



HAL
open science

A Lean and Highly-automated Model-Based Software Development Process Based on DO-178C/DO-331

Konstantin Dmitriev, Shanza Ali Zafar, Kevin Schmiechen, Yi Lai, Micheal Saleab, Pranav Nagarajan, Daniel Dollinger, Markus Hochstrasser, Florian Holzapfel, Stephan Myschik

► **To cite this version:**

Konstantin Dmitriev, Shanza Ali Zafar, Kevin Schmiechen, Yi Lai, Micheal Saleab, et al.. A Lean and Highly-automated Model-Based Software Development Process Based on DO-178C/DO-331. 39th Digital Avionics System Conference, Oct 2020, online, United States. hal-02901623

HAL Id: hal-02901623

<https://hal.science/hal-02901623>

Submitted on 17 Jul 2020

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.

A Lean and Highly-automated Model-Based Software Development Process Based on DO-178C/DO-331

Konstantin Dmitriev, Shanza Ali Zafar, Kevin Schmiechen, Yi Lai, Micheal Saleab, Pranav Nagarajan, Daniel Dollinger, Markus Hochstrasser and Florian Holzapfel

Technical University of Munich

Garching, Germany

{konstantin.dmitriev, shanza.zafar, kevin.schmiechen, yi.lai, micheal.saleab, pranav.nagarajan, daniel.dollinger, markus.hochstrasser, florian.holzapfel}@tum.de

Stephan Myschik

Universitaet der Bundeswehr Muenchen

Neubiberg, Germany

stephan.myschik@unibw.de

Abstract—The emergence of a global market for urban air mobility and unmanned aerial systems has attracted many startups across the world. These organizations have little training or experience in the traditional processes used in civil aviation for the development of software and electronic hardware. They are also constrained in the resources they can allocate for dedicated teams of professionals to follow these standardized processes. To fill this gap, this paper presents a custom workflow based on a subset of objectives derived from the foundational standards for safety critical software DO-178C/DO-331. The selection of objectives from the standards is based on the importance, degree of automation, and reusability of specific objectives. This custom workflow is intended to establish a lean and highly automated development life cycle resulting in higher quality software with better maintainability characteristics for research and prototype aircraft. It can also be proposed as means of compliance for software of certain applications such as unmanned aircraft systems, urban air mobility and general aviation. By producing the essential set of development and verification artifacts, the custom workflow also provides a scalable basis for potential future certification in compliance with DO-178C/DO-331. The custom workflow is demonstrated in a case study of an Autopilot Manual Disconnection System.

Index Terms—DO-178C, DO-331, software assurance, safety critical systems, lean software development, model-based development, requirements management, agile development

I. INTRODUCTION

Software is an essential part of modern air vehicles and it usually implements complex safety-critical functions. To ensure safety of airborne software, the aviation community uses the standard DO-178C [1] and its supplements (e.g. DO-331 [2] addressing model-based aspects) as a primary guidance for airborne software development. They describe in detail "objectives for software life cycle processes, activities that provide a means for satisfying those objectives, descriptions of the evidence in the form of software life cycle data [...] [and] variations in the objectives [...] by software level [and]

additional considerations (for example, previously developed software)" [1, p. 1]. However, achieving compliance with DO-178C significantly increases cost and time of development. The growth of project expenses for DO-178C level C software can exceed 150% compared to the development of typical high-quality consumer software per Capability Maturity Model Integration (CMMI) Level 2 or 3 [3]. While this expense may be acceptable to some extent for large transport aircraft (given the perception of risk in this category), it is a major hindrance to the advancement of technology – especially avionics – in normal category airplanes and rotorcraft. The availability of cost-effective qualified avionics solutions for these categories could prevent several general aviation (GA) accidents, such as controlled flight into terrain in poor weather or loss of control. This challenge has received increased visibility in recent years, owing to the growth of rapidly emerging and innovative markets such as urban air mobility (UAM) and unmanned aircraft systems (UAS).

Manufacturers and component suppliers developing avionics solutions for GA, UAS/UAM are cooperating with certification authorities to develop alternative means of compliance for software assurance other than DO-178C. The recently created EUROCAE and RTCA joint working group WG-117 is targeting to develop a software standard that will be specific to lower risk UAS and GA applications [4]. ASTM International has released standards on the following topics intended to streamline avionics certification: system-level verification to assure safety of the intended function (ASTM F3153 [5]), run-time assurance of complex functions (ASTM F3269 [6]), dependability of software used in UAS (ASTM F3201 [7]). These standardization development efforts target applicants with little or no experience with certification, especially in the aerospace industry. At the same time, it is also expected that software certification for high-risk operations in UAS/UAM

will still require conformity to DO-178C processes.

Many developers also use rapid prototyping methods during their research, development and flight testing phases. With constrained resources, a lack of knowledge and a still developing certification environment for their products, these companies are looking for processes as agile as their development cycles. However, certain DO-178C objectives make it difficult to reuse software previously developed out of DO-178C context. For example, in case of poor traceability between requirements of all levels down to source code, it may be easier to redo the development from scratch, rather than re-engineer trace data, improve requirements and rework source code accordingly. This aspect also contributes towards the lean though scalable development life cycle for prototype and experimental airborne software. For such categories of software products, we deem it beneficial to establish a lean but scalable software development life cycle, which helps to achieve a high level of software quality and provides a basis for potential reuse of experimental software for further certification in accordance with DO-178C.

