DOI: 10.13009/EUCASS2023-910
Aerospace Europe Conference 2023 – 10TH EUCASS – 9TH CEAS

Classification of Flight Control Law Requirements in the Context of Model-Based Software Development and Verification

*Alexander Arnold** †, Robert Luckner* and Guido Weber***
** Technische Universität Berlin, Institut für Luft- und Raumfahrttechnik*
Marchstr. 12-14, 10587 Berlin, Germany
*** Liebherr-Aerospace Lindenberg GmbH*
Pfänderstraße 50-52, 88161 Lindenberg im Allgäu, Germany

alexander.arnold@tu-berlin.de† – robert.luckner@tu-berlin.de – guido.weber@liebherr.com

Abstract
This paper describes a classification scheme for flight control law requirements to support the model-based verification in an RTCA DO-331 compliant software development process. The flight control law requirements are divided into requirement classes according to suitable verification methods. Individual classes are examined regarding possible automation for generating test cases. The characteristics of all requirement classes are described, and verification methods are demonstrated on examples from a representative industrial application. The results and the benefits of the method are discussed.

1. Introduction

Flight control systems (FCS) are developed from system requirements. The system requirements allocated to software (SRATS) are derived from these system requirements for the Flight Control Law (FCL) software design and for the verification processes. As an FCS is complex and safety-critical, its development process must adhere to strict guidelines in a certification context, such as ARP 4754A [1]. The software development aspects are covered by DO-178C [2] and its supplements. The processes for FCL development in the aviation industry are costly as they require significant resources of highly-trained engineers and qualified tools.

Model-based controller design is common industry practice today. In the MODULAR project, sponsored by the German Federal Ministry for Economic Affairs and Climate Action, one of TU Berlin's tasks addresses efficiency improvements in the model-based development process of flight control software. Software code generation from FCL design models can be automated with qualified code generators like SCADE Suite KCG. A DO-331 [3] compliant development process for model-based FCL software development was defined in MODULAR based on experience from previous research projects at the Department of Flight Mechanics, Flight Control and Aeroelasticity. This paper focusses on the verification process and its optimisation.

The first step towards more automation is to examine the methods for requirement-based verification. The FCL requirements are divided into seven requirement classes based on suitable verification means: requirements on control law structure, criteria for control law design, input filtering and command shaping, open-loop behaviour, logic circuits, output transformation, and filter design.

The classification scheme is based on the primary flight control laws of a regional jet aircraft that are representative for state-of-the-art industrial flight control laws. The FCL functions provide vertical, lateral load-factor and roll rate demand laws for manual flight with decoupling of roll, pitch, and yaw motion, automatic pitch trim, envelope protections (load factor, stall, overspeed, low-energy, pitch angle, bank angle, tail strike protection, asymmetric engine thrust, rudder deflection), gust load alleviation, degraded operation in case of non-critical sensor failures and the synchronisation between three redundant flight control computers.

The proposed requirement classification can improve the quality of verification by assigning adequate verification methods as soon as a requirement is written. Furthermore, it provides a foundation to plan verification activities, to assign responsibilities and to identify verification activities that are suitable for automation.

DOI: 10.13009/EUCASS2023-910

In this paper, the characteristics of all defined requirement classes are described and methods are selected that are suited for verification to take advantage of the model-based development process. The use of the classification scheme is demonstrated on examples from a representative industrial application. The results and the benefits of the method are discussed.

1.1 Model-Based Software Development Process

In MODULAR, a DO-331 compliant development process has been established. Figure 1 shows the MODULAR development process. It is characterised by a single set of requirements, the FCL requirements, that are used for the model-based design of the control laws on the system level and as software high-level requirements (HLR), according to DO-331 MB Example 5. The FCL requirements are derived from analysis of higher-level aircraft and system specifications. The so-called Prototype Model is derived from the FCL requirements. The Prototype Model and a flight mechanical model (FMM) that simulates the rigid-body flight dynamics are integrated into an aircraft flight simulation for early validation of the FCL requirements.

The Prototype Model is automatically translated into the FCL Software Design Model (DM), which represents the software (SW) architecture and low-level requirements[1]. Both models and the generated FCL source code are functionally identical. To show compliance with DO-331 objectives, the verification activities for Design Model and the code generation are supported by qualified tools of the Ansys SCADE Suite.

