Model Quality Objectives

For embedded software development

Author: MQO Working Group | Version 2 | Release Date 2025

MathWorks®

Revisions

Version	Date	Description	Paragraphs Modified
1.0	June 2018	First version of Model Quality Objectives	N/A
2.0	March 2025	Second version of Model Quality Objectives	All

Contents

1 Introduction

1.1 Abstract

This document presents standard quality objectives for models developed with Simulink® at different phases of the software development lifecycle. This standard, named Model Quality Objectives (MQO), has been defined by a group of leading actors from the automotive industry and MathWorks, the company that develops the MATLAB®, Simulink, and Polyspace® products. The purpose of these guidelines is to clarify and ease the collaboration when sharing Simulink models in the context of embedded software development to drive the production of higher quality and integrity software.

1.2 Intended Audience

The intended audience of this document is Simulink users and project/quality/safety managers interested in establishing a standard approach to assess the quality of design models used at different phases of an embedded software development.

1.3 Scope

The use of design models developed with the Simulink software and its toolboxes in the context of embedded software development with Model-Based Design.

1.4 Purpose

This document clarifies how Simulink design models contribute to accelerate development and verification activities from requirements specification to software implementation. Five types of models with specific purposes are introduced, each with a specific quality objective to control their proper usage. Each quality objective is a set of measurable metrics with quantified satisfaction criteria to facilitate and standardize model quality assessment.

The organizations that apply the concepts presented in this paper should experience the following benefits:

a) Shared understanding of Model-Based Design within the organization
b) Application of a quality model adapted to Model-Based Design projects and compatible with industry software quality and safety standards
c) Assessment of model quality at different phases of projects

The organizations that also collaborate with partners to execute Model-Based Design projects should experience the following benefits when applying the concepts presented in this paper:

a) Improved split of responsibility between parties at the beginning of projects
b) Common understanding of model quality
c) Common expectation on model quality when sharing models

1.5 Background and Motivation

Design models developed with the Simulink software are widely used in the industry to accelerate the development of embedded software. Those models enable engineers to accomplish various engineering tasks such as frequency-domain analysis, desktop simulation, formally-based verification, and automatic code generation. This development process is known as Model-Based Design.

Design models can be developed at a very early stage to validate requirements and quickly explore design solutions. Such models can also be incrementally refined until they reach a level of maturity that is sufficient to generate code that complies with international software safety standards. To incrementally increase the maturity of the design models, different engineering disciplines need to be involved such as system engineering, control engineering and software engineering. Collaborating with the same language, tools, and models is a great way to improve communication between engineers and reduce the project cost and development time. However, with different disciplines using design models at different project phases, confusion may arise about the contribution of models and what they represent.

An incorrect interpretation of what the models represent can lead to an incorrect use of those models and ultimately impact the quality of the software produced. Users that participate in the definition of MQO have shared many concrete use cases when underspecified models or models with insufficient maturity have been prematurely promoted as “ready for coding”. Consequently, higher development effort than planned, bugs, and difficult conversations related to responsibilities would then take place. To avoid this situation, this document proposes to clarify the role of design models for the development of embedded software and standardize measurable criteria to verify their quality.

This approach has been inspired by the Software Quality Objectives (SQO) [1] defined by a group of automotive actors and MathWorks in 2010, at a time when most exchanges between car manufacturers and suppliers were based on textual specification and manual code. This approach also aims to go one step further in the formalization of model sharing, as defined by Bosch [2] in 2014, and in the implementation of techniques and measures proposed by software safety standards such as ISO26262-6. [3]

1.6 References

Ref	Description
[1]	Patrick Briand (Valeo), Martin Brochet (MathWorks), Thierry Cambois (PSA Peugeot Citroën), Emmanuel Coutenceau (Valeo), Olivier Guetta (Renault SAS), Daniel Mainberte (PSA Peugeot Citroën), Frederic Mondot (Renault SAS), Patrick Munier (MathWorks), Loic Noury (MathWorks), Philippe Spozio (Renault SAS), Frederic Retailleau (Delphi Diesel System), Software Quality Objectives for Source Code, ERTS 2010-Conference, 2010.
[2]	S. Louvet, Robert Bosch (France) SAS, Dr. U. Niebling, Dr. M. Tanimou, Robert Bosch GmbH Model Sharing to leverage customer cooperation in the ECU software development; Toulouse, ERTS 2014-Conference, 2014.
[3]	ISO 26262 International standard for functional safety of electrical and/or electronic systems in production automobiles defined by the International Organization for Standardization (ISO) in 2011.

[4]	RTCA/Eurocae, Software Considerations in Airborne Systems and Equipment Certification, RTCA DO-331 / Eurocae ED-218, December 13, 2011.
[5]	Automotive SPICE Process Assessment / Reference Model from VDA QMC Working Group 13 / Automotive SIG
[6]	Railway applications - Communication, signalling and processing systems - Software for railway control and protection systems, EN 50128:2011
[7]	Hersteller Initiative Software (**HIS**) is an initiative from German automotive manufacturers whose goal is the production of agreed standards within the area of standard software modules for networks, development of process maturity, software test, software tools and programming of ECU's. HIS specifies a fundamental set of Software Metrics to be used in the evaluation of software.
[8]	MathWorks Advisory Board (MAB) Guidelines
[9]	MISRA C - Set of software development guidelines for the C programming language developed by MISRA (Motor Industry Software Reliability Association). Its aims are to facilitate code safety, security, portability and reliability in the context of embedded systems, specifically those systems programmed in ISO C / C90 / C99.
[10]	Newell, A. (1994). Unified theories of cognition. Harvard University Press
[11]	Peugeot Thomas (2014). System Engineering, for a Cognitive Sciences Approach. CSDM 2014

1.7 Terminology

Term	*Definition*
Simulation / Co-Simulation Model	A model developed with MATLAB, Simulink and Stateflow that will not be used in the final design. These might be preliminary models (trade studies, simplified models...) or environmental models for example. While the quality of these models does not generally impact the generated software, they are crucial for documentation, reuse and performance of the overall workflow.
Design Model	A model developed with MATLAB, Simulink and Stateflow to design software architecture and algorithms for signal processing, communication or control software. In the context of MQO, four types of design models are defined: the functional model, the architecture model, the component design model, and the component implementation model.
Model Higher Level Requirement	A requirement satisfied by a design model.
Model-Based Design	A process that systematically relies on the use of models at different phases of the system and software development process.

Model Quality Objective	A quality objective that applies to a model.
Model Quality Requirement	A textual expression that specifies a non-functional requirement of a design model.

1.8 Abbreviations

MAAB	MathWorks Automotive Advisory Board
MBD	Model-Based Design
MQO	Model Quality Objective
MQR	Model quality Requirement
SQO	Software quality Objectives

1.9 Template

The following template is used to specify MQR in section 3.2.