Use of advanced development techniques like model-based design [8] or agile process models [9] along with automation of development and verification activities with qualified tools [10] can help to reduce the development effort while achieving full compliance with DO-178C and DO-331. Below, we discuss the lean model-based software development life cycle (*custom workflow*), which utilizes these advance techniques along with a further reduction of the development effort by introducing the *custom subset* of DO-178C and DO-331 objectives. We selected the objectives for the *custom subset* based on criteria of importance, automation and reuse with the following goals:

- 1) Reduce non-critical process activities to minimize development efforts while maintaining a software quality level similar to DO-178C level C.
- 2) Enable the completion of software development to achieve full compliance with DO-178C per level C or higher in the future without wide rework of the data produced using the custom workflow.

This paper is structured in the following sections. Section II describes the methodology for the selection of the *custom subset* of objectives. In section III, a general overview of the workflow to satisfy the selected objective using *MathWorks* and own tools is given. This is followed by an explanation of in-house processes and tools to facilitate the *custom workflow* in sections III-A to III-D. Section IV describes the achieved level of compliance to DO-178C and DO-331 using the *custom workflow*. A case study is performed by developing an *Autopilot Manual Disconnection System* (AMDS) using the proposed workflow and is described in section V. Section VI discusses the planned enhancement for the *custom workflow*. At the end, conclusions are given in section VII.

II. OBJECTIVES SELECTION CONCEPT

We considered the following criteria to select the DO-178C and DO-331 objectives for the *custom subset*:

- Importance of the objective
- Level of automation for objective implementation
- Level of potential reuse of objective data (artifacts)

Importance of the objective was evaluated based on following factors:

- 1) In DO-178C, the objectives that need to be satisfied during software development depend on software Levels A to E resulting from the safety assessment process. Level A corresponds to the most critical *Catastrophic* failure condition category and requires most objectives to be satisfied with maximum rigor while Level E corresponds to *No Safety Effect* and requires no objectives to be satisfied. Objectives applicable to all software levels from A to D are considered more important than the objective assigned to only higher levels (i.e. level C or higher). Examples include the definition of high-level software requirements and testing of executable object code for compliance with high-level requirements.
- 2) Relevance of the objectives to other aerospace and non-aerospace software development standards (ASTM F3201 [7], ISO 26262-6 [11], ISO 26262-8 [12], EN 50128 [13] have been considered). For example, traceability down to source code is not required by DO-178C/DO-331 for Level D, but EN 50128 and ISO 26262-8 recommend the complete traceability down to source code.

Level of automation was another key factor for selecting the objectives for the *custom subset*. If the objective can be automated using a tool, it can help to improve overall software quality without significant increase of manual efforts. Examples include:

- Requirements verification objectives (tables MB.A-3 and MB.A-4 of DO-331) which can be substantially automated for model-based software (MBS) components leveraging model simulation. Test cases and procedures developed for model simulation can be further seamlessly reused for executable code testing.
- Code verification objectives (table A-5/MB.A-5 of DO-178C/DO-331). These objectives are not applicable to lower software levels (Level D and below) and hence are not assigned with high importance as per the criteria above. However, they can be efficiently automated leveraging *Simulink Code Inspector*¹ [14], *Polyspace Bug Finder*² and *Polyspace Code Prover*³ tools with very little additional manual effort.
- Objectives for code structural coverage (table MB.A-7 of DO-331), which can be automated for MBS code, reusing the test cases for model simulation and applying the *Simulink Coverage tool*⁴.

Level of potential reuse was considered as another factor due to the increased value of the artifacts, which can be

¹<https://www.mathworks.com/products/simulink-code-inspector.html>

²<https://www.mathworks.com/products/polyspace-bug-finder>

³<https://www.mathworks.com/products/polyspace-code-prover>

⁴<https://www.mathworks.com/products/simulink-coverage>

leveraged without major modification across multiple projects. The value of such artifacts and corresponding objectives versus efforts spent for their development is multiplied by reuse and therefore such objectives are considered more relevant to the *custom subset*. For example, certain parts of project development and verification plans and standards can be written in a generic way to be reusable for other development projects being implemented in a similar environment.

We evaluated DO-178C and DO-331 objectives and assigned each objective with a high, medium and low level of applicability for each selection criteria. Based on the total score qualitatively assessed as a combination of all three criteria levels, we selected DO-331 and DO-178C objectives for the *custom workflow*. This qualitative assessment was based on consensus between the authors. Fig. 1 and 2 show the summary of the selected DO-331 objectives applicable to MBS components and DO-178C objectives applicable to manually coded low-level software (LLS) components respectively. Some of the objectives are not applicable to the *custom workflow* and shown as N/A on the figures. For example, certification liaison objectives are not applicable because a formal certification process is not implemented in the *custom workflow*. These figures also indicate the compliance level achieved for each objective in the implemented *custom workflow*. This is further discussed in section IV.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
MB.A-1	P	P	P	-	P	-	-									
MB.A-2	F	P	F	F	-	F	F	F	F	F						
MB.A-3	F	F	P	F	F	P	F	-	-	F						
MB.A-4	F	F	P	F	F	P	F	F	F	P	F	F	NA	-	-	F
MB.A-5	F	F	F	F	F	P	-	NA	NA							
MB.A-6	F	F	P	-	F											
MB.A-7	-	F	P	F	-	F	F	F	-	NA	NA	NA				
MB.A-8	F	F	P	P	P	P										
MB.A-9	-	P	-	-	-											
MB.A-10	NA	NA	NA													

Full compliance	F
Partial compliance	P
Not included in the custom workflow	-
Not applicable for the custom workflow	NA

Fig. 1. DO-331 Compliance Summary for Model-based Software

III. WORKFLOW OVERVIEW

The workflow to achieve the *custom subset* of DO-178C/DO-331 objectives discussed in section II can be structured in these parts:

- Planning activities for applicable objectives corresponding to table A-1/MB.A-1 of DO-178C/DO-331.
- Development activities corresponding to table A-2/MB.A-2 of DO-178C/DO-331.
- Verification activities corresponding to table A-3/MB.A-3 through A-7/MB.A-7 of DO-178C/DO-331.
- Configuration management and quality assurance activities corresponding to tables A-8/MB.A-8 and A-9/MB.A-9 of DO-178C/DO-331.