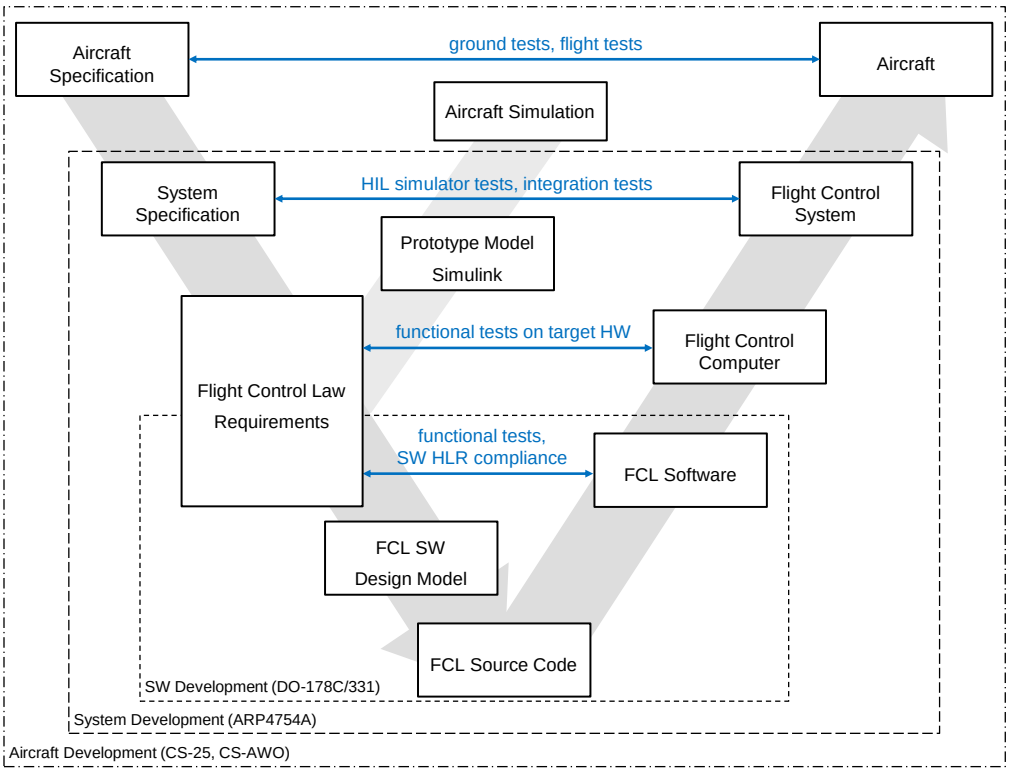

Figure 1: Model-based V development process

While mature tools for the model-based design activities are available, automation of the verification activities is a topic of ongoing research. Three steps toward the overarching objective of automating verification activities are explored in this paper. First, the FCL requirements are categorised to identify requirement types that lend themselves to automated verification methods. Secondly, the functional identity of the Prototype Model in Simulink, the FCL SW Design Model and the FCL Source Code enables the reuse of verification artefacts of the system-level controller development for FCL SW tests. Thirdly, expressing the FCL requirements themselves as specification models enables automated generation of test cases.

[1] DO-178C definition: low-level requirements are software requirements from which source code can be directly implemented without further information.

DOI: 10.13009/EUCASS2023-910

1.2 Related Work

The classification of functional requirements in this paper is focussed on safety-critical, real-time embedded systems. ARP 4754A [1] contains a classification of requirements by type. There are four main types for system requirements: safety requirements, functional requirements, additional certification requirements and derived requirements.
The safety process is a specific focus of the development process outlined in ARP 4754A, and the system safety analysis outlined in ARP 4761 [4], as expressed by its own requirement type. Traceability between requirements, the design, source code to the executable object code is another major objective of development assurance. Derived requirements cannot be traced directly to higher-level requirements and must be fed into the system safety analysis process. The functional requirements of the system are subdivided into: customer requirements, operational requirements, performance requirements, physical and installation requirements, maintainability requirements and interface requirements.

Generic examples for the requirement type attribute are given in the ISO/IEC/IEEE international standard on systems and software requirements engineering [5]: Functional requirements describe functions or tasks to be performed by the system. Performance requirements supplement a functional requirement in terms of performance expressed quantitatively. Interface requirements define how a system interacts with external systems, or how system elements interact. These three types are the focus in MODULAR.
Process requirements and quality requirements are covered in detail in the guidance for safety-critical software in ARP 4754A and DO-178C/331. Process Requirements in the context of safety-critical airborne software arise from the certification specifications (CS-25 for large aeroplanes [6]) and acceptable means of compliance set by the certification authorities, and the aforementioned industry standards.
The last two example types given in [5] are usability/quality-in-use requirements and human factor requirements. The former provides the basis for the design and evaluation of systems to meet the user needs. The latter contains characteristics for the outcomes of interaction with human users (and other stakeholders) in terms of safety, performance, effectiveness, efficiency, reliability, maintainability, heath, well-being and satisfaction. These categories are not further explored in this paper.

The MODULAR classification aims for a more detailed decomposition of functional, performance and interface requirements. The external interface is described in the interface control document (ICD) of the FCL SW. Internal interfaces that exist between the signals consumed and command signals computed by the FCL algorithm and the FCL SW interface are divided into two requirement types. The FCL requirements classification is motivated by the overarching goal of streamlining the planning of verification activities and automating test generation where possible.

Souyris et. al. [7] discuss research activities at Airbus into the use of formal methods in a DO-178 context. Formal verification methods have been used for parts of critical embedded software of commercial aircraft [8]. Airbus employs so called “unit proofs” to prove that the executable object code complies with functional properties defined in the design phase. This replaces low-level unit testing in the classical V-development process. The formal verification is applied to small “units” of software, especially logic functions.
This is echoed by more recent work at GE Aviation during their development of their proprietary tool ASSERT (Analysis of Semantic Specifications and efficient generation of Requirements-based Tests) based on satisfiability modulo theories formulas. Moitra et. al. [9] conclude that “Requirements that work well with ASSERT are functional requirements that use Boolean logic, simple algorithmic expressions, and simple timing (or state machines). Requirements that contain complex control laws, complex algorithms, or complex timing may be better suited for modelling in a different environment.”
Automatic generation of DO-178 test procedures from test cases using observers in SCADE was explored by a team at Liebherr-Aerospace Toulouse [10]. Model-based specification for basic aircraft behaviour with observers in Modelica was explored by Kuhn et. al. [11], who cover frequency domain specifications.

2. FCL Requirements Classification

The FCL SW is an application running on a flight control computer (FCC), which is part of the digital Flight Control System (FCS). Extraneous system software that does not affect the FCL control loop directly is not considered here, such as device drivers, input/output communication and consolidation, system monitoring, or self-test and maintenance functions. Only an overall system delay is included in the model of the flight control/aircraft loop.

DOI: 10.13009/EUCASS2023-910

The requirements classification is based on a set of requirements for the Normal Mode FCL, which provide attitude control for manual flight of a regional jet. The FCL SW functions comprise the discrete flight control law algorithm at its core, which provides tracking of reference signals (vertical and lateral load factors, roll rate, pitch rate, pitch angle and bank angle) in combination with flight envelope protections (angle-of-attack, high-speed, low-energy, extreme attitudes including tail-strike) based on the pilot commands. The control laws adapt to the configuration and the flight condition of the aircraft via gain scheduling.