<table>
<tr><td>Requirement ID</td><td>Requirement title</td></tr>
<tr><td>Description</td><td>A description including a measurable satisfaction criterion on model, generated code or executable generated code.</td></tr>
<tr><td>Recommendation level</td><td>Level of recommendation for each model quality objective:
Empty i.e. N/A
Recommended i.e. Recommended for early verification
Mandatory
<table><tr><td>MQO-0</td><td>MQO-1</td><td>MQO-2</td><td>MQO-3</td><td>MQO-4</td></tr><tr><td>XXX</td><td>XXX</td><td>XXX</td><td>XXX</td><td>XXX</td></tr></table></td></tr>
<tr><td>Notes</td><td>Further information to clarify the requirement description.</td></tr>
<tr><td>References / Examples of techniques</td><td>References or examples of techniques to implement the requirement with MATLAB/Simulink.</td></tr>
<tr><td>Rationale</td><td>Justification for quality</td></tr>
<tr><td>Last update</td><td>MQO Version when requirement was last updated</td></tr>
</table>

1.10 Authors

This document was prepared by the MQO working group composed of representatives from MathWorks, automotive OEMs and suppliers.

V1 Main Contributors

Jérôme Bouquet	Renault
Stéphane Faure	Valeo
Florent Fève	Valeo
Matthieu Foucault	PSA
Ursula Garcia	Bosch
François Guérin	MathWorks
Thierry Hubert	PSA
Florian Levy	Renault
Stéphane Louvet	Bosch
Patrick Munier	MathWorks
Pierre-Nicolas Paton	Delphi
Alain Spiewek	Delphi
Yves Touzeau	Renault

V2 Main Contributors

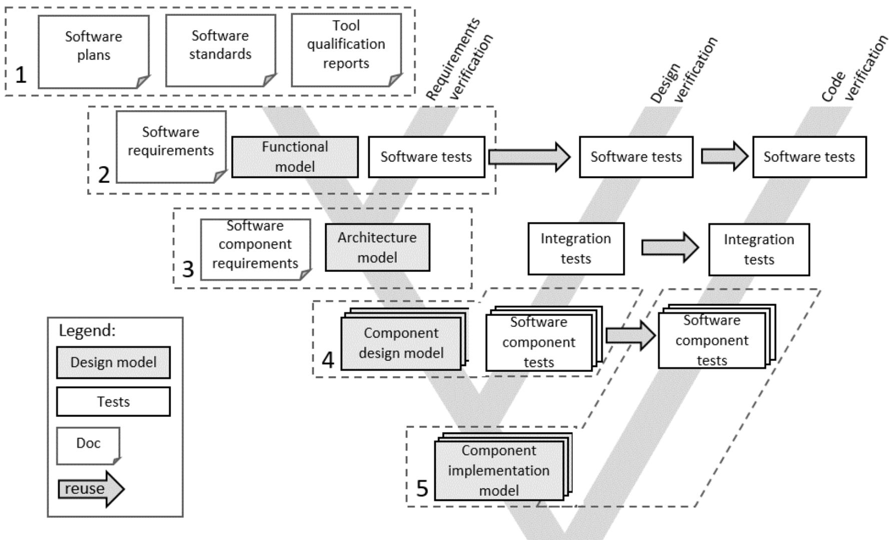
Florent Fève	Valeo
Ursula Garcia	Bosch
Jean Duprez	Airbus
Tahina Bezanahary	Airbus
Stephane Follic	Schneider
Christophe Ducamp	Airbus DS
Jean-Paul Marcade	MathWorks
Ibrahim Saddoug	MathWorks
Laurent Royer	MathWorks

2 Software Development with Design Models

2.1 Overview

This document defines a development approach based on four types of design models supporting the left-hand side of the V-cycle.

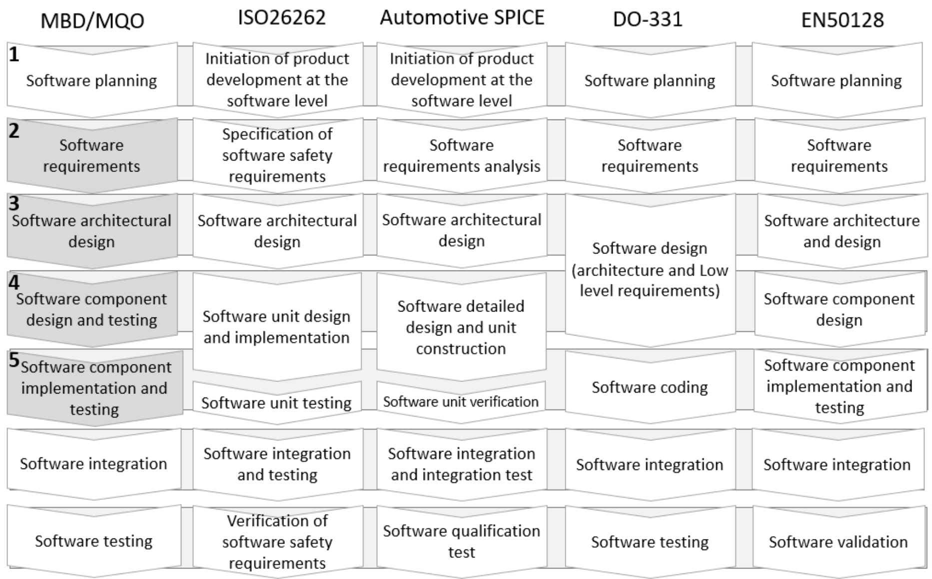
Figure 1: Model-Based Design



The Model-Based Design/MQO software development lifecycle involves five specific phases marked as 1 to 5 in Figure 1 Sections 3.1 to 3.5 will provide greater details on the phases.

Figure 2 shows how the Model-Based Design/MQO software development lifecycle maps to other software development lifecycles from the industry. The phases supported by design models are highlighted with a dark background, and Model-Based Design is referred to as MBD.

Figure 2: Model-Based Design / MQO software phases versus other industry standards [3], [4], [5], [6]



2.2 Software planning phase

This section defines the planning activities that must be carried out to prepare the use of design models. This is recommended for the use of functional models and mandatory for the use of architecture, component design, and component implementation models. Most of these concepts are already imposed by safety standards such as DO-331 [5].

2.2.1 Scope definition

All design models may not be applicable to all projects. For instance, the scope of Model-Based Design can be reduced to the development of a single software component or only used to support the software architectural design specification. The project shall define the software development phases that will be supported by design models. Each design model shall be managed independently as a work product of the software development phase it belongs to.

2.2.2 Tools definition

The tools that support the development and verification of design models shall be identified and classified at the beginning of the project. Those tools shall be qualified, if required by the project.

2.2.3 Standards definition

The modeling standard used to support the development of design models shall be defined prior to entering the software architecture phase. The coding standard used to support the development of design models shall be defined prior to entering the software component implementation phase, or ideally, prior to entering the software component design phase.

2.2.4 MQR identification and allocation

The MBD quality requirements (MQR) defined in 3.2 shall be identified and agreed to by the project stakeholders at the beginning of the project. **Some MQR shall be tailored to the project requirements** (e.g., model coverage or model complexity). Each MQR shall be allocated to a project stakeholder.