	1	2	3	4	5	6	7	8	9	10	11	12	13
A-1	P	P	P	-	-	-	-						
A-2	F	P	P	-	-	F	F						
A-3	-	-	-	-	-	P	-						
A-4	-	-	-	-	-	-	-	-	-	-	-	-	NA
A-5	-	-	F	F	-	P	-	NA	NA				
A-6	F	F	-	-	F								
A-7	-	F	P	-	-	-	-	-	-				
A-8	F	F	P	P	P	P							
A-9	-	-	-	-	-								
A-10	NA	NA	NA										

Full compliance	F
Partial compliance	P
Not included in the custom workflow	-
Not applicable for the custom workflow	NA

Fig. 2. DO-178C Compliance Summary for Low-level Software

We are using *MathWorks* products for automation of many objectives due to legacy projects and supported *MathWorks*-based solutions but the workflow can also be adapted for other commercial Model-based Development (MBD) toolchains, for instance *Ansys SCADE Suite*⁵. An overview of the development and verification activities for the software development workflow is shown in Fig. 3. Configuration management and quality assurance are applied for all artifacts and not shown on the figure for simplicity.

Planning activities can require significant manual effort but the reusability of artifacts across different projects is high as mentioned in section II. For our lean *custom workflow*, planning activities include defining the life cycle in form of an interactive user guide detailing the activities to be performed. This guide also includes the description of the tools to be used and how to use them for MBS and LLS components. Another aspect of the planning process are software development standards. For MBS, DO-331 also requires a *Software Model Standard*. As a part of this, *Modeling Guidelines* based on a subset of *MathWorks* modeling guidelines [15] are developed at our institute for *Design Models*. These are models which are used to generate source code directly. Additional guidelines are also added to overwrite or further restrict functionalities based on experience gathered during previous projects (e.g. for testing or to avoid rounding errors and overflows). *Naming Conventions* are also developed at our institute for consistency and compliance with MISRA C [16] and the *Modeling Guidelines* with a focus on flight dynamics and control.

Development Activities' entry point are textual requirements authored in *Polarion*. In model-based development, a textual requirement may represent different levels of system and software requirements, depending on the selected scenario of model usage as described in DO-331 MB.1.6.3. The *custom workflow* is implemented based on the MB Example 5 practice (see DO-331 Table MB.1-1) for MBS components. We capture system requirements in textual format and further implement