Upstream of the FCL algorithm, the SW inputs are suitably transformed to form reference and feedback signals of the control laws. This ranges from unit conversions, limitation of value ranges, dead zones, and look-up tables to reconfiguration for fault isolation in case of failed sensor signals to provide a graceful degradation of the FCL in abnormal conditions.
Downstream of the FCL algorithm, the surface commands are allocated to the individual control surfaces and transformed to actuator position commands. Furthermore, a host of status information is provided to the FCS and other aircraft systems, such as the high-lift and engine control systems.
Throughout the FCL SW, internal mode logics activate individual functions, such as automatic lift dump upon touchdown, tail-strike protection, gust load alleviation or the envelope protection functions. The FCL SW runs on three redundant primary flight control computers, which require a synchronisation mechanism to compensate of asynchronous effects for logical switches and integrator states. Each FCC is a duplex system with an asynchronous control and monitor lane.

2.1 Requirement Classes

Classification of HLR regarding verification methods, results in the following seven requirement types:
Requirements on Open-Loop Behaviour exclusively specify the behaviour of FCL output signals in response to FCL input signals. This direct relationship can be verified at the FCL SW interface without a closed-loop simulation. The requirement type is characterised by prescribing certain feed-forward behaviour or design decisions in detail, e.g.

REQ-1: Upon touchdown, the FCL shall command all spoilers to extend fully.

REQ-2: The three FCCs shall synchronize their Ground Mode statuses by majority voting.

Requirements on Control Law Structure specify the functional architecture, signal flow and modularization. Typically, closed-loop behaviour of the system or aircraft is specified and accompanied by design criteria (non-functional requirements) that specify the required performance.

REQ-3: The turn compensation function shall compensate for the incremental load factor $\Delta n_z = 1/\cos\phi$ required to maintain a steady turn up to bank angles of ± 45°.

These requirements can be directly verified through reviews of the design, i.e. by manually checking the load factor command computation contain a provision for the turn compensation using the bank angle in the case of REQ-3. Preferably, the verification of the associated design criteria suffices.

Design Criteria for Control Laws specify required characteristics of the aircraft dynamics, i.e. closed-loop behaviour. These requirements are the basis for control law design – on the system level. Compliance is demonstrated in tests with closed-loop simulations, e.g.

REQ-4: During a manual horizontal steady turn without disturbances, the turn compensation function shall limit the deviation in altitude to ± 100 m.

For testing, the aircraft dynamics are modelled by a flight mechanical model (FMM) that computes the aircraft state and generates the respective sensor signals, e.g. roll rate or pitch angle. The FMM is coupled with the FCL software to simulate the flight control/aircraft control loop. Figure 2 shows the two relevant control loops for the FCL design. The outer loop involves a pilot or pilot model. In MODULAR, the requirements were established for the inner flight control/aircraft loop. Test vectors at the FCL SW input (1) and expected outputs (2) are extracted from closed-loop simulations, which have shown – on system level – that the closed-loop behaviour (3) complies with the requirements. For SW tests, the input test vectors are used for stimulation of the FCL SW and the output vectors define the expected output values. System-level tests are the main source for tests that address model (and inherently code) coverage.

DOI: 10.13009/EUCASS2023-910

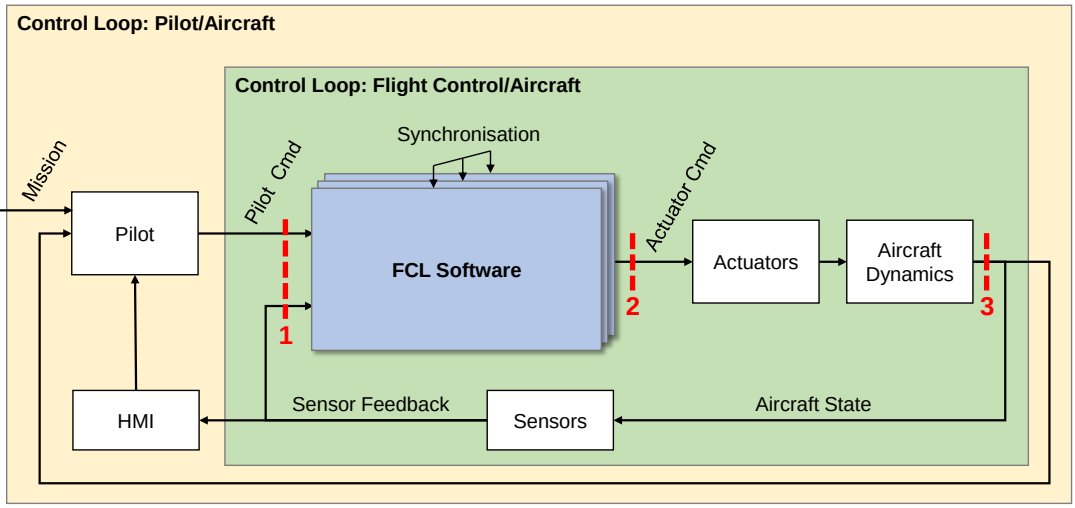

Figure 2: Flight control loops with simplified FCL SW inputs (1) outputs (2) and aircraft state (3)

Requirements on Logic Circuits specify a condition under which a certain action shall be taken, such as the activation or deactivation of a function. The condition consists of relational statements, logic statements including temporal logic, and state machines, e.g.

REQ-5: The FCL shall activate Ground Mode if both sides of the main landing gear are compressed AND the filtered calibrated airspeed is below 60 kt persists for 0.1 s.

Logic circuits are well suited for automation of test generation to achieve the specified code coverage such as modified decision and condition coverage (MCDC) for Level A software. Defining expected values for the FCL SW output may be challenging if the logical state, e.g. “Ground Mode is active” is not visible at the SW interface.

Requirements on Input Processing and Command Shaping specify the transformation of input signals to internal command signals. The transformation may involve simple unit conversions, scaling, saturation limits, or look-up tables. Reconfiguration of faulty inputs and sensor data consolidation may be included in this category if these functions are not implemented separately from the FCL software.

REQ-6: The FCL shall calculate the incremental load factor command as a function of the sidestick pitch command as shown in Figure 3.

