

# Model Quality Objectives for embedded software development with MATLAB and Simulink

Jérôme Bouquet, Stéphane Faure, Florent Fève, Matthieu Foucault, Ursula Garcia, François Guérin, Thierry Hubert, Florian Levy, Stéphane Louvet, Patrick Munier, et al.

## ▶ To cite this version:

Jérôme Bouquet, Stéphane Faure, Florent Fève, Matthieu Foucault, Ursula Garcia, et al.. Model Quality Objectives for embedded software development with MATLAB and Simulink. 9th European Congress on Embedded Real Time Software and Systems (ERTS 2018), Jan 2018, Toulouse, France. hal-02156122

## HAL Id: hal-02156122 https://hal.science/hal-02156122

Submitted on 18 Jun 2019

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

## Model Quality Objectives for Embedded Software Development with MATLAB and Simulink

Jérôme Bouquet (Renault), Stéphane Faure (Valeo), Florent Fève (Valeo), Matthieu Foucault (PSA Peugeot Citroën), Ursula Garcia (Robert Bosch), François Guérin (MathWorks), Thierry Hubert (PSA Peugeot Citroën), Florian Levy (Renault), Stéphane Louvet (Robert Bosch), Patrick Munier (MathWorks), Pierre-Nicolas Paton (Delphi Technologies), Alain Spiewek (Delphi Technologies)

**Abstract:** This paper presents standard quality objectives for models developed with Simulink<sup>®</sup> 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<sup>®</sup>, Simulink, and Polyspace<sup>®</sup> products. The purpose of this standard is to clarify and ease the collaboration between OEM and suppliers when sharing Simulink models in the context of embedded software development to drive the production of higher quality and integrity software.

This paper first defines a software development approach based on four types of design models used at four different phases of the software development lifecycle. Then, a specific quality objective, named MQO, is proposed for each type of model. Each objective is defined as a set of quality characteristics with some measurable criteria named Model Quality Requirement (MQR). Some additional guidelines are provided to manage the planning and quality assessment activities related to MQO and MQR. This paper concludes with some expected impact on the adoption of MQO by the automotive industry.

#### **1** 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. OEM and tier-one suppliers 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. In order 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]

#### 2 Purpose and Scope

The purpose of this paper is a) to define the main use cases of design models for software development and b) to define a standard and generic approach to assess the quality of models depending on their use cases.

### **3** Software Development with Design Models

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





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 / MQC	software ph	ases versus
	other industry	y standards [3]	, [4], [5], [6]	
MBD/MOO	ISO26262	Automotive SPICE	DO-331	EN50128

NIDD/ NIQO			00 001	ENSOIZE
1 Software planning	Initiation of product development at the software level	Initiation of product development at the software level	Software planning	Software planning
2 Software requirements	Specification of software safety requirements	Software requirements analysis	Software requirements	Software requirements
<b>3</b> Software architectural design	Software architectural design	Software architectural design	Software design	Software architecture and design
4 Software component design and testing	Software unit design and implementation	Software detailed design and unit	level requirements)	Software component design
5 <sub>Software</sub> component implementation and testing	Software unit testing	Software unit verification	Software coding	Software component implementation and testing
Software integration	Software integration and testing	Software integration and integration test	Software integration	Software integration
Software testing	Verification of software safety requirements	Software qualification test	Software testing	Software validation

## **3.1 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].

<u>Scope definition</u>: 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.

<u>Tools definition</u>: 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.

<u>Standards definition</u>: 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.

<u>MQR identification and allocation</u>: The MQR shall be identified and agreed to by the project stakeholders at the beginning of the project. Some MQR shall be adapted to the project requirements (e.g. model and code coverage criteria). Each MQR shall be allocated to a project stakeholder. <u>Strategy to achieve MQR</u>: 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.

<u>MQR conformance demonstration</u>: 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.

