establishing software load control is the same as for traditional projects.

Software Life Cycle Environment Control Is Established

Establishing software life cycle environment control is the same as for traditional projects.

Software Quality Assurance Process

The following table contains a summary of the software quality assurance process objectives from DO-178C, including the objective, applicable DO-178C reference sections, and software levels applicable to the objective is applicable to. The table also describes the potential impact to the process when using Model-Based Design.

Table A-9: Software Quality Assurance Process

	Objective	Ref Sections	Activity Ref Sections	Software Levels	Model-Based Design Process Impact
1	Assurance is obtained that software plans and standards are developed and reviewed for compliance with DO-178C and for consistency.	8.1.a	8.2.b 8.2.h 8.2.i	A, B, C	No impact
2	Assurance is obtained that software life cycle processes comply with approved software plans.	8.1.b	8.2.a 8.2.c 8.2.d 8.2.f 8.2.h 8.2.i	A, B,C,D	No impact
3	Assurance is obtained that software life cycle processes comply with approved software standards.	8.1b	8.2.a 8.2.c 8.2.d 8.2.f 8.2.h 8.2.i	A, B, C	No impact

Table A-9: Software Quality Assurance Process (Continued)

	Objective	Ref Sections	Activity Ref Sections	Software Levels	Model-Based Design Process Impact
4	Assurance is obtained that transition criteria for the software life cycle processes are satisfied.	8.1.c	8.2.e 8.2.h 8.2.i	A,B,C	
5	Assurance is obtained that software conformity review is conducted.	8.1.d	8.2.g 8.2.h 8.3	A,B,C,D	

The following sections describe in more detail the potential impacts for each software quality assurance process objective when using Model-Based Design, if applicable, as compared to traditional development.

Assurance Is Obtained That Software Plans and Standards are Developed and Reviewed for Compliance With DO-178C and For Consistency

Obtaining assurance that software plans and standards are developed and reviewed for compliance to DO-178C for consistency is the same as obtaining it for traditional projects.

Assurance is Obtained That Software Life Cycle Processes Comply with Approved Software Plans

Obtaining assurance that the software life cycle processes comply with approved software plans is the same as obtaining it for traditional projects.

Assurance is Obtained That Software Life Cycle Processes Comply with Approved Software Standards

Obtaining assurance that the software life cycle processes comply with approved software plans is the same as obtaining it for traditional projects.

Assurance is Obtained That Transition Criteria for the Software Life Cycle Processes are Satisfied

Obtaining assurance that transition criteria for the software life cycle processes are satisfied is the same as obtaining it for traditional projects.

Assurance is Obtained That Software Conformity Review is Conducted

Completing software conformity review is the same as completing it for traditional projects.

Certification Liaison Process

The following table contains a summary of the certification liaison process objectives from DO-178C, including the objective, applicable DO-178C reference sections, and software levels applicable to the objective. The table also describes the potential impact to the process when using Model-Based Design.

Table A-10: Certification Liaison Process

	Objective	Ref Sections	Activity Ref Sections	Software Levels	Model-Based Design Process Impact
1	Communication and understanding between the applicant and the certification authority is established.	9.a	9.1.b 9.1.c	A, B, C, D	No impact
2	The means of compliance is proposed and agreement with the Plan for Software Aspects of Certification is obtained.	9.b	9.1.a 9.1.b 9.1.c	A, B, C, D	No impact
3	Compliance substantiation is provided.	9.c	9.2.a 9.2.b 9.2.c	A, B, C, D	No mpact

The following sections describe in more detail the potential impact for each certification liaison process objective when using Model-Based Design, if applicable, as compared to traditional development.

Communication and Understanding Between the Applicant and the Certification Authority Is Established

Establishing communication and understanding between the applicant and the certification authority is the same as for traditional projects.

The Means of Compliance Is Proposed and Agreement with the Plan for Software Aspects of Certification is Obtained

Proposing the means of compliance and obtaining agreement with the Plan for Software Aspects of Certification (PSAC) is the same as for traditional projects.

Compliance Substantiation Is Provided

Providing compliance substantiation is the same as for traditional projects.

Acronyms

Acronyms

API	Application Programming Interface
CRI	Certification Review Item
EASA	European Aviation Safety Agency
FAA	Federal Aviation Administration
IP	Issue Paper
PIL	Processor-In-the-Loop
PSAC	Plan for Software Aspects of Certification
RMI	Requirements Management Interface
RTOS	real-time operating system

References

Normative References

The Motor Industry Software Reliability Association. *MISRA AC AGC: Guidelines for the application of MISRA-C:2004 in the context of automatic code generation, ISBN 978-906400-02-6 (PDF), November 2007.* MIRA Limited, 2004.

SAE International. *Guidelines for Development of Civil Aircraft and Systems*, 2010.

Index