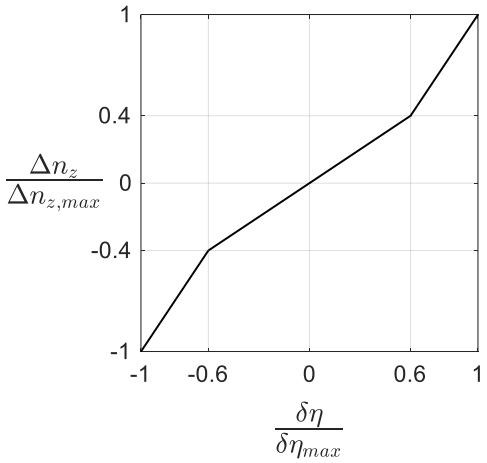

Figure 3: Look-up table for incremental load factor command as a function of sidestick pitch command

These requirements represent the link between the controlled aircraft states (internal commands) and the (inceptor) commands at the FCL SW input. The requirements are necessary to establish expected values for controlled aircraft states when designing closed-loop test cases. Stand-alone verification of the requirement is complicated if the command is not directly visible as an output at the FCL SW interface.

DOI: 10.13009/EUCASS2023-910

Requirements on Output Transformation specify rate limits, saturation limits and the transformation of internal signals to output signals, e.g. transformation of internal aileron roll command to left and right aileron commands.

REQ-7: The FCL shall limit the elevator deflection command rate to ± 30°/s.

REQ-8: The FCL shall scale the spoiler deflection command by 50% for the inboard spoiler pair, 70% for the centre spoiler pair and 100% for the outboard spoiler pair.

The existence of correctly configured limiter and rate limiter functions can be checked during model review. Critical output limits, such as rudder deflection limitation, have to be tested. They can be stimulated by appropriate input signals. However, stimulation of the output limitation from the FCL SW interface may be non-trivial, especially if internal control loops are involved upstream of the limiter.

Requirements on Filter Design specify responses to stimuli in the frequency domain. The requirements are used to design linear SISO filters such as low-pass filters for sensor signals.

REQ-9: The FCL shall low-pass filter the measured calibrated airspeed with the cut-off frequency of 2 rad/s and a high-frequency roll-off of -20 db/decade.

The requirements of filter design are verified individually as elements of a model element library. In MODULAR, the FCLib model element library [12] is used, which contains commonly used filters. The verification task on the FCL SW level is then reduced to reviewing the correct use (integration) and configuration of the model library element instances.

2.2 Application to Flight Control Laws

The classification scheme was applied to a different set of requirements for the automatic flight control laws of a HALE UAV for validation. Table 1 shows the split of the requirements by class for each set of requirements. The large number of control law structure requirements in the AFCL requirements set is an indication of the higher degree of automation. In contrast to the HALE UAV AFCL, the outer loop guidance and navigation control functions are not part of the regional jet Normal Mode FCL.
The relatively high number of requirements in the open-loop behaviour class for the regional jet NM FCL is related to the higher number of visible internal status signals that are synchronised between the three FCCs. The relatively low number of design criteria requirements result from the fact that no requirements for the NM FCL had been available. As far as possible, the requirements were re-engineered from the FCL design in MODULAR.

Table 1: FCL requirements by requirement class

Requirement Class	**Regional Jet NM**	**HALE UAV AFCL**
Control law structure	36	126
Design criterion	16	75
Logic circuit	15	89
Open-loop behaviour	51	18
Input filtering and command shaping	12	}68[a]
Output transformation	11	
Filter design	15	0[b]
Total	156	386

[a] combined class for interfacing requirements
[b] requirements class not used

The responsibilities between system and software engineers follows from requirements classification and associated verification methods. Table 2 shows the preferred verification method for each requirement class. Verification of requirements on control law structure and design criteria belong to the controller design on the system level with closed-loop simulations. Requirements on open-loop behaviour are verified with low effort at the FCL software interface. Requirements on logic circuits are well suited for automated test case generation at the FCL SW interface if the stimulation can be derived from requirements on input filtering and command shaping and the expected values can be derived from requirements on output transformation.

DOI: 10.13009/EUCASS2023-910

Table 2: Preferred verification method (**X**) for requirement classes

Requirement Class	Model Review	Test at Software Interface	Closed-loop Simulation
Control law structure	**X**	-	-
Design criterion	-	-	**X**
Logic circuit	x	**X**	x
Open-loop behaviour	x	**X**	x[a]
Input filtering and command shaping	x	x[b]	**X**
Output transformation	x	**X**	x[c]
Filter design	**X**[d]	x[b]	-

[a] higher effort than test at SW interface
[b] viability is highly dependent on signal paths in Design Model
[c] stimulation may be non-trivial
[d] frequency domain analysis of isolated library element

3. Verification Examples

The MODULAR FCL verification process is structured in three phases:

Phase 1: Non-formal[2] verification of the FCL SW based on simulations in Simulink that are performed in the system development process. The development of Test Cases and Test Procedures is based on SRATS. It comprises the definition of expected results and pass/fail conditions, and translation of Simulation Cases and Simulation Procedures into SCADE Test Environment for Host.

Phase 2: Verification of the SCADE Design Model on a host PC producing formal[2] test results, model coverage results and code coverage results with SCADE Test Environment for Host.

Phase 3: The generated Source Code is provided for integration and tests on the target hardware. Verification cases from Phase 2 are reused on the target hardware to show compliance of Executable Object Code (EOC) with SRATS.

The same requirement-based tests are used in Phase 2 for simulation of the DM to verify compliance with HLR and in Phase 3 to show compliance of the EOC with SRATS. Automated test generation from Specification Models (SMs) using the Simulink Design Verifier (SLDV) during Phase 1 is examined in section 3.1. Simulink Design Verifier (SLDV) is a formal methods plug-in for Simulink/Stateflow models that also features design error detection and property proving. The viability of reusing simulation cases for SW tests on the target hardware as part of Phase 3 is discussed in section 3.2.

3.1 Automated Tests Generation with Simulink Design Verifier

The following aspects were considered for test generation with SLDV

- The SM is only based on (textual) requirements.
- The SM shall be independent of the design.
- The FMRA FCLib can be used (narrows design space as discretisation for filters is given).
- Model referencing is used to model dependencies between SMs in Simulink.
- Internal signals of the DM are not available for tests on the target HW due to constraints of the hardware test environment.
- MATLAB R2016b is used for compatibility with the existing tools for translation from SL to SCADE.