2.2.5 Strategy to achieve MQO

Once the MQR has been defined for the project, a strategy shall be defined to achieve the objective. Such a strategy can include intermediate steps corresponding to project milestones, specific training, or a tools migration process. For instance, it is recommended to gradually increase the coverage criteria and not wait for the final version of the software to perform most of the test development effort.

2.2.6 MQR conformance demonstration

The conformance with the project MQR shall be planned and demonstrated at the end of the project. The verification of each MQR shall lead to the production of a report produced by the project stakeholder responsible of the MQR. Sufficient justifications must be provided when MQR are not met (e.g., missing coverage should be justified). The person in charge of assessing the compliance shall have the necessary skills to understand the justifications.

2.3 Software requirements phase

This section focuses on the functional model developed during the software requirement phase.

2.3.1 Roles of the functional model

The role of the functional model is to clarify and refine complex dynamic behaviors that need to be translated into software requirements.

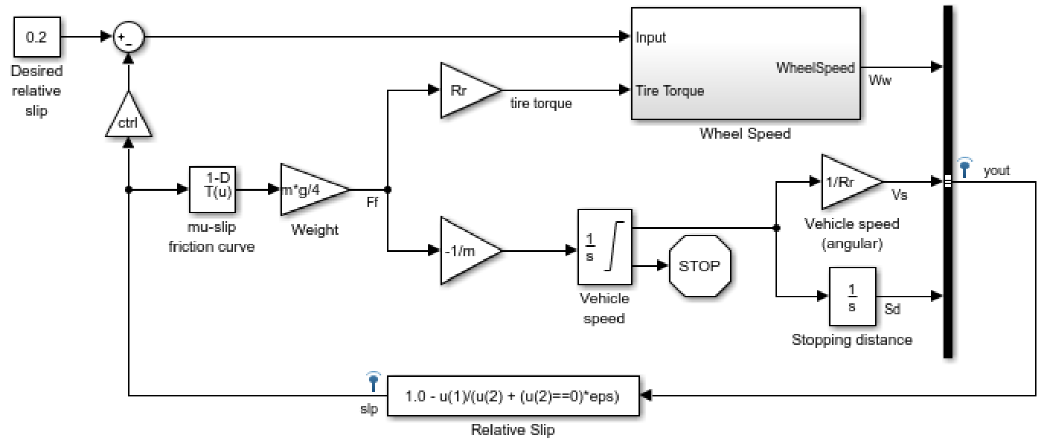
In most cases, the functional model and the software requirements are concurrently developed by the person in charge of the software requirements. This functional model engineer supports the stabilization of the software requirements (the “what”) while identifying good design solutions (the “how”) that could be further elaborated during the design and implementation phases. The functional model is often referred to as an executable specification because it provides a functional behavior that satisfies the functional requirements. However, the functional model does not replace the software functional requirements. The functional model contributes to the validation activities of the software requirements.

2.3.2 Main characteristics of the functional model

The functional model focuses on the correctness of algorithms and equations. It does not have to consider design constraints related to embedded software development. However, when developing the functional model, it should anticipate the main characteristics of the hardware platform and their impact on the software requirements.

The functional model does not have to be representative of all the software functional requirements. Figure 3**Error! Reference source not found.** shows an example of a functional model using continuous time and is limited to a small function of a larger software.

Figure 3: An example of functional model (Anti-Lock Braking system)



2.4 Software architectural design phase

This section focuses on the architecture model developed during the software architectural design phase.

2.4.1 Role of the architecture model

The role of the architecture model is to contribute to the specification of the software architectural design.

Graphical notation is naturally well-suited to defining an organization of components, representing interfaces and connections, and specifying component scheduling. For a complex architecture, it is not conceivable to develop such a diagram without a proper modeling language and a computer-aided design tool such as Simulink.

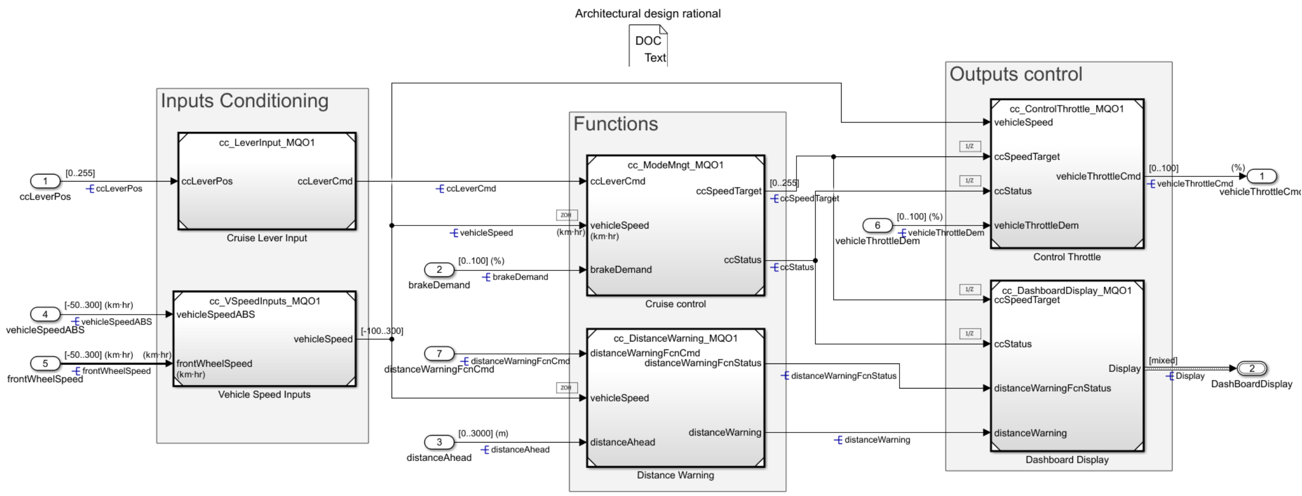
2.4.2 Main characteristics of the architecture model

The architecture model fully specifies the static software architectural design (e.g., component models, interfaces) and provides links/references to the component design models that will be built or are already built. The architecture design model is associated with a data dictionary that defines the data and interfaces of the software and its components.

The architecture model directly contributes to the design activities and is therefore subject to conformance with industry quality standards, safety standards, and/or architecture standards (e.g., traceability to requirements, compatibility with architecture standard).

The next figure shows an example of an architecture model that references component models represented by model references.

Figure 4: Example of architecture model



2.5 Software component design and testing phase

This section focuses on the component design model developed during the software component design and testing phase.

2.5.1 Role of the component design model

The role of the component design model is to provide a complete specification of the software component design and support its verification with dynamic and static analysis.

The use of a high-level modeling and programming language enables better management of the complexity of algorithms and reduces the probability of design errors. The support of simulation and static analysis contributes to elimination of design errors.

