A SYSTEMATIC AND AGILE APPROACH TO DEVELOPING DO-178C COMPLIANT MODEL-BASED SAFETY-CRITICAL SOFTWARE

P. Panchal*, L. Hein*, S. Myschik*, F. Holzapfel§
*Institute for Aeronautical Engineering, University of the Bundeswehr Munich, 85521 Neubiberg, Germany
§ Institute of Flight System Dynamics, Technical University of Munich, 85748 Garching, Germany

Abstract

This paper presents a holistic safety-critical software development process according to DO-178C/DO-331 standards, which is being used at the Institute of Aeronautical Engineering at the University of the Bundeswehr Munich. A software development process for embedded systems is comprised of several steps, including requirements gathering and analysis, design, implementation, testing, integration, and maintenance of the components. When it comes to safety-critical applications, due to the safety standards and certification requirements, the cumbersomeness of these processes increases significantly. This is because the certification requirements demand that the system must fulfill certain objectives before it can come into use. Similarly, in the case of aerospace software applications, such as flight controllers or motor controllers, a set of objectives defined by the standards DO-178C must be fulfilled according to the criticality level detailed in the DAL (Design Assurance Level). To assist the development process, a systematic model-based toolchain is developed and implemented. This approach is presented in this research along with its application for battery controllers. The toolchain ensures required traceability of the artifacts and requirements-based verification and validation of the software.

1. INTRODUCTION

The aviation industry mandates compliance to the standards as a requirement to enter the software industry. To be compliant means to prioritize the safety of the passengers and crew of the aircraft while maintaining its efficiency. Aerospace industries thriving to cope up with the technology must ensure safety, incorporate flexibility in development, and integrate new features. However, these tasks contradict themselves, since the strict methodologies that are to be followed according to the standards restrict the development processes. New features mean increased lines of code, complexity, and criticality.

Hence, there is a significant need for developing integrated toolchains that can ensure flexibility of development like incorporating change in requirements along with ensuring traceability. On the other hand, this increases the cost of the project due to the number of tools and experienced engineers needed [1]. Especially for small-scale industries and startups without established processes, a toolchain helps in overcoming difficulties of maintaining artifacts and consistency throughout the development process. This information is usually the intellectual property of large-scale companies and is not readily available to the public hindering the development pace of small-scale industries.

Software development in the aviation industry often follows the generic V-Model process where the software requirements are derived from the system requirements and are then further implemented by the software logic followed by its tremendous testing. This structured approach then creates an infeasibility of incorporating changes at a later stage of development as it requires complete retesting and redevelopment of the software, resulting in increased time and cost. This problem is known as a ‘big-freeze’ problem [2].

In the context of the project ELAPSED [3], a novel electric propulsion system is being developed consisting of a battery based on a multilevel battery management system with a capacity of 17 kWh [4–6] for an electrically powered glider at the University of the Bundeswehr Munich. This battery then powers an electric motor with a power output of approximately 70 to 80 kW. For this propulsion system, a safety-critical battery controller and a motor controller is being developed. The toolchain developed in this research is applied for the development of the battery controller software currently and will be applied to develop the motor controller in the future. The software development for the battery and the motor controller of these components must adhere to the DO-178C/DO-331 [7, 8] standards when seeking certification. At the current development stage, only battery controller development is concerned, and the motor controller represents the future work of the project.

The paper is organized into the following chapters: Chapter 2 presents the overview of the complete toolchain developed in this research. It explains each stage of the development process with the required tools and application examples wherever necessary. Finally, chapter 3 and 4 shows the future work and a strong conclusion in the end respectively.

2. SOFTWARE DEVELOPMENT TOOLCHAIN

This section will explain the complete software development toolchain with the required tools and relevant examples.

2.1. Overview

Adopting the well-known V-Model in software development enhances software safety through verification and