### **3.2 Software Requirements Phase**

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

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 may not be needed if the software functional requirements are simple to implement, nor does it have to be representative of all the software functional requirements. Figure 3 shows an example of a functional model using continuous time and is limited to a small function of a larger software.



Figure 3. Example of a functional model (anti-lock braking

#### 3.3 Software Architectural Design Phase

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

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).

Figure 4. Example of architecture model.



#### 3.4 Software Component Design and Testing Phase

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

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.



**3.5 Software Component Implementation and Testing Phase** This section focuses on the component implementation model developed during the software component design and testing phase. The role of the component implementation model is to enable the generation of production code for a specific embedded target and basic software.

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<sup>®</sup>). Each component implementation model is associated with a data dictionary that defines its interface parameters and monitored signals.

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

Hardware bo	ard: No	ne			-						
Code Genera	tion syste	em target fil	e: ert.tle	2							
Device vendo	r: ARM	Compatible			•	Devic	e type:	ARM	Cortex		-
<ul> <li>Device detail</li> </ul>	ails										
Number	of bits						Largest	atomic	size		
char:	8	short:	16	int:	32						
long:	32	long long:	64	float:	32		integer:		Long		~
double:	64	native:	32	pointer:	32		floating-	point:	Double		v
size_t:	32	ptrdiff_t:	32				5				
Byte orde	ring:	l	.ittle Endi	ian	Ŧ	Sign	ied integ	er divi	sion rounds to:	Zero	¥

Shift right on a signed integer as arithmetic shift

Support long long

#### 3.6 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 shares its interface definition with the architecture model of Figure 4. Whenever consistency is required, reuse is encouraged.

Figure 7 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 7 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 by the level of maturity and importance of the different modeling aspects described above. Figure 7 indicates the levels of maturity and importance based on the following definitions and representations:

- Maturity level: high (Production) / low (Prototyping)
- Importance level: mandatory (solid line) / recommended (dotted line)

Figure 7. Design model relationships and contribution to
prototyping and production development.

		Design model aspect				
		Architectural	Algorithmic	Code generation		
del	Functional model	Prototyping	Prototyping	Prototyping		
Architecture model		Production	Prototyping	Prototyping		
e of de:	Component design model	Production	Production	Prototyping		
م الم imp	Component implementation model	Production	c) U reuse (2 Production	Production		

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 way algorithms are implemented can be adapted to address non-functional requirements (e.g. optimization, safety). 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.

### 4 Design Model Quality

#### 4.1 Overview

As design models are critical for software development using Model-Based Design, their quality must be carefully assessed. Design models can automatically transform into other design artifacts such as documentation, source code, or executables. Therefore, the quality objectives defined on the design 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.

Software development phase	Type of design model	Model Quality Objective
Software requirements phase	Functional model	MQO-1
Software architectural design phase	Architecture model	MQO-2
Software component design and testing phase	Component design model	MQO-3
Software component implementation and testing phase	Component implementation model	MQO-4

Table 1. Model Quality Objectives of design models.

Table 2 provides the list of Model Quality Requirements applicable to achieve the quality objective of each type of design model.

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

Table 2. Overview of Model Quality Requirements of MQOs.

M: Mandatory

R: Recommended for early verification

Note: An additional MQR to verify the generated source code against the model can be required in the context of DO-331.

## 4.2 Model Quality Requirements

This section provides further details on the MQR introduced in Table 2.

MQR-01	Model layout						
Description	The model shall define Simulink and Stateflow <sup>®</sup> diagrams that are completely visible on A4 paper size.						
Recommendation	MQO-1	MQO-1 MQO-2 MQO-3 MQO-4					
level	Mandatory	Mandatory	Mandatory	Mandatory			
Notes	Fit to view with a zoom ratio	smaller than 80% is hard	er to read on screen and lik	kely not to be readable on			
	A4 paper size.						
	The model zoom ratio is visit	ole at the center of the mo	del status bar below the di	agram.			
References /	- Simulink subsystem	8					
Examples of	- Stateflow sub-charts						
techniques	- Simulink bus						
Rationale	Printing a Simulink model ca	n be necessary to archive	or share models as docum	ents.			
	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 buse	es and model references o	r subsystems.				