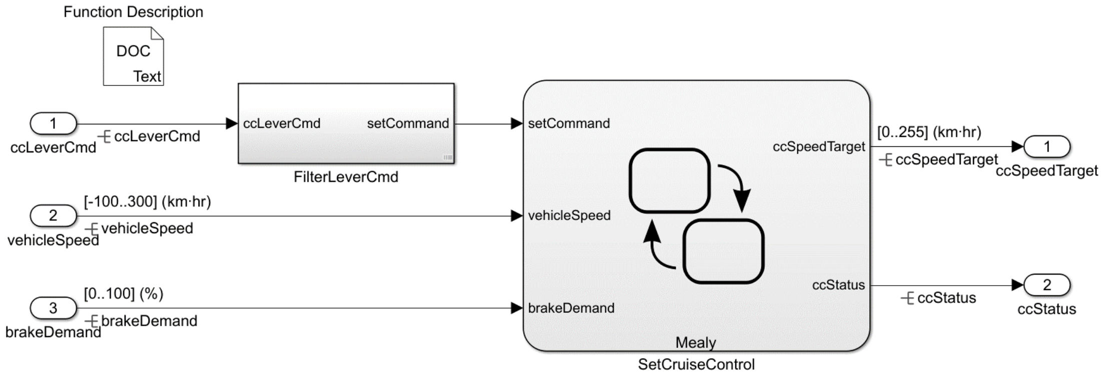
2.5.2 Main characteristics of the component design model

The component design model fully specifies the algorithms and equations that will be part of the embedded software and excludes any elements used for debugging or prototyping such as measurement points or override mechanisms. Each component design model is associated with a data dictionary that defines its interface, parameters, and monitored signals.

The component model directly contributes to the development activities and is therefore subject to conformance with industry quality standards, safety standards, and/or design standards (e.g., conformance to modeling standard, traceability to requirements).

Figure 5 shows an example of a component design model with fully defined interfaces and sub-functions implemented with state machines.

Figure 5: Example of component design model



2.6 Software component implementation and testing phase

This section focuses on the component implementation model developed during the software component design and testing phase.

2.6.1 Role of the component implementation model

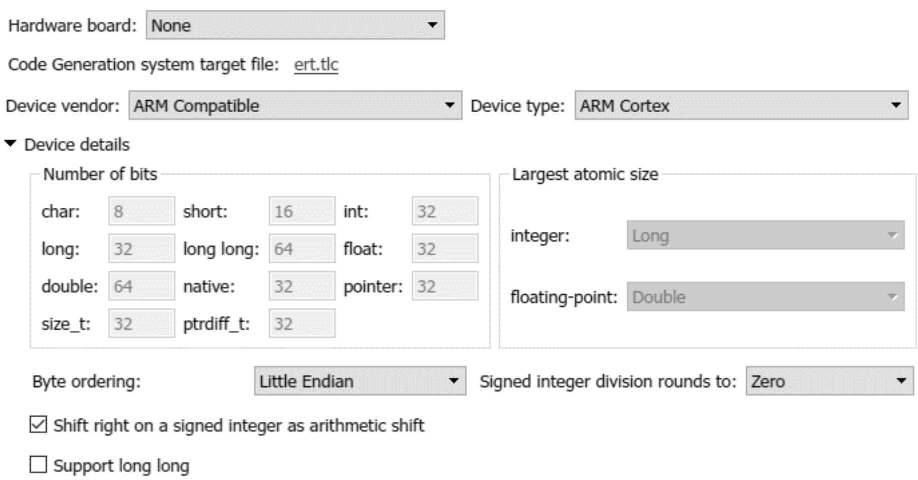
The role of the component implementation model is to enable the generation of production code for a specific embedded target and basic software.

2.6.2 Characteristics of the component implementation model

The component implementation model fully specifies the software component implementation. Implementation details are added to the data dictionary to refine how to represent parameters and signals in the target memory. Code configuration options and customization are defined to integrate the generated code with specific basic software functions, so they match the target characteristics (e.g., byte ordering) and satisfy the component code memory footprint and execution performance requirements allocated to the software component.

The generated code of the component implementation model directly contributes to the development activities and is therefore subject to conformance with the industry quality standard, safety standard, and/or coding standard (e.g., MISRA C® [9]). Each component implementation model is associated with a data dictionary that defines its interface parameters and monitored signals. Figure 6 shows the target hardware configuration of Embedded Coder.

Figure 6: Example of code generation configuration for the component implementation model



2.7 Simulation / co-simulation models

This section focuses on the models that are not used to generate the final software but that are produced / used in the design phase. These could be for example environmental models, test scenarios or simplified models used for trade studies.

2.7.1 Role of the simulation / co-simulation models

The role of these models is specific to each project. One could argue that they shouldn't be monitored as they do not directly impact the generated software. But they can have purposes that put them in the critical path:

- Performance when co-simulating / testing
- Reused in other projects
- Documentation

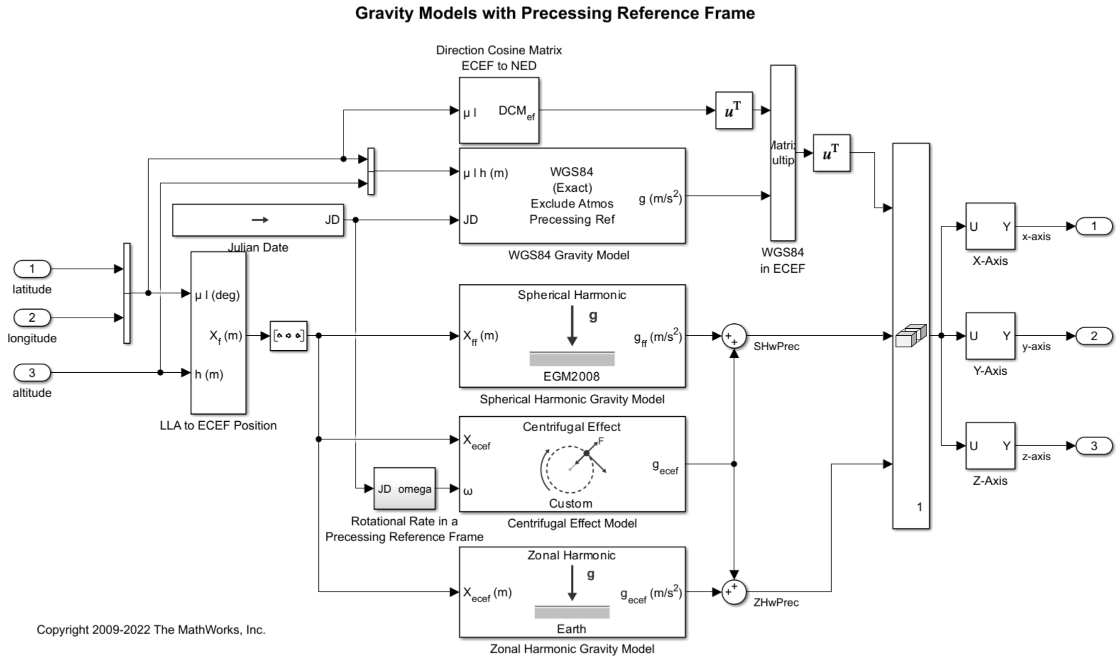
2.7.2 Characteristics of the simulation / co-simulation models

Generic simulation / co-simulation model can have different goals, hence different characteristics. A common characteristic is often the level of fidelity. Developing highly representative models is not always a productive idea as these models take longer to develop, and usually longer to simulate.