⁵<https://www.ansys.com/products/embedded-software/ansys-scade-suite>

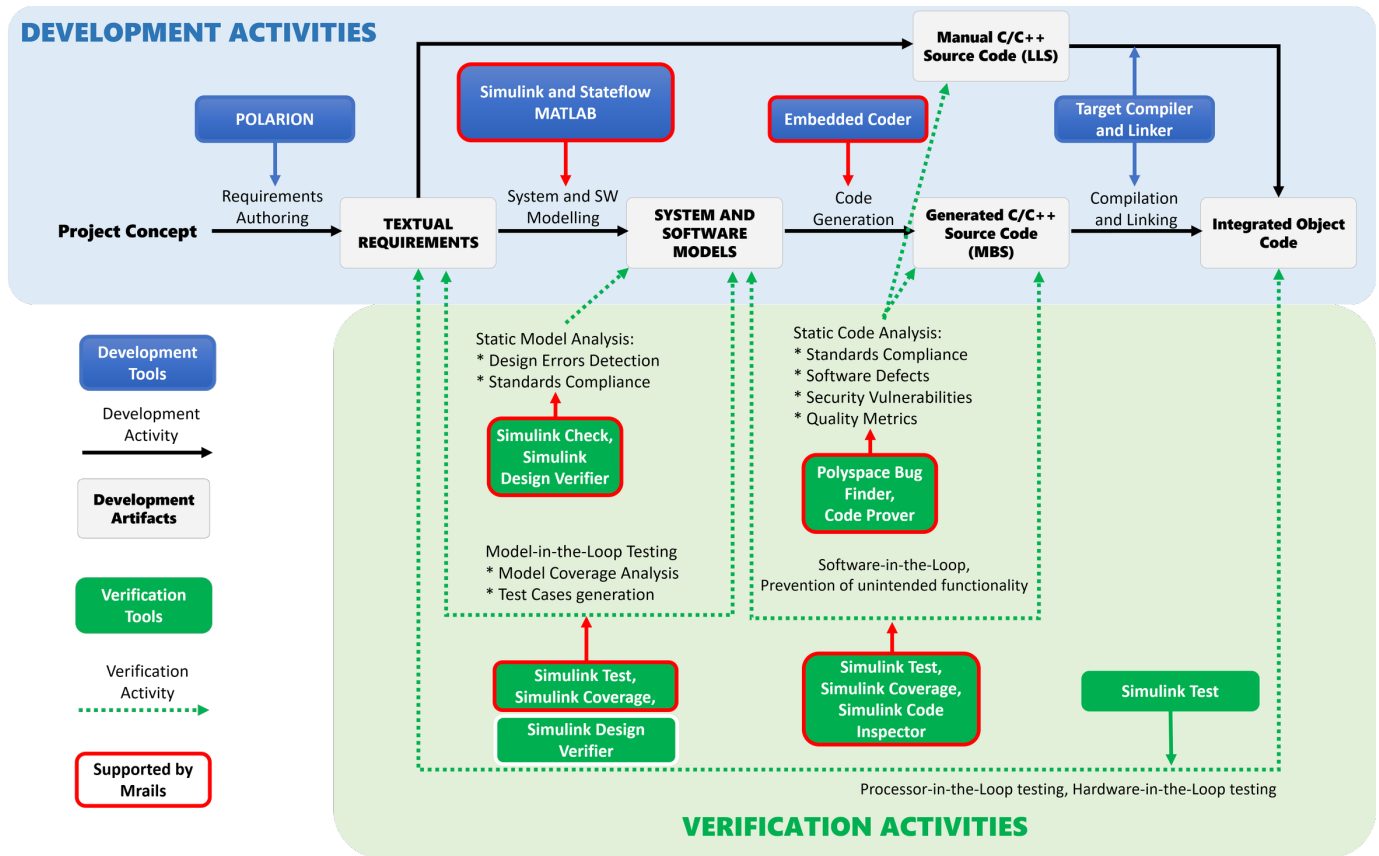


Fig. 3. Development and verification workflow using MathWorks based toolchain for safety critical systems

them as *Design Models* in *Simulink* for the MBS part (implementing functional capabilities) and refine textual software requirements for the LLS part (scheduling layer, hardware interfaces). *Simulink* models represent low-level requirements and software architecture. Source code is then automatically generated from *Design Models* using *Embedded Coder*. Further details about the modeling environment used for *Design Models* can be found in [17].

For the LLS part, source code is manually written based on the refined LLS requirements and intended to be generally reused across projects implemented on the same hardware. Finally, LLS code is integrated with the auto-generated code for MBS code and compiled to get target compatible object code.

Verification activities can be greatly automated and performed early for MBS using *Design Models*. DO-331 allows simulations as means of compliance for the key objectives in table MB.A-3 (Verification of Outputs of Software Requirement Process) and further objectives in table MB.A-4 (Verification of Outputs of Software Design Process). Example of these objectives include *high-level requirement comply with system requirements, requirements are accurate and consistent, algorithms are correct*. Requirement-based simulation cases can be further fully reused for software-in-the-loop (SIL) and Hardware-in-the-loop (HIL) testing of the executable ob-

ject code to satisfy some objectives of table MB.A-6 (Testing of Outputs of Integration Process). Using *Simulink Test*⁶ for simulation and testing helps achieving objectives in table MB.A-7 (Verification of Verification Process Results) by facilitating structural coverage collection, requirement traceability, and reviewing and managing test results. Reusability of test cases for HIL using *Simulink Test* is further discussed in subsection III-D.

Checking models during development for design errors using *Simulink Design Verifier*⁷ and for conformance to the *Software Model Standard* using *Simulink Check*⁸ helps fulfilling other objectives of MB.A-3 and MB.A-4 (Verification of Outputs of Software Design Process). Furthermore, some objectives corresponding to the verification of source code (both manually coded for LLS and auto-generated for MBS) in table A-5/MB.A-5 (Verification of Outputs of Software Coding and Integration Process) are automated using static code analysis techniques for compliance with standards like MISRA C, detecting run-time errors and for proving absence of some run-time error through *Polyspace Bug Finder* and *Polyspace Code Prover*. Additionally for MBS, objectives for source code compliance to low-level requirements and

⁶<https://www.mathworks.com/products/simulink-test.html>

⁷<https://www.mathworks.com/products/simulink-design-verifier>

⁸<https://www.mathworks.com/products/simulink-check>

software architecture in table MB.A-5 are accomplished by *Simulink Code Inspector*.

Configuration management activities are implemented by using *Polarion* for requirements management and *GitLab* for other artifacts. These tools implement essential activities required for the DO-178C/DO-331 *Control Category 2*: identification, traceability and change tracking. More stringent activities required for the DO-178C/DO-331 *Control Category 1* may hinder rapid prototyping and are not included in the *custom workflow*.

Quality assurance activities are implemented using the tool *mrails* described below and are limited to tracking the completion status of the planned activities. For example, the left panel of the *mrails* status overview window presented in Fig. 3 shows the status of the planned workflow activities.

The proposed *custom workflow* enables high automation in both development and verification activities owing to MBD, which puts the foundation for a lean workflow based on DO-178C. Few examples of MBD for safety critical systems in literature are [18]–[20].

In the following subsections, we will discuss the specific solutions employed at the institute which facilitate the implementation of the lean *custom workflow* in addition to the *MathWorks* tools mentioned above.

A. Requirements Management

This section presents the requirement management setup for our *custom workflow*. We used the tool *Polarion*, which is a web-based application life cycle management solution by *Siemens Digital Industry Software*, to manage the textual requirements and their trace data. While writing the requirements, we applied the *FRETISH* language of the tool *FRET* (Formal Requirements Elicitation Tool) as much as possible. It was developed at *NASA Ames Research Center* and was presented in [21]. The tool *SimPol*, which was developed at our institute, is used to create bidirectional traceability between requirements on *Polarion* and *Design Models* and test cases in *Simulink Test*. Further details in the context of requirement derivation, simulation and validation are discussed in [22], [23].

Artifact Hierarchy for our *custom workflow* is shown in Fig. 4. It includes the work item types (e.g. requirements) but also the link roles. Lower level requirements refine upper level requirements, however, as a simplification, it is also possible that requirements refine those from the next but one level (e.g. from software to system) or that requirements refine other requirements from the same level (e.g. component) because a higher level of detail is necessary within one level. Even though the arrows are only pointing in one direction, the traces are still bidirectional (e.g. forward trace: refines / backward trace: is refined by). Additionally, review checklists are used to hold traceability evidence based on the automatic trace reports described below. The other verification objectives of tables A-3/MB.A-3 and A-4/MB.A-4 of *DO-178C/DO-331*, which require manual review, can be included in the review

checklist when increasing the compliance level with respect to the standard.