MOD 02	36.11					
MQR-02	Model comments					
Description	The model comments shall provide a description of the model itself and the following types of elements:					
-	<ul> <li>Simulink subsyste</li> </ul>	m				
	- Simulink function	and S-function mask				
	- Stateflow chart su	h_chart_truth_table_state_tra	ansition table and flowcha	rt		
	Statellow chart, su	ELAD for ation blooks and	and for ations	Γι –		
	- Simulink and MA	I LAB function blocks and s	sub-functions			
Recommendation	MQO-1	MQO-1 MQO-2 MQO-3 MQO-4				
level	Mandatory Mandatory Mandatory Mandatory					
Notes	A comment can include a m	nix of text, equations, diagra	ams, and pictures.			
	A comment can be embedd	ed in the model or a link can	n be established from the n	nodel to a separate and		
	accessible document.			-		
	The quality of the comment	s is not in the scope of this	requirement and shall be a	ssessed by peers during		
	the model review.					
References /	- Insertion of blocks for documentation					
Examples of	- Description in Simulink subsystems masks					
techniques	- Stateflow diagram	s annotations				
	- Comments in Sim	ilink and MATLAB function	on 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.

MOR-03	Model links to requirements						
Description	The model elements that specify algorithms and calculations shall trace to the model higher level						
Description	requirements	eny algorithms and calcula	ations shall trace to the mo	del lligher level			
	The design model elements that specify interface shall trace to the software interface requirements or						
	The design model elements that specify interface shall trace to the software interface requirements or						
D	software component interfac	e requirements.					
Recommendation	MQO-1 MQO-2 MQO-3 MQO-4						
level	Mandatory	Mandatory	Mandatory	Mandatory			
Notes	A model element is implicitl	y traced to a model higher	level requirements if one of	of its parents is traced			
	(e.g. its parent subsystem).						
	The model shall trace to the	right level of requirements	:				
	- Functional model as	nd architecture model shall	l trace to software requiren	nents			
	- Component design	model and component imp	lementation model shall tra	ace to software			
	component requirer	nents					
	The correctness of the links	o model higher level requi	rements is not in the scope	e of this requirement and			
	shall be assessed by peers du	ring the model review.	L	1			
	When model references are u	used inside component desi	ign and implementation mo	odels, each referenced			
	model shall trace to its own	nodel higher level require	ments.	· · · · · · · · · · · · · · · · · · ·			
References /	- Bidirectional links	petween model and require	ement tool				
Examples of							
techniques							
Rationale	Traceability to requirements	eases static model verifica	tion against requirements	It facilitates:			
radonalo	- Requirement covers	ore analysis	anon against requirements.	it inclinates.			
	Impact analysis on	design following changes	on requirements				
	- Impact analysis of	intended on veglage design	to be present in the model				
	- Identification of uni	intended or useless design	to be present in the model				

1000.04		•			
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 require	ments.	-		
Recommendation	MQO-1 MQO-2 MQO-3 MQO-4				
level	Mandatory	Recommended	Mandatory	Mandatory	
Notes	The model tests shall be de	erived from and traced to all	l model higher level requ	irements which verification	
	strategy is testing.				
	Each test shall have a defin	ned procedure, stimuli, and	expected outputs.		
	The model test environme	nt shall not impact the beha	vior of the model under to	est.	
	The correctness of the test	s and links to model higher	level requirements are no	ot in the scope of this	
	requirement and shall be a	ssessed by peers during the	tests review.	-	
References /	- Stimuli and exped	cted outputs time series			
Examples of	- Test sequences an	nd test oracles			
techniques	- Automation of test procedure, execution, and reporting				
Rationale	The simulation of the design model enables the discovery of design errors at design time. It can also				
	contribute to refining mod	el higher level requirements	s or correcting and validation	ting requirement-based	
	tests.	_	-		