A good simulation / co-simulation model is the one that help you answer a specific question in a reasonable amount of time. Sometimes a first order model is enough (system level trade studies), sometimes a detailed one is necessary (transient behavior of a system).

If a high-fidelity model exists, it can sometimes be more efficient to use it as a baseline to develop a new simplified model that answers the question instead of reusing it directly.

Figure 7: Model used to compare different gravity models



2.8 Relationship between design models

Each design model shall be independently managed as a work product of the software development phase in which it belongs. At the same time, design models can share design information and shall be consistent. For instance, the component design model in Figure 5 share its interface definition with the architecture model of The next figure shows an example of an architecture model that references component models represented by model references.

Figure 4Whenever consistency is required, reuse is encouraged.

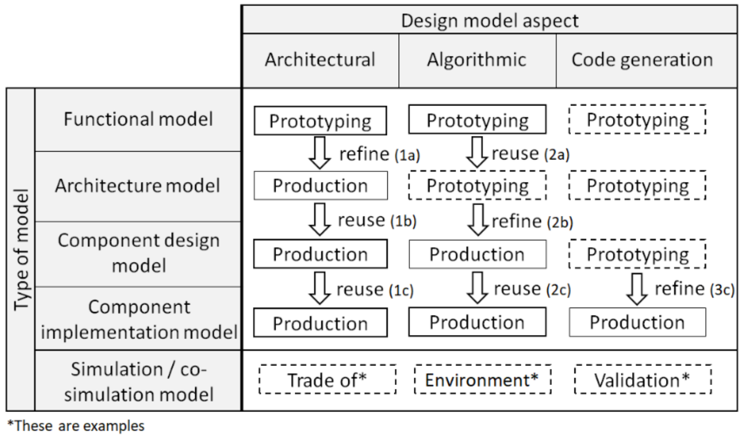
Figure 8 indicates which aspects can be reused between design models (“reuse” arrow). It also provides guidance on which aspects of design models can be partially reused to accelerate development (“refine” arrow). The arrows on Figure 8 can apply to the following modeling aspects of design models:

- Architectural aspect: interface, scheduling, partitioning, intercomponent control and data flow, etc.
- Algorithmic aspect: mathematical calculation, component control and data flow, state machine, truth table, etc.
- Code generation aspect: memory management, data access, function prototype, code optimization, etc.

The design models differ from each other’s by the level of maturity and importance of the different modeling aspects described above. Figure 8indicates the levels of maturity and importance based on the following definitions and representations:

- Maturity level: High (Production) / Low (Prototyping)
- Importance level: Mandatory (plain line) / Recommended (dotted line)

Figure 8: Design model relationships and contribution to prototyping and production development



The functional model shall have structured algorithms that can contribute to the validation of the software requirements with modeling and simulation. A model's code generation configuration for rapid prototyping can be useful to validate the software requirements with a real-time environment. The development focus shall be on the software requirement (not represented on the figure). The entire model shall be considered a prototype.

The architecture model shall define the component interface and scheduling of the software architectural design. The architectural design aspect of the functional model can serve as a baseline to initiate the development of the software architecture for production (1a). The prototype algorithms of the functional model can populate the architecture model to enable early dynamic verification of the model in simulation to evaluate the impact of the architecture on the functional behavior (2a). A prototype code generation configuration representative of the software architecture standard (e.g., AUTOSAR) can be created to achieve early verification of the impact of the functional behavior in real time and its integration with software and hardware (e.g., AUTOSAR RTE).

The component design model shall fully define the software component design with its structure, scheduling, and algorithms. The interface of the model shall be consistent with, and can be reused from, the architecture model (1b). The prototype algorithms developed for the functional model can serve as a baseline to define the production algorithms (2b). A prototype code generation configuration can be used for early verification of the non-trivial impacts of the design model on the generated code (e.g., compliance with the coding standard, level of code coverage versus model coverage, code expansion).

The component implementation model shall define both the software component design and implementation. The structure, scheduling, and algorithms shall be reused from the software component design model (1c, 2c). The code generation configuration shall be used for production code generation and shall then be compatible with the software coding standard and the target hardware.

Simulation / co-simulation models do not appear on this figure at it focuses on the design process.

3 Models Quality

3.1 Overview

As design models are critical for development using Model-Based Design, their quality must be carefully assessed. Models can automatically transform into other design artifacts such as documentation, source code, or executables. Therefore, the quality objectives defined on the models shall impact the models themselves as well as their derived products. A specific quality objective is defined for each type of design model to account for their specific role.

Table 1: Model Quality Objectives of models

Design model name	Quality Objective
Simulation / Co-Simulation Model	MQO-0
Software Functional Model	MQO-1
Software Architecture Model	MQO-2
Software Component Design Model	MQO-3
Software Component Implementation Model	MQO-4

Table 2 below provides the list of Model Quality Requirement (MQR) applicable to achieve the quality objective of each type of model. The details of each MQR are specified in section 3.2.

Table 2: Overview of Model Quality Requirements of MQOs

MQR ID	MQR Title	MQO-0	MQO-1	MQO-2	MQO-3	MQO-4
MQR-01	Model layout	M	M	M	M	M
MQR-02	Model comments	R	M	M	M	M
MQR-03	Model links to requirements		M	M	M	M
MQR-04	Model testing against requirements		M	R	M	M
MQR-05	Model compliance with modeling standard	M		M	M	M
MQR-06	Model data	M		M	M	M
MQR-07	Model size				M	M
MQR-08	Model complexity				M	M
MQR-09	Model coverage				M	M
MQR-10	Model robustness				M	M
MQR-11	Generated code testing against requirements				R	M
MQR-12	Generated code compliance with coding standard				R	M
MQR-13	Generated code coverage				R	M
MQR-14	Generated code robustness				R	M
MQR-15	Generated code execution time					M
MQR-16	Generated code memory footprint					M
MQR-17	Model repository	M	M	M	M	M
MQR-18	Artefacts management	M	M	M	M	M

MQR-19	Graphical complexity	R	M	R	R	R
MQR-20	Model Data Storage	M	M	M	M	M
MQR-21	Clone Detection	M	M	M	M	M