The automated test generation is based on a three-tiered model hierarchy shown in Figure 4.

A Specification Model is an executable representation of a single requirement in Simulink. Its inputs are explicitly independent of the FCL SW interface defined in the ICD. Inputs are created as needed, behaviour that is not specified in the requirement is not part of the associated SM. The SM generates expected output values. The expected output may be an abstract property that is either true or false. This definition of the Specification Model interprets the DO-331 definition as follows:

[2] Formal with respect to achieving RTCA DO-178C/331 objectives

DOI: 10.13009/EUCASS2023-910

- The SM is an abstract representation of functional and performance characteristics of a certain FCL SW component. The levels of abstraction of the SM and an equivalent textual representation are the same. In contrast to a Design Model, the SM does not contain a (more detailed) design that complies with the requirement under consideration.
- The internal control and dataflow of the SM does not specify internal design details such as the internal control flow, the internal data flow or the internal data structure of the FCL SW component.

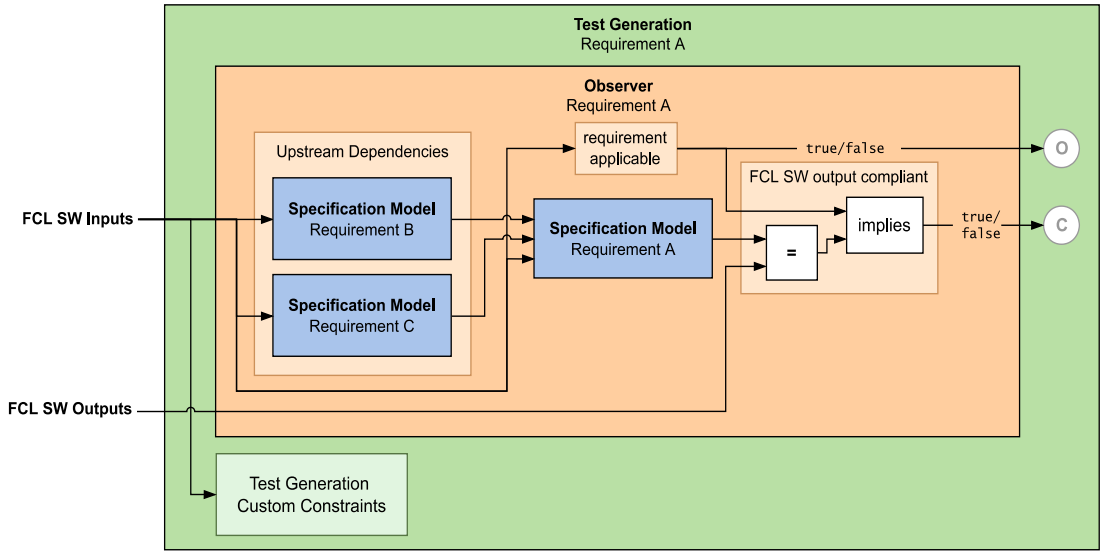

Figure 4: Test generation with specification models in Simulink

An Observer determines if the FCL SW output is compliant with the functional or performance characteristics (referred to as behaviour here) specified by the SM. It includes the SM of the associated requirement and models the signal flow between the FCL SW inputs and the SM. This may include upstream dependencies on other requirements by referencing their Specification Models.

A requirement may be constrained in its applicability. This is reflected by the first output signal of the observer (b1req_act in Figure 5). The expected output values generated by the SM are compared to the actual FCL SW outputs, resulting in the pass-fail output signal (b1req_val in Figure 5). The FCL output is said to be valid (compliant) with regard to a requirement

- if the FCL output matches the expected value from the SM,
- or the requirement is not applicable (not active).

The Observer embodies the manual design of SW test cases by a verification engineer and must be reviewed as such. The following questions have to be considered for the manual test case design:

- Which inputs at the SW interface can influence the specified behaviour?
- Which outputs at the SW interface are available to determine that the FCL SW complies with the considered requirement?
- What is the success criterion?
- Which other requirements represent boundary conditions that must be considered?

The Test Generation Model references the Observer. It is used to generate test in automated way with Simulink Design Verifier (SLDV). It contains SLDV Test Objectives (**O**) and Test Conditions (**C**) connected to the Observer, and a subsystem with constraints on the FCL inputs for test case generation.

Figure 5 shows the model of the Observer for REQ-2 from section 2.1:

REQ-2: The three FCCs shall synchronize their Ground Mode statuses by majority voting.

The voter is specified in the model SM_OnGroundSync, which generates the expected signal for the “on ground” status from the internal “on ground” state b1onGround_own and the two external status signals from the other two FCCs. The internal “on ground” state is defined by upstream requirements that are modelled in SM_OnGround (bottom left) and SM_Vcasf (top left):

DOI: 10.13009/EUCASS2023-910

REQ-5: The FCL shall activate Ground Mode if both sides of the main landing gear are compressed AND the filtered calibrated airspeed is below 60 kt persists for 0.1 s.

REQ-9: The FCL shall low-pass filter the measured calibrated airspeed with the cut-off frequency of 2 rad/s and a high-frequency roll-off of -20 db/decade.

The library elements FILTLP1b (first-order low-pass filter with a time constant of 0.5 s) and CONFLS (sample based confirmation of a leading edge with a confirmation period of 10 time steps for the sampling rate of 100 Hz) are used.