Cross-Platform Artifact Linking is done with *SimPol* between the requirements on *Polarion* and the *Design Models* and test cases in *MATLAB/Simulink*. The tool creates model and test case surrogate work items as described in Fig. 4, which serve as a representation on *Polarion* for the actual models and test cases. Automatic highlighting of links where either side has changed enables an impact analysis of such changes. The available links can also be used to identify the impact of changes to other work items before the changes are even made.

In accordance with the wording of *DO-331*, the *Design Models* have the link role *implements* to software requirements and to those component and system requirements that are allocated to software. Model-based test cases have the link role *verifies* to requirements on *Polarion*. An overview of the steps to link between *Polarion* and *Simulink Test* is given in Fig. 5. In the first two steps, the requirements on *Polarion* are linked with the *Design Models* in *Simulink* and the test cases in the *Simulink Test Manager*. The link from model-based test case to *Design Model* is already established during the creation of the test case and its harness. The third step is to upload the test run records to *Polarion*. A custom routine processes the records from *Simulink Test Manager* into an XML file. A custom extension on the *Polarion* server processes this XML file, creates the test run records and links them to the test case surrogates.

Automatic Trace Reports can be generated on *Polarion* with a custom widget as soon as the work items are created and linked with each other. These reports correspond to parts of *Trace Data* required by section 11.21 of *DO-178C*. In addition to the actual trace table, the report also consists of a pie-chart with the distribution of the trace status of the work items for a quick overview. Derived requirements, which are identified by a custom field, are one example where it is justified that a requirement does not have an upstream trace.

The custom widget was inspired by a widget for test case coverage that is part of the default *Polarion* installation, however, our widget was developed to be applicable to our requirement traceability and even more use cases. It also supports checking traces in both directions (e.g. also from lower to higher level requirements).

B. Automation of Life cycle Activities

A process-oriented build tool (*mrails*) for MBS using the *MathWorks* toolchain was developed at our institute by Hochstrasser [10] which greatly reduces the efforts required for certification-ready software development by automating the life cycle activities discussed in section III. In Fig. 3, activities supported and automated by *mrails* in the overall workflow are highlighted.

For the development of *Design Models*, *mrails* integrates the modeling environment and supports the creation of 'ready-by-construct' building elements such as reusable models, or *Simulink* bus objects compatible with *Modeling Guidelines*.

with minimum additional effort. Fig. 6 depicts the overview of the CI process. Based on the contents of a new commit from the developers to the source control server, certain series of tests, which are mostly in the form of *Mrails jobs*, are triggered. The test results are automatically fed back to the developers via *GitLab* and to *Polarion* via the pre-configured links between the requirements, *Design Models* and the test cases that are described in section III-A. A *git* branch merge into the release or master branch will be rejected if the tests do not pass. This guarantees the quality of the software and allows merges to be performed more often without waiting for manual review.

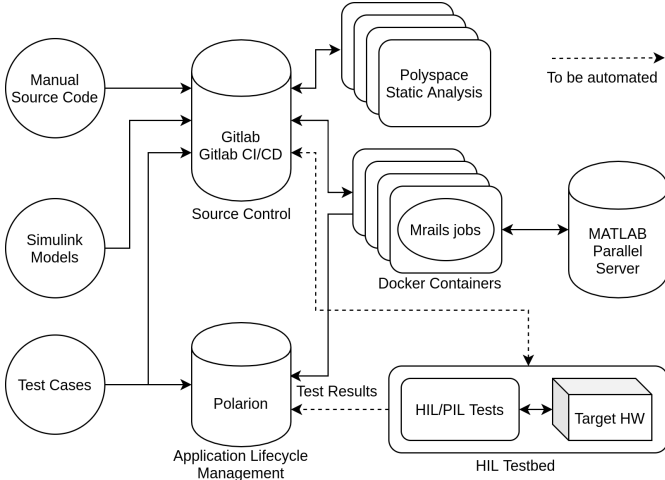


Fig. 6. Continuous Integration Overview

D. Hardware-in-the-Loop Tests

HIL simulations operate a unit-under-test within a representation of its real-world environment [26]. Therefore, a target which executes the developed software is connected to a real-time simulation of the later environment the target will be operating in. The basic concept of HIL uses the target’s original interfaces for input stimulation from the simulation environment as well as for collecting the target’s output into the simulation environment. Thus, a HIL simulation allows to evaluate the target’s integrity under real world conditions like input-output interaction, real-time behavior, interface delays and jitters.

Certain objectives in DO-178C pertaining to the verification of Executable Object Code cannot be satisfied with simulation alone and need to be executed on the target environment [27]. A classical approach is the definition and creation of HIL test cases for requirements-based testing. These test cases are run in HIL simulation and demonstrate, if passed, that the software satisfies the requirements and can be considered as successfully verified [28].

Since HIL testing is a time-consuming process [28], already available test cases from previous verification methods are reused to reduce the implementation effort.

As already introduced in section III-A, test cases are created in *Simulink Test* and linked to the corresponding requirement.

These test cases are used to verify the functional software requirements implemented by the *Design Model* in *Simulink* via MIL simulations. To avoid additional effort in creating dedicated test cases for HIL simulation, thus also reducing an error prone manual transfer between MIL and HIL simulation test cases, *Simulink Test Manger* is used to execute identical MIL test cases in HIL. Besides streamlining the verification process by reducing manual effort, another benefit is that traceability between requirements and HIL test cases is automatically ensured as the test case itself is already linked to the requirement.

In order to be able to apply the same test case as MIL in HIL simulation, the connection of test case vectors to the *Design Model’s* inputs and outputs, is substituted by a hardware interface between the development environment which is running the *Simulink Test Manager* and the HIL real-time simulation as shown in Fig. 7.