MQR-05	Model compliance with modeling standard				
Description	The model shall be compli	ant with the modeling stand	lard.		
Recommendation	MQO-1	MQO-2	MQO-3	MQO-4	
level		Mandatory	Mandatory	Mandatory	
Notes	The modeling standard shall be defined during the project software planning phase and shall be compatible with the software safety standard, software design standard, coding standard, and targeted hardware (e.g. floating-point support). Model compilation warnings and errors reported by Simulink diagnostics are considered modeling standard violations. The modeling standard could be adapted to software architectural design modeling and software component design modeling				
References /	- MathWorks modeling guidelines for high-integrity systems				
Examples of	(Include compatil	bility with MISRA C® comp	pliance)		

techniques	<ul> <li>MathWorks Automotive Advisory Board Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow</li> </ul>
Rationale	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

MQR-06	Model data					
Description	The model I/O signals, calibrations, and observable signals shall be fully defined with the following properties: <ul> <li>Name</li> <li>Description</li> <li>Design min/max</li> <li>Initial value (output only)</li> <li>Data type (e.g. base type, fixed-point type, enumerated type, structured type)</li> <li>Size</li> <li>Physical unit</li> <li>Safety integrity level</li> <li>Memory storage</li> </ul>					
Recommendation	MQO-1	MQO-2	MQO-3	MQO-4		
level		Mandatory	Mandatory	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	- Simulink data objects					
techniques	- Simulink data dic	- Simulink data dictionary				
Rationale	Model data are part of the	design and need to be fully	defined. For instance, inc	correct or unknown data		
	integrity level or data desig	gn min/max can impact the	model and software relia	bility and robustness.		

MOP 07	Model size				
MQK-07					
Description	The model shall have less th	The model shall have less than 500 elements including:			
	- The number of Simulin	ık blocks			
	- The number of MATLA	AB executable lines of cod	les		
	- The number of Stateflo	w transition, states, and co	onnections		
	- The number of truth tab	oles decision			
Recommendation	MQO-1	MQO-2	MQO-3	MQO-4	
level			Mandatory	Mandatory	
Notes	The model reference block of	only counts as one element	•		
	The company standard utilit	y function (e.g. Simulink l	ibrary 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					
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	e models cannot be increm	entally tested		
	Generated code of very large	e models camot de meren	ienturry tested.		

MQR-08	Model complexity				
Description	The model and its subsystems, Stateflow charts, and MATLAB functions shall have a local cyclomatic complexity lower or equal to "30".				
Recommendation	MQO-1 MQO-2 MQO-3 MQO-4				
level	Mandatory Mandatory				
Notes	Local complexity is the cyclomatic complexity for objects at their hierarchical level.				
	Aggregated cyclomatic con	nplexity is the cyclomatic co	omplexity of an object ar	nd its descendants.	

	The threshold of 30 for local cyclomatic complexity is a recommendation and can be adapted on a project basis. The number 30 for cyclomatic complexity has been derived from the HIS (Hersteller Initiative
	Software) code metric and adapted to Model-Based Design.
References /	Cyclomatic complexity is a measure of the structural complexity of a model. It approximates the McCabe
Examples of	complexity measure for code generated from the model. The McCabe complexity measure is slightly
techniques	higher on the generated code due to error checks that the model coverage analysis does not consider.
	To compute the cyclomatic complexity of an object, such as a block, chart, or state, model coverage uses
	the following formula:
	$c = \sum_{1}^{N} (o_n - 1)$
	N is the number of decision points that the object represents and $o_n$ is the number of outcomes for the <i>n</i> th
	decision point. The tool adds one to the complexity number for atomic subsystems and Stateflow charts.
Rationale	Cyclomatic complexity is a leading testability metric. Test harness can be created for simulation at model,
	subsystem, chart, and MATLAB function level.