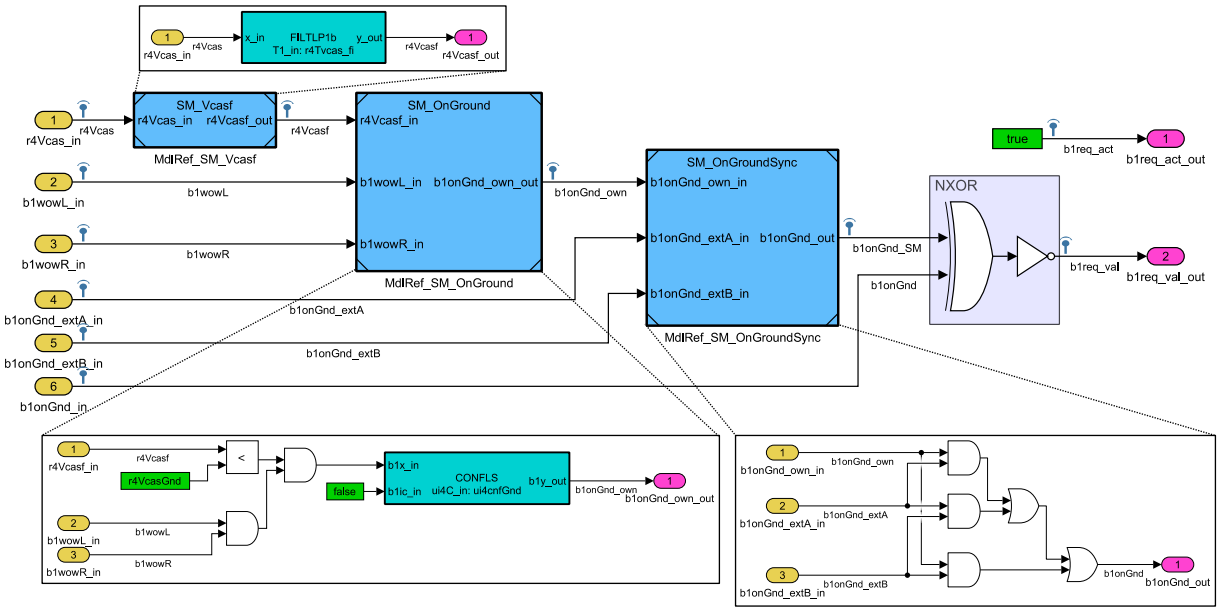

Figure 5: Oracle model for REQ-9 in Simulink

Figure 6 shows the test case that was generated with SLDV under the constraint that the measured calibrated airspeed is in the range of 0 to 200 m/s and the “on ground” status is false in the initial time step. SLDV generates short test cases that achieve the selected model coverage objectives, MCDC coverage in this example. The coverage objectives are added automatically by SLDV. The repetition of the test case in SCADE showed no difference between SL and SCADE. Repetition of this test case on the FCC did not achieve the exact combinations of logical states designed by SLDV due to non-deterministic timing and sampling delays of the test setup in MODULAR.

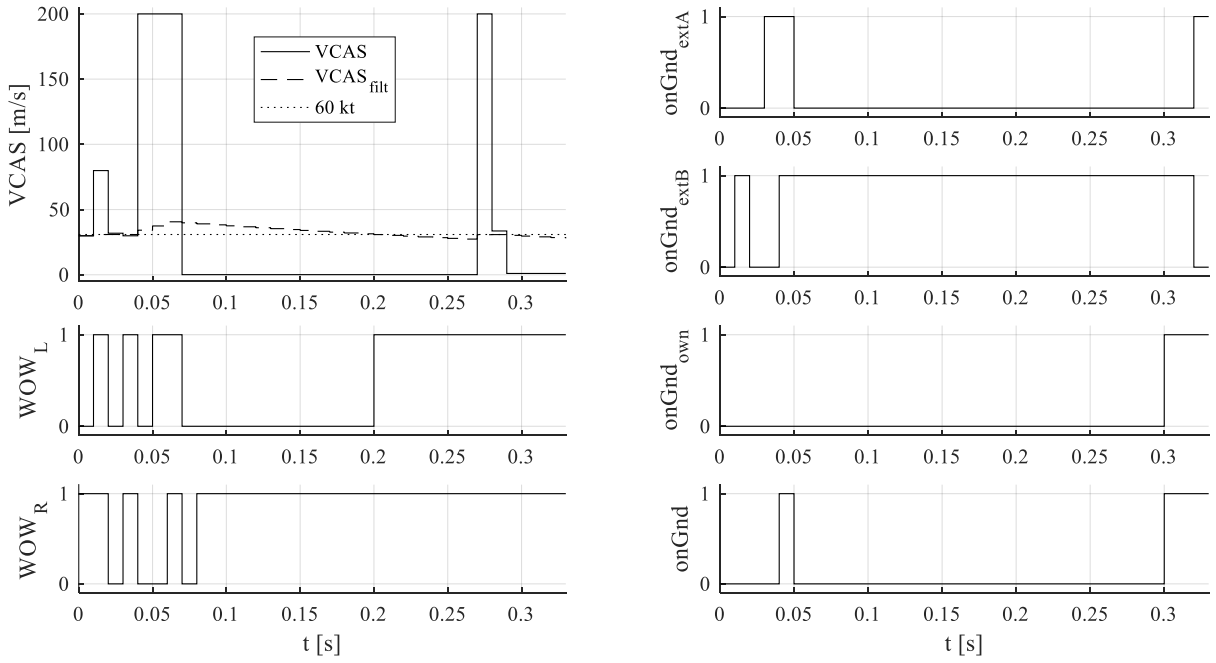

Figure 6: Test case generated for REQ-9 with SLDV

DOI: 10.13009/EUCASS2023-910

3.2 Reuse of Closed-Loop Simulation Cases for SW Verification

Simulation cases, which are used during the development of the Prototype Model in closed-loop simulations in Simulink that belong to the system development process, are reviewed for suitability as verification test cases. The simulation cases are analysed to determine which SRATS have been partially or fully covered. These simulation cases form the basis for defining expected results and pass/fail criteria for the open-loop SW verification. The test cases are converted for use in SCADE Test Environment for Host and for subsequent HW/SW integration tests on the target hardware.

This fulfils the criteria in [3, MB.B.16] for using model coverage analysis to assess test coverage of the LLR:

- High-Level Requirements-based tests are developed.
- These tests are run on the Executable Object Code to verify compliance of the Executable Object Code to High-Level Requirements.
- These same tests are used for the simulation of the Design Model to verify compliance of the Design Model to the High-Level Requirements.