M: Mandatory
R: Recommended

3.2 Model Quality Requirements

3.2.1 Model layout

MQR-01	Model layout
Description	The model needs to be readable on common screens / paper sizes. This is equivalent to a Simulink canvas of 2000 x 1500 pixels.
Recommendation level	<table><tr><td>MQO-0</td><td>MQO-1</td><td>MQO-2</td><td>MQO-3</td><td>MQO-4</td></tr><tr><td>Mandatory</td><td>Mandatory</td><td>Mandatory</td><td>Mandatory</td><td>Mandatory</td></tr></table>
Notes	The 2000 x 1500 pixels canvas is equivalent to an A4 paper (or letter size paper) printed with ~80% zoom ratio and can be tailored.
References / Examples of techniques	- Simulink subsystems - Stateflow sub-charts - Simulink bus
Rationale	Printing a Simulink model can be necessary to archive or share models as documents. Splitting models into several sheets for printing makes them difficult to read A model diagram that can be completely displayed on screen improves readability and eases model review. Reducing the size of the diagrams forces the model developer to better organize large model and data into hierarchical structures of buses and model references or subsystems.
Last update	2.0

3.2.2 Model comments

MQR-02	Model comments
Description	The model comments shall provide a description of the model itself and the following types of elements: - Simulink subsystem - Simulink function and S-function mask - Stateflow chart, sub-chart, truth table, state transition table, and flowchart - Simulink and MATLAB function blocks and sub-functions
Recommendation level	<table><tr><td>MQO-0</td><td>MQO-1</td><td>MQO-2</td><td>MQO-3</td><td>MQO-4</td></tr><tr><td>Recommended</td><td>Mandatory</td><td>Mandatory</td><td>Mandatory</td><td>Mandatory</td></tr></table>
Notes	A comment can include a mix of text, equations, diagrams, and pictures. A comment can be embedded in the model or a link can be established from the model to a separate and accessible document. The quality of the comments is not in the scope of this requirement and shall be assessed by peers during the model review.
References / Examples of	- Insertion of blocks for documentation - Description in Simulink subsystems masks - Stateflow diagrams annotations

techniques	- Comments in Simulink and MATLAB function codes
Rationale	Like code, a model without comments is harder to understand by peers. Lack of description can negatively impact the efficiency of the peer review activity and maintenance activities.
Last update	2.0

3.2.3 Model links to requirements

<table>
<tr><td>MQR-03</td><td>Model links to requirements</td></tr>
<tr><td>Description</td><td>The model elements that specify algorithms and calculations shall trace to the model higher level requirements.

The design model elements that specify interface shall trace to the software interface requirements or software component interface requirements.</td></tr>
<tr><td>Recommendation level</td><td><table><tr><td>MQO-0</td><td>MQO-1</td><td>MQO-2</td><td>MQO-3</td><td>MQO-4</td></tr><tr><td></td><td>Mandatory</td><td>Mandatory</td><td>Mandatory</td><td>Mandatory</td></tr></table></td></tr>
<tr><td>Notes</td><td>A model element is implicitly traced to a model higher level requirement if one of its parents is traced (e.g., its parent subsystem).

The model shall trace to the right level of requirements:
- Functional model and architecture model shall trace to software requirements
- Component design model and component implementation model shall trace to software component requirements
The correctness of the links to model higher level requirements is not in the scope of this requirement and shall be assessed by peers during the model review.

When model references are used inside component design and implementation models, each referenced model shall trace to its own model higher level requirements.</td></tr>
<tr><td>References / Examples of techniques</td><td>- Bidirectional links between model and requirement tool</td></tr>
<tr><td>Rationale</td><td>Traceability to requirements eases static model verification against requirements. It facilitates:
- Requirement coverage analysis
- Impact analysis on design following changes on requirements
- Identification of unintended or useless design to be present in the model</td></tr>
<tr><td>Last update</td><td>2.0</td></tr>
</table>

3.2.4 Model testing against requirements

MQR-04	Model testing against requirements
Description	The model shall produce the expected outputs when exercised by tests derived from and traced to the model higher level requirements.

<table>
<tr><td>Recommendation level</td><td><table><tr><td>MQO-0</td><td>MQO-1</td><td>MQO-2</td><td>MQO-3</td><td>MQO-4</td></tr><tr><td></td><td>Mandatory</td><td>Recommended</td><td>Mandatory</td><td>Mandatory</td></tr></table></td></tr>
<tr><td>Notes</td><td>This is often called Model In the Loop testing (MIL).

The model tests shall be derived from and traced to all model higher level requirements which verification strategy is testing.

Each test shall have a defined procedure, stimuli, and expected outputs.

The model test environment shall not impact the behavior of the model under test.

The correctness of the tests and links to model higher level requirements are not in the scope of this requirement and shall be assessed by peers during the tests review.</td></tr>
<tr><td>References / Examples of techniques</td><td>- Formalized requirements can help deriving design and test without breaking the digital continuity
- Stimuli and expected outputs time series
- Test sequences and test oracles
- Automation of test procedure, execution, and reporting</td></tr>
<tr><td>Rationale</td><td>The simulation of the design model enables the discovery of design errors at design time. It can also contribute to refining model higher level requirements or correcting and validating requirement-based tests.</td></tr>
<tr><td>Last update</td><td>2.0</td></tr>
</table>

3.2.5 Model compliance with modeling standard

<table>
<tr><td>MQR-05</td><td>Model compliance with modeling standard</td></tr>
<tr><td>Description</td><td>The model shall be compliant with the modeling standard.</td></tr>
<tr><td>Recommendation level</td><td><table><tr><td>MQO-0</td><td>MQO-1</td><td>MQO-2</td><td>MQO-3</td><td>MQO-4</td></tr><tr><td>Mandatory</td><td></td><td>Mandatory</td><td>Mandatory</td><td>Mandatory</td></tr></table></td></tr>
<tr><td>Notes</td><td>The modeling standard(s) shall be defined during the project software planning.

Model compilation warnings and errors reported by Simulink diagnostics are considered modeling standard violations.

Additional workflows might require additional standard compliance.</td></tr>
<tr><td>References / Examples of techniques</td><td>- MathWorks Advisory Board (MAB) Guidelines[8] is a great starting point
- Company Wide standard
- MathWorks modeling guidelines for high-integrity systems
(HIS - Include compatibility with MISRA C® compliance)</td></tr>
<tr><td>Rationale</td><td>The model standard can enforce best practices and define a subset of the modeling language that limits the possibility of incorrect use of the language.</td></tr>
<tr><td>Last update</td><td>2.0</td></tr>
</table>

3.2.6 Model data

MQR-06	Model data
Description	The model I/O signals, calibrations, parameters and test points shall be fully defined. The required properties depend on the model objectives but might include the following: • Name • Description • Design min/max • Initial value (output only) • Data type (e.g., base type, fixed-point type, enumerated type, structured type) • Size • Physical unit • Safety integrity level • Memory storage
Recommendation level	MQO-0: Mandatory; MQO-1: ; MQO-2: Mandatory; MQO-3: Mandatory; MQO-4: Mandatory
Notes	The compute method is necessary for data coming from external software, driver, or communication network. An initial value or safe value can be added for output and safety critical data. Memory storage only needs to be defined in the component implementation model. Display format for measured signal and calibration for floating point is recommended.
Examples of techniques	- Simulink data objects - Simulink data dictionary
Rationale	Model data are part of the design and need to be fully defined. For instance, incorrect or unknown data integrity level or data design min/max can impact the model and software reliability and robustness.
Last update	2.0

Recommendation level:

MQO-0	MQO-1	MQO-2	MQO-3	MQO-4
Mandatory		Mandatory	Mandatory	Mandatory

3.2.7 Model size

MQR-07	Model size
Description	The model size shall be limited. This can either be a limit in terms of number of elements: - The number of Simulink blocks - The number of MATLAB executable lines of codes - The number of Stateflow transition, states, and connections - The number of truth tables decision Or a limit in terms of depth.
Recommendation level	MQO-0: Recommended; MQO-1: ; MQO-2: ; MQO-3: Mandatory; MQO-4: Mandatory
Notes	For the number of blocks, an empirical limit can be set to 500. For the depth, we recommend 7 [10] [11].