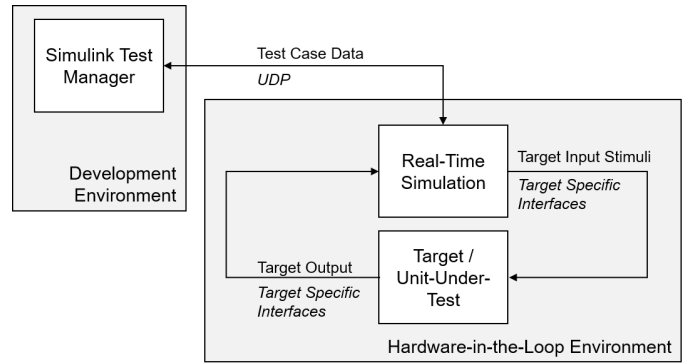


Fig. 7. HIL Environment

We have used UDP communication between the *Simulink Test Manager* which is running the test cases test sequence and the HIL real-time simulation. UDP was selected as a standardized interface between physical machines which can be easily adapted to test cases depending on their input and output vectors also known as test data. *Simulink* furthermore directly provides a block-set for UDP communication. In our example, it is assumed that both the development environment and the HIL real-time simulation are on different computing units or physical machines. The software running on the target platform is receiving the same stimuli from the HIL real-time simulation as the design model receives during MIL simulation. The same holds for its outputs. Using this method, the MIL and HIL test case results are obtained from an identical test case and the results of both tests are supposed to be equivalent as long as the software is implemented correctly.

IV. LEVEL OF COMPLIANCE

DO-178C and DO-331 specify various activities, which should be implemented to achieve the objectives of these standards. The artifacts to be produced during the activities is specified in section 11 of DO-178C and DO-331. Some of the required activities and elements of the artifacts have

been considered as not applicable to the *custom workflow*. While implementing the *custom workflow*, we identified and assessed such inapplicable activities and artifact elements. If they were considered as reproducible without major rework of preexisting artifacts, we excluded them from the custom workflow. Examples are:

- Elements of the planning data intended for a formal certification process with certification authorities do not apply to the *custom workflow* (e.g. certification considerations and schedule, system and software description).
- Activities of the configuration management process specific to the operation phase of the aircraft (e.g. software part numbering, load control and aircraft configuration records).

We also omitted some activities, which cannot be efficiently automated with the tool, however, can be reproduced without major rework afterwards for formal certification. For example:

- Completeness of traceability between requirements, models, test cases and results was verified in the *custom workflow* using automated trace reports. However, manual verification of trace data for consistency and correctness was skipped as a time-consuming manual task, which can be implemented afterwards for formal certification.
- Verification activities for LLS requirements which cannot be automated and need extensive manual review.

Omitted activities or artifact elements resulted in partial compliance for some of the objectives implemented in the *custom workflow*. We identified the following levels of compliance for the selected objectives:

- Full when all activities associated with the objective implemented and artifacts produced
- Partial when some activities associated with the objective are not fully implemented or artifact elements not produced.

A summary of the compliance level achieved in the custom workflow is shown in Fig. 1 for DO-331 and Fig. 2 for DO-178C.

V. A CASE STUDY

The *custom workflow* explained above was applied to the development of an experimental *Autopilot Manual Disconnection System* (AMDS). The simplicity of the system ensured that a complete workflow from aircraft requirement authoring in Polarion down to HIL testing could be demonstrated with limited project resources.

Using AMDS, the pilot has aircraft command priority over the autopilot by having the ability to manually disconnect the autopilot at any time during the flight. External interfaces of AMDS are shown in Fig. 8. The pilots interact with the AMDS using two pushbuttons, one on each of the control inceptor. Under normal mode of operation, if any of the pushbutton is pressed, the system disconnects the autopilot within 100 ms by controlling the transmission clutch.

A control unit with two isolated and independent channels simultaneously processing the desired function is used. This

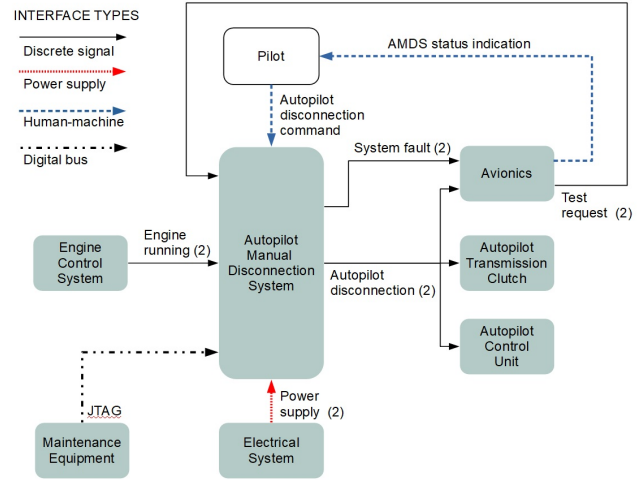


Fig. 8. External Interfaces of AMDS

redundancy ensures that a single failure doesn't lead to loss of primary function. The control unit also implements features typical for airborne controllers, including comprehensive built-in tests and operation-safe maintenance capabilities.

We developed the AMDS software according to the *custom workflow* and tools discussed in section III. After eliciting the requirements on *Polarion* as described in section III-A, the AMDS functionality is implemented in *Simulink*, whereas test cases assessing the temporal behavior and mode logics based on requirements are authored in *Simulink Test*. Trace data is created between *Simulink* models and *Simulink* test cases, and requirements on *Polarion* using *SimPol*.

During the development, we used *mrails* described in section III-B to execute code generation and all the model and code verification activities shown in Fig. 3 (excluding HIL testing which is implemented in a real-time environment). A snippet of *mrails*' status overview showing model coverage is given in Fig. 9. After pushing to *Gitlab*, the CI executes the whole set of tests and static checks to make sure that the modifications are ready to be merged.