In this case, simulation in combination with model coverage analysis can support the assessment of test coverage of the LLR contained in the DM.

The practical challenge in MODULAR originated from the test setup of the FCC Test Environment: Communication and processing delays distort the timing of input and output signals when compared with the SL simulation. Figure 7 shows a test case with side stick pitch block inputs, which was simulated in Simulink and repeated on the FCC hardware. The timing of the input step and the first output step coincides in Simulink by design. In the FCC Test Environment, the inputs are transmitted via CAN bus to the FCC and outputs are transmitted back to the host computer. The CAN busses' rate is set to 100 Hz. The step in the sidestick pitch command $\delta\eta$ at 40 s is recorded 11 ms later than in the simulation. The reaction in the elevator command η_{cmd} is received after an additional 13 ms. These timing differences need to be considered when comparing data from tests on the target hardware (FCC) with the Simulink simulation data.

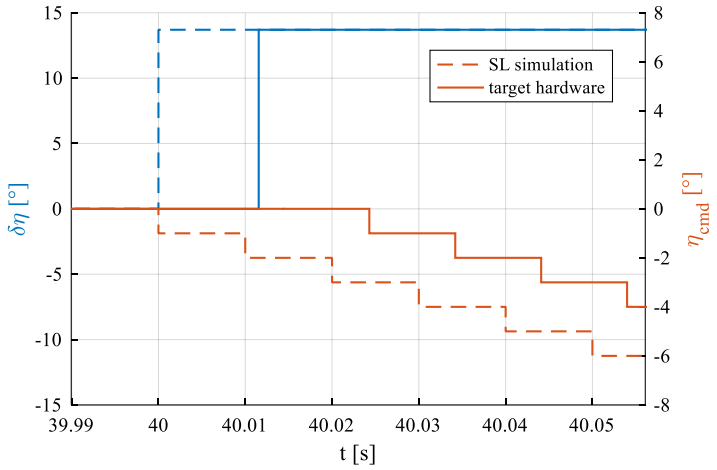

Figure 7: Pitch block input and elevator command timing comparison

A second phenomenon further complicates the comparison. The FCC step time slightly deviates from the nominal step time of 10 ms set in the simulation. The effect is pronounced for long test cases with a duration of more than a few seconds, which is a typical length for the test cases based on closed-loop simulations. This limitation of the test setup specific to MODULAR would be addressed in an actual certification project.

Figure 8 shows the elevator command for a test case of an aborted steep approach. The elevator commands received from the FCC closely match the simulation on a macroscopic level. However, small deviations occur as the input data is sampled slightly faster on the FCC than in the simulation. The difference between the FCC command (black/green) and the simulation (blue) becomes apparent when focusing on the time between 0.6 s and 0.8 s. The sensor data of a single time step in the simulation is sampled twice by the FCC. In this example, the maximum difference of the elevator command is in the range of 0.03°, which is orders of magnitudes larger than the threshold for comparing the Simulink simulation with SCADE Test results of 10^{-5}, which is close to the numerical resolution of the single-precision floating-point data type.

DOI: 10.13009/EUCASS2023-910

The recorded CAN data was fed back into the Simulink simulation to confirm that the differences are caused by the CAN communication delays and input sampling of the FCC. The recorded input data was resampled 18.2 ms before the output signals from the FCC were recorded for each time step. Figure 8 shows the resulting Simulink simulation data with the timing compensation in green on top of the FCC elevator command in black. The elevator commands are virtually identical, with a maximum residual error of 0.0016°, which is considered acceptable in MODULAR.

The analysis of the effects of the FCC hardware timing is necessary as differences between the simulation environment and hardware-in-the-loop test environment need to be justified in a certification context.

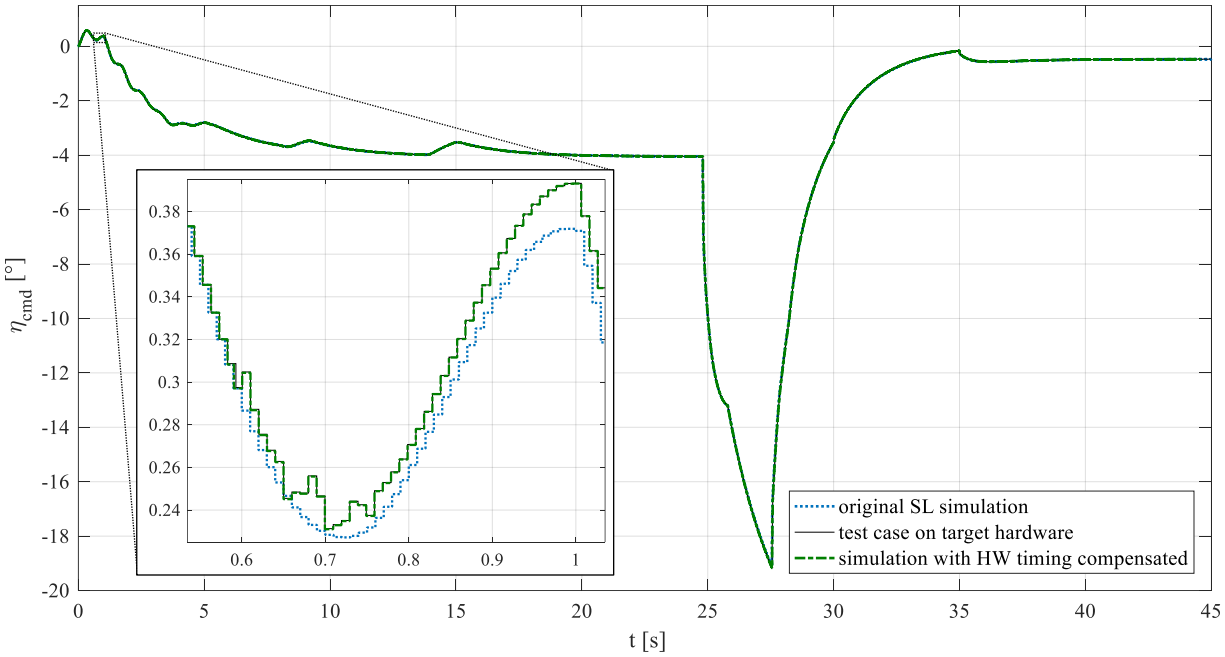

Figure 8: Comparison of elevator command during steep approach simulations and test on target hardware

The example shows that the order of testing set out in DO-331 MB.B.16 is important. Non-deterministic communication and timing delays affect the input test data as seen by the FCC. The testing approach allows us to quickly reproduce the target test in SL simulations to address any discrepancies between the HW tests and simulation environments and record representative model coverage.