doi: 10.25967/610224

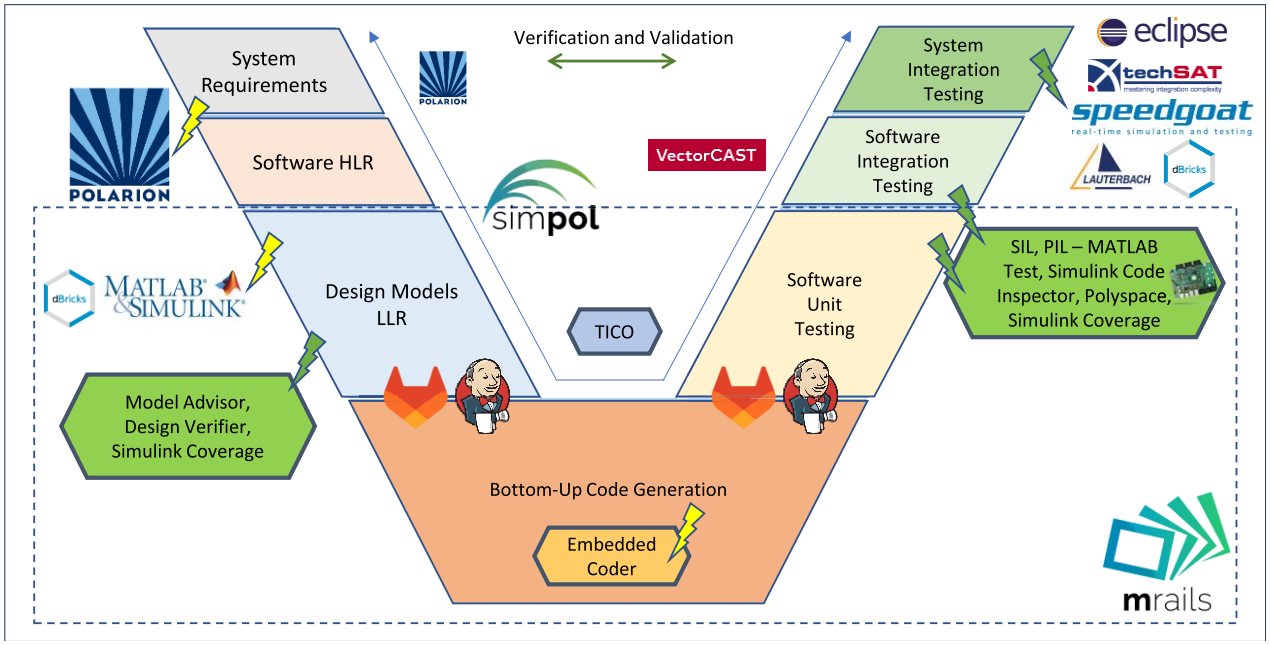


Fig. 1. V-Model for software development (functional part) along with the tools

validation conducted at various developmental phases. This methodology aids in the early detection of defects, and it also conforms to the DO-178C standards. When coupled with model-based software development, the V-Model not only amplifies the benefits of this systematic approach, but also provides the flexibility to integrate models as form of requirements, visually depict the software, facilitate automated code generation and verification, and improve collaboration within distributed development teams.

The software developed using the approach presented is divided into two parts: (1) functional code, which encompasses the logic, and (2) the application framework, which includes interfaces and low-level driver code. The model-based software development V-model of the functional part along with the tools is shown in Fig. 1.

In the case of application framework, only the code generation process and unit testing changes which is explained later. The process starts from defining the requirements of the shareholders or the customer. These requirements then define the objectives of the project and are then used to define the system requirements. Siemens Polarion PLM [9] tool is used to record the system requirements, software and hardware requirements, test cases, software and hardware development plans, change control and problem reporting.

From the derived requirements, hardware development follows a similar development process as software, but it is not discussed in this research as it is out of scope. The software low-level requirements are then derived from the software high-level requirements. Functional low-level requirements are in the form of design models developed in MATLAB/Simulink. The traceability of all the artifacts is mentioned later. For the development of the design models, a process-oriented build tool is used which is called as Mrails [10–15]. This build-tool provides a development framework in MATLAB/Simulink [16] featuring modular model-based development methodologies.

The build-tool is being implemented and developed at the Institute for Aeronautical Engineering, University of the Bundeswehr Munich, and the Institute of Flight System Dynamics, Technical University of Munich. The build-tool also provides automatic jobs that uses Embedded Coder to generate code for the functional part to the software in MATLAB. Modeling is performed according to the guidelines of DO-331. Certain modeling guidelines are set, and static model analysis is also performed using the build-tool in MATLAB. In case of the application software, the low-level software requirements are stored in Polarion in textual form, like the high-level requirements. The code for application software part is written manually in Eclipse IDE [17].

After the code is generated, the static testing is performed using Polyspace [18] whereas the simulation testing is performed using Simulink Tests and VectorCAST [19] for the functional and application part respectively. Since model-based approach is used for developing the functional part, the advantages of model-based testing, model-in-the-loop (MIL), software-in-the-loop (SIL), processor-in-the-loop (PIL) and hardware-in-the-loop (HIL) can be leveraged easily.