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				
	nigher level requirements.				
Recommendation			1		
level	MQO-1	MQO-2	MQO-3	MQO-4	
			Mandatory	Mandatory	
Notes	The structural coverage criteri	a chosen shall be at leas	st conformant to the structura	al coverage criteria	
	imposed by the software safety	y integrity level.			
References /	Types of coverage analysis available on Simulink model:				
Examples of	- Execution Coverage (EC)				
techniques	- 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 ide	entify untested design, u	intestable design, or unintend	ded design.	

MQR-10	Model robustness					
Description	The model shall be robust in	The model shall be robust in normal and abnormal operating conditions.				
Recommendation level	MQO-1 MQO-2 MQO-3 MQO-4 Mandatory Mandatory					
			¥ 1	· · · ·		
Notes	In normal operating condition	n, inputs and tunable para	meters values are within the	eir design ranges.		
	In abnormal operating condition	ion, inputs, and tunable p	arameters values are outside	e 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 /	- Test generation based on relational boundary coverage					
Examples of	- Formally-based verification technique with abstract interpretation					
techniques	- Defensive programming					
Rationale	Model robustness verification	n prevents edge case or in	correct use of model, which	h can cause unexpected		
	results or simulation errors.	- •		*		

MQR-11	Generated code testing aga	inst requirements				
Description	The model generated code traced to the model higher	shall produce the expected level requirements	outputs when exercised	by tests derived from and		
Recommendation	MQO-1	MQO-2	MQO-3	MQO-4		
level	Recommended 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	<ul> <li>Test reuse from component design model testing</li> <li>Test generation for back-to-back testing</li> </ul>
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.

MQR-12	Generated code standard compliance				
Description	The generated code shall be co	ompliant with the codi	ing standard.		
Recommendation	MQO-1	MQO-2	MQO-3	MQO-4	
level			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 /	- MISRA C 2012 for safety				
Examples of	- CERT C for cyber security				
techniques					
Rationale	Coding standard verification i	s required to verify the	e output of the code generat	tor.	

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 requ	irements.			
Recommendation	MQO-1	MQO-2	MQO-3	MQO-4	
level			Recommended	Mandatory	
Notes	The structural coverage criter	ia shall be at least cont	formant to the structure cover	age criteria imposed by	
	the software safety integrity le	evel.			
	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 /	Types of coverage analysis available on the generated code:				
Examples of	- Statement Coverage for Code Coverage				
techniques	- 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.		_		

MQR-14	Generated code robustness						
Description	The model generated code shall be robust in normal and abnormal operating conditions.						
Recommendation	MQO-1	MQO-2	MQO-3	MQO-4			
level			Recommended	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						
	<ul> <li>Integer overflows</li> <li>Out of design range</li> <li>Out of bound array</li> </ul>						
	The level of robustness shall be compliant with the software safety integrity level.						
References /	- Test generation based on relational boundary coverage						
Examples of	- Formally-based verification technique with abstract interpretation						
techniques	- Defensive programming						
Rationale	Code robustness verification is required to verify the output of the code generator						

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	MQO-1	MQO-2	MQO-3	MQO-4		
level				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 so 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 process					
	the-loop) and the expected outputs shall still be obtained.					
References /	- Profiling in processor-in-the-loop from Simulink					
Examples of						
techniques						
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.					

MQR-16	Generated code memory footprint					
Description	The model generated code memory footprint shall be measured and verified.					
Recommendation	MQO-1	MQO-2	MQO-3	MQO-4		
level				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 /	- Stack estimation tool	l				
Examples of						
techniques						
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.					

## 5 Conclusion

This paper clarifies how Simulink design models contribute to accelerate development and verification activities from software requirements specification to software implementation. Four types of design models with specific purposes have been 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 in order 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) Clear 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

## **6** References

[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 - International Organization for Standardization. ISO 26262 Road vehicles Functional Safety Part 1-10, 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 3.0 / 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