The test cases generated by SLDV rely on the exact sequence of inputs to achieve the coverage goals. The generated test cases tend to be short. However, small differences in the inputs that are caused by the HW timing delays can quickly ruin the precise combination of logical states in each time step.

4. Conclusions

As FCS are complex and safety-critical, its development process must adhere to strict guidelines in a certification context. The objective of the MODULAR development process is to optimise the verification activities with regards to efficiency by leveraging model-based design methods. Three contributions towards this goal are examined in this paper. Firstly, the FCL Requirements - a single set of requirements that is used for controller design on the system level and as software high-level requirements - are categorised to identify requirement types that lend themselves to automated test generation. This first step also clarifies responsibilities for verification tasks that are shared between the system and software development processes. The early consideration of verification means during writing of the requirements can improve the quality of the requirements. The classification scheme was developed for the manual flight control (Normal Mode) for a regional jet aircraft. It was validated on a second set of flight control law requirements for the automatic flight control of a HALE UAV. This validation showed that the requirement classification is suited to support verification activities.

DOI: 10.13009/EUCASS2023-910

Secondly, modelling requirements types that do not need closed-loop simulations of the flight control loop were explored by specification models using the Simulink Design Verifier. The approach is based on specification models that are independent of the controller design. The relationship between a Specification Model and the software inputs and outputs is modelled in form of an observer model. This model represents the up- and downstream dependencies between requirements relevant to test case generation. The observer is amended by user-defined constraints for test generation. SLDV can then automatically generate test cases at the FCL software interface. This restriction on externally visible signals is imposed by the FCC Test Environment used in MODULAR. The use of Specification Models has advantages. However, its application to larger and dynamic elements of the FCL software requires further research.

Thirdly, the identity of the interface of the Prototype Model in Simulink, the FCL SW Design Model and the FCL Source Code in the MODULAR development process enables the reuse of verification artefacts of the system-level controller development for FCL SW tests. The reuse of simulation cases for tests on the target hardware was validated on a representative industrial example. The asynchronous communication with the target FCC and its timing presents a challenge when comparing simulation cases with the tests run on the FCC in the test setup. Accounting for these delays when feeding the recorded target test back into the simulations reduced the differences to an acceptable level. In a certification context, any differences between the testing of the EOC integrated on the target hardware and the simulation environments need to be justified. Nevertheless, the repetition of system-level tests on target reached a code coverage above 95% in case of the HALE UAV AFCL software.

Acknowledgements

The development process used in MODULAR is based on the model-based development process established by Bryan Laabs and Dimitry Chernetsov in the FCL-Methods project and previous work by Georg Walde at the Department for Flight Mechanics, Flight Control and Aeroelasticity.

The work presented in this paper was funded by the Federal Ministry for Economic Affairs and Climate Action (BMWK) as part of its Federal Aviation Research Programme (LUFO VI-1) on the basis of a decision by the German Bundestag within the scope of the joint research project MODULAR and TU Berlin's partner project MODULAR-TUB (grant number 20Y1910C). The authors gratefully acknowledge the support.

References

[1] SAE ARP 4754A. 2010. Aerospace Recommended Practice: Guidelines for Development of Civil Aircraft and Systems. Society of Automotive Engineers.

[2] RTCA DO-178C. 2011. Software Considerations in Airborne Systems and Equipment Certification. Radio Technical Commission for Aeronautics.

[3] RTCA DO-331. 2011. Model-based development and verification supplement to DO-178C and DO-278A. Radio Technical Commission for Aeronautics.

[4] SAE ARP 4761. 1996. Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment. Society of Automotive Engineers.

[5] ISO/IEC/IEEE 29148:2018. 2018. ISO/IEC/IEEE International Standard - Systems and software engineering - Life cycle processes - Requirements engineering.

[6] CS-25. 2021. Certification Specifications and Acceptable Means of Compliance for Large Aeroplanes: CS-25. Ammendment 27. European Union Aviation Safety Agency.

[7] Souyris, J., V. Wiels, D. Delmas, and H. Delseny. 2009. Formal Verification of Avionics Software Products. In: *FM 2009: Formal Methods (Lecture Notes in Computer Science)*. 532–546.

[8] Moy, Y., E. Ledinot, H. Delseny, V. Wiels, and B. Monate. 2013. Testing or Formal Verification: DO-178C Alternatives and Industrial Experience. In: *IEEE Softw*. vol. 30, no. 3: 50–57.

[9] Moitra, A., *et al*. 2019. Automating requirements analysis and test case generation. In: *Requirements Eng*. vol. 24, no. 3: 341–364.

[10] Ochoa Escudero, C., R. Delmas, T. Bochot, M. David, and V. Wiels. 2018. Automatic Generation of DO-178 Test Procedures. In: *NASA Formal Methods (Lecture Notes in Computer Science)*. 399–415.

[11] Kuhn, M. R., M. Otter, and T. Giese. 2015. Model Based Specifications in Aircraft Systems Design. In *Proceedings of the 11th International Modelica Conference, Versailles, France*. 491–500.

[12] Chernetsov, D., B. Laabs, R. Paul, and R. Luckner. 2023. Model-Based Development of a Library with Standard Functions for Safety-Critical Flight Control Laws. In: *Proceedings of the Aerospace Europe Conference 2023, Lausanne*.