Recommendation level:

MQO-0	MQO-1	MQO-2	MQO-3	MQO-4
Recommended			Mandatory	Mandatory

	The model reference block & subsystem reference block only counts as one element. The company standard utility function (e.g., Simulink library block, MATLAB function file) only counts as one element. Please refer to MathWorks guidance on large-scale modeling in Simulink documentation.
References / Examples of techniques	Model references, subsystem references, libraries, ..
Rationale	Very large models are more difficult to merge and are more likely to be modified by several users at the same time. Smaller models are more likely to be reusable and easily configurable. Generated code of very large models cannot be incrementally tested.
Last update	2.0

3.2.8 Model complexity

MQR-08	Model complexity
Description	The model and its subsystems, Stateflow charts, and MATLAB functions shall have a limited local model complexity .
Recommendation level	MQO-0: – ; MQO-1: – ; MQO-2: – ; MQO-3: Mandatory ; MQO-4: Mandatory
Notes	Local complexity is the complexity for objects at their hierarchical level. Aggregated complexity is the cyclomatic complexity of an object and its descendants. This document does not impose a threshold as it should be adapted to each company / project based on experience and habits. This being said, if we try to adapt the HIS [7] code metric to Model-Based Design, we can recommend 15 for MATLAB Code, 30 for state charts and 20 for other models as a starting point.
References / Examples of techniques	Model complexity is a measure of the structural complexity of a model. It approximates the McCabe cyclomatic complexity measure for code generated from the model. There's only a loose correlation between model complexity and the cyclomatic complexity of the generated code. This being said, model complexity is a good indicator of model health.
Rationale	Cyclomatic complexity is a leading testability metric.
Last update	2.0

3.2.9 Model coverage

MQR-09	Model coverage
Description	The model structure shall be fully covered by the test suite that is derived from and traced to the model higher level requirements.
Recommendation level	<table><tr><th>MQO-0</th><th>MQO-1</th><th>MQO-2</th><th>MQO-3</th><th>MQO-4</th></tr><tr><td></td><td>Recommended</td><td>Recommended</td><td>Mandatory</td><td>Mandatory</td></tr></table>
Notes	The structural coverage criteria chosen shall be at least conformant to the structural coverage criteria imposed by the software safety integrity level. For MQO -3 & MQO-4, we recommend a 100% coverage. For MQO-1 & MQO-2, we recommend not necessarily to enforce a coverage ratio but rather to at least share/display the coverage to stakeholders so that they can build trust in the model or analyze the gap.
References / Examples of techniques	Types of coverage analysis available on Simulink model: - Execution Coverage (EC) - Decision Coverage (DC) - Condition Coverage (CC) - Modified Condition/Decision Coverage (MCDC) EC, DC, CC, MCDC, saturation on integer overflow coverage, and relational boundary coverage can be used to measure the model structural coverage.
Rationale	Model coverage enables to identify untested design, untestable design, or unintended design.
Last update	2.0

3.2.10 Model robustness

MQR-10	Model robustness
Description	The model shall be robust in normal and abnormal operating conditions.
Recommendation level	<table><tr><th>MQO-0</th><th>MQO-1</th><th>MQO-2</th><th>MQO-3</th><th>MQO-4</th></tr><tr><td></td><td></td><td></td><td>Mandatory</td><td>Mandatory</td></tr></table>
Notes	In normal operating condition, inputs and tunable parameters values are within their design ranges. In abnormal operating condition, inputs, and tunable parameters values are outside their design ranges. Robustness shall prevent errors such as:

	- Divisions by zero - Integer overflows - Out of design range - Out of bound array The level of robustness shall be compliant with the software safety integrity level.
References / Examples of techniques	- Test generation based on relational boundary coverage - Formally-based verification technique with abstract interpretation - Defensive programming
Rationale	Model robustness verification prevents edge case or incorrect use of model, which can cause unexpected results or simulation errors.
Last update	2.0

3.2.11 Generated code testing against requirements

MQR-11	Generated code testing against requirements
Description	The model generated code shall produce the expected outputs when exercised by tests derived from and traced to the model higher level requirements
Recommendation level	MQO-0: – ; MQO-1: – ; MQO-2: – ; MQO-3: Recommended ; MQO-4: Mandatory
Notes	For MQO-03, tests can be run in software-in-the-loop. For MQO-04, tests shall be run in processor-in-the-loop. A representative hardware or an emulator can be used in place of the actual processor.
References / Examples of Techniques	- Test reuse from component design model testing - Test generation for back-to-back testing
Rationale	Code testing is required to verify the output of the code generator and compiler or cross-compiler, linker, load, and flash utilities. For MQO-3, code testing in software-in-the-loop increases confidence in the code generator.
Last update	2.0

Recommendation level (MQR-11):

MQO-0	MQO-1	MQO-2	MQO-3	MQO-4
			Recommended	Mandatory

3.2.12 Generated code compliance with coding standard

MQR-12	Generated code standard compliance
Description	The generated code shall be compliant with the coding standard.
Recommendation level	(see below)

MQO-0	MQO-1	MQO-2	MQO-3	MQO-4
			Recommended	Mandatory

Notes	The coding standard shall be defined during the project software planning phase and shall be compatible with the software safety standard, software architecture standard, and targeted hardware (e.g., floating-point support). The modeling standard shall anticipate the compliance with the coding standard. The project coding standard can be tailored for generated code.
References / Examples of techniques	- MISRA C 2012 - CERT C
Rationale	Coding standard verification is required to verify the output of the code generator.
Last update	2.0

3.2.13 Generated code coverage

MQR-13	Generated code coverage
Description	The model generated code structure shall be fully covered by all the tests that are derived from and traced to the model higher level requirements.
Recommendation level	MQO-0: – ; MQO-1: – ; MQO-2: – ; MQO-3: Recommended ; MQO-4: Mandatory
Notes	The structural coverage criteria shall be at least conformant to the structure coverage criteria imposed by the software safety integrity level. The model tests shall be reused to cover the structure of the generated code. The code coverage can be different than the model coverage depending on the blocks used (e.g., look-up table interpolation algorithm) or code generation optimization options (e.g., for loop unrolling).
References / Examples of techniques	Types of coverage analysis available on the generated code: - Statement Coverage for Code Coverage - Condition Coverage for Code Coverage - Decision Coverage for Code Coverage - Modified Condition/Decision Coverage (MCDC) for Code Coverage
Rationale	Code coverage is required in addition to model coverage to verify that the code generator do not add unintended functionalities.
Last update	2.0

3.2.14 Generated code robustness

MQR-14	Generated code robustness
Description	The model generated code shall be robust in normal and abnormal operating conditions.