The functional software is then integrated into the application framework in the Eclipse project. This software is then tested in real-time using a HIL testbench presented in [20]. The modular HIL testbench allows to exchange the hardware and perform rigorous testing. For testing, requirements-based test cases are used which ensures required software coverage is achieved. For PIL testing, Lauterbach Trace32 debugger [21] setup is used that provides integration with Simulink.

The interfaces between software and hardware are realized in an interface management tool called dBricks, which contains automated data integrity and correctness checks to ensure unambiguous description of interfaces. In terms of traceability of all the artifacts from requirements stage to the testing stage, several tools are used. The system and software high-level requirements are stored in Polarion

itself, so a hyperlink ensures the bidirectional linkage. The low-level requirements of the functional software which is the design model in Simulink is linked to the high-level requirements in Polarion using a tool called SimPol [22]. The traceability between the model, code and its test case are ensured within the Simulink environment. The application code is linked to the requirements via the test cases defined in VectorCAST. To ensure an agile workflow, Git is used as a version control software and a structured workflow is followed. A continuous integration server based on Jenkins [23] is setup and used for automated testing and providing closed-loop requirements verification. The following sub chapters contain detailed explanations of the development stages.

2.2. System Requirements and Design Process

The requirements are stored in Polarion tool which is also used for other tasks like change control, problem reporting, creating baselines and for writing plans. Fig. 2 shows a snapshot of software high-level requirements and
Fig. 3 shows a snapshot of plans for safety aspects of certification. In this manner, all the other development and verification plans are stored.
Fig. 4 shows a part of software configuration index created in Polarion that houses all the plans and connects them