The auto-generated software (MBS) is integrated with LLS and hardware which are reused from the real flight control computer (FCC) described in [29]. Building the final loadable executable object code including both MBS and LLS parts is implemented in CI.

To satisfy target specific objectives of requirement-based testing, HIL simulations are performed using TechSAT ADS2¹³. As described in section III-D, MIL test cases are reused for HIL simulations using the UDP interface between HIL and Simulink environment. Equivalence between MIL, SIL and HIL cases have shown that the implemented software works as expected in the target environment and complies with requirements.

By implementing the proposed workflow for the AMDS

¹³<https://techsat.com/technology-and-products/ads2/>

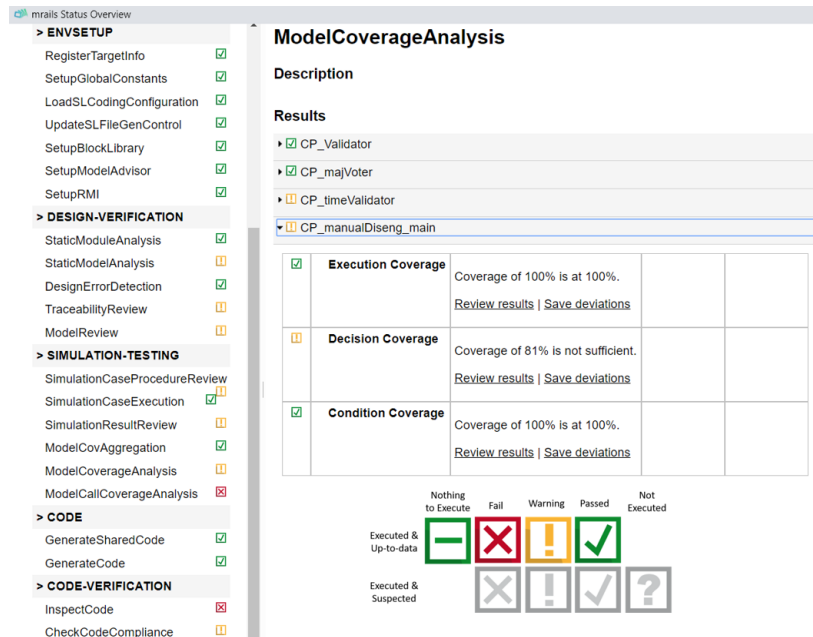


Fig. 9. Mrails Status Window

case study, the following properties of the *custom workflow* were demonstrated:

- The overall development effort is estimated to be about 50% less than typical effort for level C software development by a team without certification experience. Per [3], the metric observed in the industry for level C is about 15 lines of code per person per day which includes verification, documentation and other efforts associated with certification. In the case study, the estimated overall performance rate was about 30 lines of code per person per day for the MBS component.
- High software quality with reduced manual effort. By using early comprehensive testing at model level via simulation with model coverage, no bugs in the MBS component were detected during further SIL and HIL testing. By using modelling guidelines and design error detection earlier for models, no source code defects and only 21 MISRA advisory warnings were found in the MBS code without actively resolving any errors or warnings.

VI. FUTURE WORK

In future work we are planning on upgrading the developed prototype to DO-178C/DO-331 level A and estimate the efforts required to close the gap between the custom workflow and formal DO-178C/DO-331 certification. Another work in progress is applying the custom workflow to a more complex control system to study the compatibility of the approach with a full-scale development project.

For requirements management, we will evaluate the possibility to integrate *FRET* into *Polarion* as an extension. We will also apply the analysis capabilities of *FRET* for linear

temporal logics. The custom widget of the automatic trace reports can also be used for coverage reports of model-based validation which will be part of our extended *custom workflow*. The uploaded test run records on *Polarion* are also not yet used for automatic verification reports but this will be implemented in the near future. It is also possible to configure *Polarion* to link manually written *C/Java/XML* code to work items. These links have not yet been included in the trace reports but they are planned for implementation in the future.

For CI integration, we plan to include automatic HIL tests on the HIL testbed with target hardware in the CI.

VII. CONCLUSIONS

The paper has introduced a lean, highly-automated and scalable software development workflow intended for prototype and experimental airborne systems. The proposed workflow is based on the subset of DO-178C and DO-331 objectives which we selected based on criteria of importance, available automation and reuse potential to streamline the development. Extensive automation of development and verification activities using the qualified model-based commercial toolchain along with the own custom tools and continuous integration environment was leveraged to reduce manual development effort while maintaining a high software quality. This *custom workflow* is aimed to be used a basis for potential future certification in full compliance with DO-178C/DO-331 without wide rework of the produced data.

Using the proposed *custom workflow*, we developed a small system prototype to study the advantages and limitations of this method. We showed that development fulfilling the selected DO-178C objectives using the *custom workflow* can reduce development efforts by about 50% comparing to average metrics observed in the industry for DO-178C level

C software. Yet, a high software quality was achieved with respect to the number of bugs detected and compliance to coding guidelines.

We plan in our future work to estimate the efforts needed to close the gap between the *custom workflow* and level A software development and also explore the relevance of the workflow to a full-scale development of a complex system.