Recommendation level	MQO-0: / MQO-1: / MQO-2: / MQO-3: Recommended / MQO-4: Mandatory
Notes	In normal operating condition, inputs and tunable parameter values are within their design ranges. In abnormal operating condition, inputs and tunable parameter values are outside their design ranges. Robustness shall prevent errors such as: - Divisions by zero - Integer overflows - Out of design range - Out of bound array The level of robustness shall be compliant with the software safety integrity level.
References / Examples of techniques	- Test generation based on relational boundary coverage - Formally-based verification technique with abstract interpretation - Defensive programming
Rationale	Code robustness verification is required to verify the output of the code generator
Last update	2.0

3.2.15 Generated code execution time

MQR-15	Generated code execution time
Description	The model generated code running on the production target shall be instrumented to measure and verify the execution time.
Recommendation level	MQO-0: / MQO-1: / MQO-2: / MQO-3: / MQO-4: Mandatory
Notes	Worst case execution time shall be specified during software architectural design phase. The execution time shall include the generated code and its calling functions (e.g., basic software services). The production target can be an emulator or a representative hardware. The model tests can be reused on the generated code running on the production target (aka processor-in-the-loop) and the expected outputs shall still be obtained.
References / Examples of techniques	- Profiling in processor-in-the-loop from Simulink
Rationale	The component software execution time shall be measured prior the component integration to verify compatibility with architecture requirements, avoid shortage of hardware resource, and enable reuse of component on different architecture.
Last update	2.0

3.2.16 Generated code memory footprint

MQR-16	Generated code memory footprint
Description	The model generated code memory footprint shall be measured and verified.
Recommendation level	MQO-0: – ; MQO-1: – ; MQO-2: – ; MQO-3: – ; MQO-4: Mandatory
Notes	Memory footprint, such as RAM, ROM, and stack, shall be specified during software architectural design phase. The memory footprint shall include the generated code and its calling functions.
References / Examples of techniques	- Stack estimation tool - Code generator metrics
Rationale	The component software memory footprint shall be measured prior the component integration to verify compatibility with architecture requirements, avoid shortage of hardware resource, and enable reuse of component on different architecture.
Last update	2.0

Recommendation level:

MQO-0	MQO-1	MQO-2	MQO-3	MQO-4
				Mandatory

3.2.17 Model repository

MQR-17	Model repository
Description	Each model shall be documented appropriately. In addition to technical documentation, the model artefacts need to include information related to: • The developer / maintainer • The compatible software versions • The required dependencies • The lifecycle of the model • The validation activities that have been performed • The objectives of the model • The applicability of the model
Recommendation level	MQO-0: Mandatory ; MQO-1: Mandatory ; MQO-2: Mandatory ; MQO-3: Mandatory ; MQO-4: Mandatory
Notes	This information can be stored as a separate artefact. We also recommend using similar repository structures across projects to simplify reuse & integration.
References / Examples of techniques	- Model Identity Card [1](MIC) - Simulation Model Meta Data (SMMD)

Recommendation level:

MQO-0	MQO-1	MQO-2	MQO-3	MQO-4
Mandatory	Mandatory	Mandatory	Mandatory	Mandatory

[1] https://mic.irt-systemx.fr/mic

Rationale	This information is required to facilitate model reuse across an organization. The lifecycle of a model is a duration during which the model is considered to be relevant. Extending this lifecycle requires a model audit.
Last update	2.0

3.2.18 Artefacts management

MQR-18	Artefacts management
Description	A model should be part of a project that gathers all the corresponding artefacts (data, documentation, metadata, validation artefacts, ..). The lifecycle of the project shall be managed
Recommendation level	MQO-0: Mandatory; MQO-1: Mandatory; MQO-2: Mandatory; MQO-3: Mandatory; MQO-4: Mandatory
Notes	MATLAB Projects and the Git integration can be used for this purpose.
References / Examples of techniques	- MATLAB Projects - Git / SVN
Rationale	Most of the models are not standalone files but a set of files. Having a project gathering all the artefacts eases model sharing, configuration management as well as maintainability. As a project can contain different types of models, files can be tagged to apply specific MQO to specific models. A MATLAB project usually corresponds to a Git project.
Last update	2.0

3.2.19 Graphical complexity

MQR-19	Graphical complexity
Description	A given layer of a model should have from 3 to 10 concepts (7 being optimal). The concepts being the number of inputs/outputs and the number of components.
Recommendation level	MQO-0: Recommended; MQO-1: Mandatory; MQO-2: Recommended; MQO-3: Recommended; MQO-4: Recommended
Notes	
References / Examples of techniques	
Rationale	In the “Unified theories of cognition” [10][11], Adam Newell states that the brain cannot manipulate more than 7 ideas simultaneously. While this number might not be the same for

	all of us, limiting the number of concepts manipulated in a given layer of a model helps designers, reviewers and other stakeholders to better understand them. “Concepts” is defined here as something that brings information to the user and has a semantic meaning. SysML v2 concepts is a non-exhaustive list that can be used as a starting point (Parts, Attributes, Ports, States, Connections, ..)
Last update	2.0

3.2.20 Model data storage

MQR-20	Model Data Storage
Description	Model data storage needs to be anticipated during the planning phase. Each referenced model needs to have its own data dictionary. The base workspace shall not be used. Callbacks functions creating data should be avoided.
Recommendation level	MQO-0: Mandatory; MQO-1: Mandatory; MQO-2: Mandatory; MQO-3: Mandatory; MQO-4: Mandatory
Notes	The base workspace should not be used as it’s not protected against data collision. The model workspace can be used but it does not allow you to store all the information (e.g., busses), hence the recommendation to use data dictionaries.
References / Examples of Techniques	- Simulink data dictionaries - Model workspace
Rationale	Data management is often a late concern. Data collisions issues are extremely hard to debug and separating the data into several files later is also cumbersome.
Last update	2.0

3.2.21 Design pattern duplication

MQR-21	Avoid design pattern duplication.
Description	Design pattern duplication (clones and similar clones) shall be detected and replaced to avoid duplicated data.
Recommendation level	MQO-0: Mandatory; MQO-1: Mandatory; MQO-2: Mandatory; MQO-3: Mandatory; MQO-4: Mandatory
Notes	As a rule, copy & paste should be avoided and replaced by proper componentization techniques (referenced models, referenced subsystems, libraries, variants). When 2 components only differ by their parametrization, similar techniques can still be applied
References / Examples of	- Clone detector is useful to detect and replace clones and similar clones

Techniques	
Rationale	Clones and similar clones grow the model size and introduce potential robustness issues.
Last update	2.0