REFERENCES

- [1] *Software Considerations in Airborne Systems and Equipment Certification*, RTCA, Inc. Std. RTCA DO-178C, 2011.
- [2] *Model-Based Development and Verification Supplement to DO-178C and DO-278A*, RTCA, Inc. Std. RTCA DO-331, 2011.
- [3] E. Thomas, "Certification cost estimates for future communication radio platforms," *Rockwell Collins Inc., Tech. Rep.*, 2009.
- [4] New working group: Wg-117 / topics on software advancement. EUROCAE. [Online]. Available: <https://www.eurocae.net/news/posts/2020/february/new-working-group-wg-117-topics-on-software-advancement/>
- [5] *Standard Specification for Verification of Avionics Systems*, ASTM International Std. ASTM F3153-15, 2015.
- [6] *Standard Practice for Methods to Safely Bound Flight Behavior of Unmanned Aircraft Systems Containing Complex Functions*, ASTM International Std. ASTM F3269-175, 2017.
- [7] *Standard Practice for Ensuring Dependability of Software Used in Unmanned Aircraft Systems (UAS)*, ASTM International Std. ASTM F3201-16, 2016.
- [8] L. Marcil, "MBD, OOT and Code Generation: A Cost-Effective Way to Speed Up HMI Certification," in *SAE Technical Paper*. SAE International, 10 2012. [Online]. Available: <https://doi.org/10.4271/2012-01-2137>
- [9] D. J. Coe and J. H. Kulick, "A model-based agile process for DO-178C certification," in *Proceedings of the International Conference on Software Engineering Research and Practice (SERP)*. The Steering Committee of The World Congress in Computer Science, Computer , 2013, p. 1.
- [10] M. Hochstrasser, S. Myschik, and F. Holzapfel, "A process-oriented build tool for safety-critical model-based software development," *Change*, vol. 2, no. 3, p. 4, 2018.
- [11] *Road vehicles Functional safety Part 6: Product development at the software level*, ISO (International Organization for Standardization) Std. ISO 26262-6:2018, 2018.
- [12] *Road vehicles Functional safety Part 8: Supporting processes*, ISO (International Organization for Standardization) Std. ISO 26262-8:2018, 2018.
- [13] *Railway applications - Communication, signalling and processing systems - Software for railway control and protection systems*, DIN Std. DIN EN 50128:2012-03, 2012.
- [14] M. Conrad, T. Erkkinen, M. Englehart, X. Lin, A. R. Nirakh, B. Potter, J. Shankar, P. Szpak, and J. Yan, "Automating code reviews with simulink code inspector," in *MBEES*, 2012, pp. 31–36.
- [15] Modeling guidelines. MathWorks. [Online]. Available: <https://www.mathworks.com/help/simulink/modeling-guidelines.html>
- [16] *Guidelines for the Use of the C Language in Critical Systems*, MISRA paperback, March 2013, ISBN 978-1-906400-11-8.
- [17] M. Hochstrasser, S. P. Schatz, K. Nürnberger, M. Hornauer, S. Myschik, and F. Holzapfel, "Aspects of a consistent modeling environment for do-331 design model development of flight control algorithms," in *Advances in Aerospace Guidance, Navigation and Control*. Springer, 2018, pp. 69–86.
- [18] R. G. Estrada, G. Sasaki, and E. Dillaber, "Best practices for developing do-178 compliant software using model-based design," in *AIAA Infotech@ Aerospace (I@A) Conference*, 2013, p. 4566.
- [19] J. Krizan, L. Ertl, M. Bradac, M. Jasansky, and A. Andreev, "Automatic code generation from matlab/simulink for critical applications," in *2014 IEEE 27th Canadian Conference on Electrical and Computer Engineering (CCECE)*. IEEE, 2014, pp. 1–6.
- [20] T. Erkkinen and M. Conrad, "Safety-critical software development using automatic production code generation," *SAE Transactions*, pp. 474–485, 2007.
- [21] D. Giannakopoulou, T. Pressburger, A. Mavridou, J. Rhein, J. Schumann, and N. Shi, "Formal requirements elicitation with fret," in *CEUR Workshop Proceedings*, vol. 2584, 2020. [Online]. Available: <http://ceur-ws.org/Vol-2584/PT-paper4.pdf>
- [22] K. Schmiechen, M. Hochstrasser, J. Rhein, C. Schropp, and F. Holzapfel, "Traceable and model-based requirements derivation, simulation, and validation using matlab simulink and polarion requirements," in *AIAA Scitech 2019 Forum*, 2019.
- [23] V. Meidinger, K. Schmiechen, and F. Holzapfel, "A model-based requirement validation process for handling qualities of evtols," in *Deutscher Luft- und Raumfahrtkongress 2019*. Deutsche Gesellschaft für Luft- und Raumfahrt - Lilienthal-Oberth e.V., 2019.
- [24] M. Hochstrasser, S. Myschik, and F. Holzapfel, "Application of a process-oriented build tool for flight controller development along a DO-178C/DO-331 process," in *Model-Driven Engineering and Software Development*, S. Hammoudi, L. F. Pires, and B. Selic, Eds. Cham: Springer International Publishing, 2019, pp. 380–405.
- [25] C. Comar, F. Gasperoni, and J. Ruiz, "Open-do: An open-source initiative for the development of safety-critical software." IET, 2009.
- [26] M. Bacic, "On hardware-in-the-loop simulation," in *44th IEEE Conference on Decision and Control, and the European Control Conference 2005*, 2005.
- [27] "Complete verification and validation for do-178c," White Paper, VECTOR, 2019.
- [28] A. Ray, C. Ackermann, R. Cleaveland, C. Shelton, and C. Martin, "Chapter 6 - functional and nonfunctional design verification for embedded software systems," ser. *Advances in Computers*, M. V. Zelkowitz, Ed. Elsevier, 2011, vol. 83, pp. 277 – 321. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/B9780123855107000060>
- [29] K. Nürnberger, M. Hochstrasser, and F. Holzapfel, "Execution time analysis and optimisation techniques in the model-based development of a flight control software," *IET Cyber-Physical Systems: Theory & Applications*, vol. 2, no. 2, pp. 57–64, 2017.