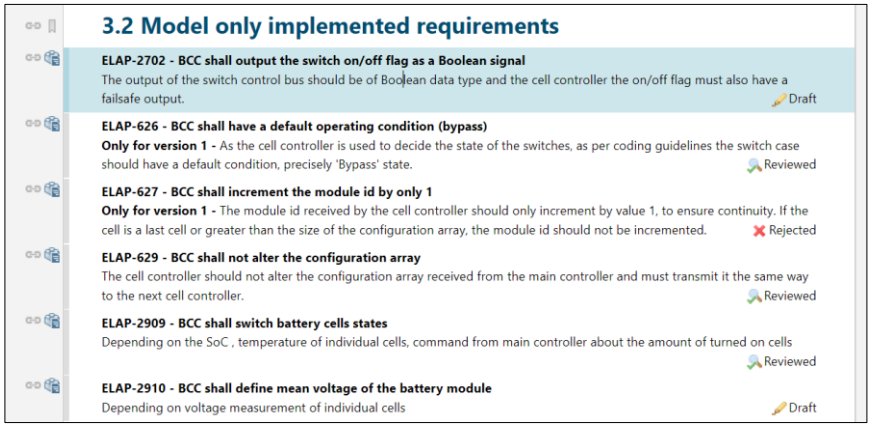


Fig. 2. Software High-Level Requirements in Polarion

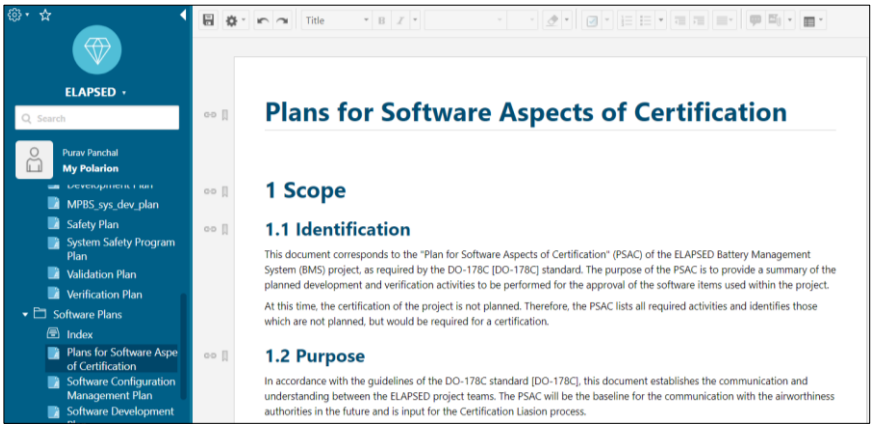


Fig. 3. PSAC report in Polarion

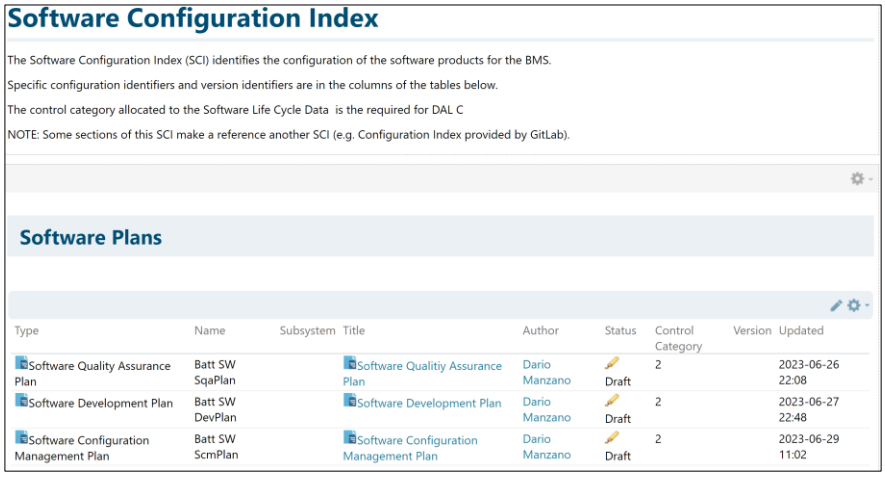
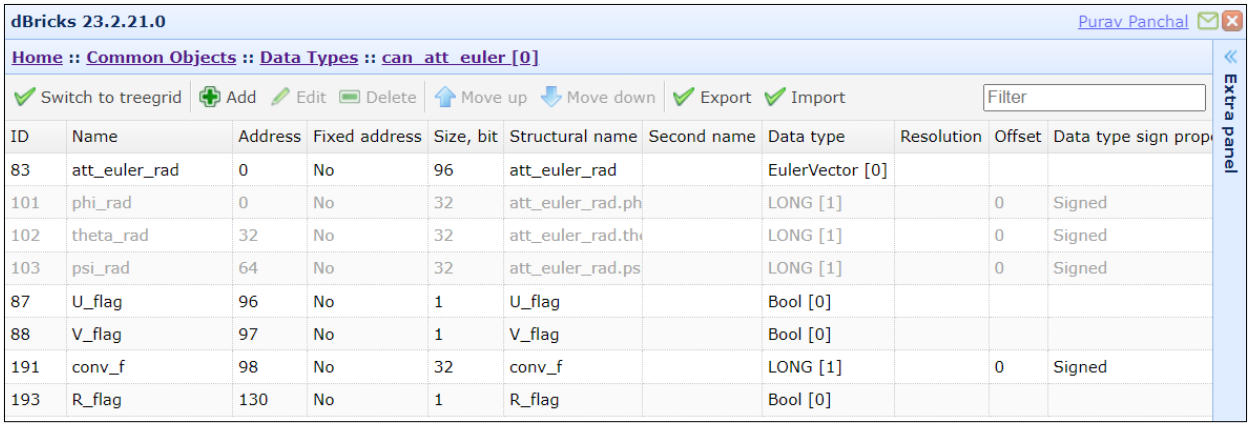


Fig. 4. Software Configuration Index in Polarion



dBricks 23.2.21.0 — Purav Panchal

Home :: Common Objects :: Data Types :: can_att_euler [0]

Switch to treegrid | Add | Edit | Delete | Move up | Move down | Export | Import | Filter | Extra panel

ID	Name	Address	Fixed address	Size, bit	Structural name	Second name	Data type	Resolution	Offset	Data type sign prop
83	att_euler_rad	0	No	96	att_euler_rad		EulerVector [0]			
101	phi_rad	0	No	32	att_euler_rad.ph		LONG [1]		0	Signed
102	theta_rad	32	No	32	att_euler_rad.th		LONG [1]		0	Signed
103	psi_rad	64	No	32	att_euler_rad.ps		LONG [1]		0	Signed
87	U_flag	96	No	1	U_flag		Bool [0]			
88	V_flag	97	No	1	V_flag		Bool [0]			
191	conv_f	98	No	32	conv_f		LONG [1]		0	Signed
193	R_flag	130	No	1	R_flag		Bool [0]			

Fig. 5. Snapshot of a signal data type stored in dBricks

using hyperlinks. The system requirements are based on CS-22 requirements and are presented in [24, 25]. The software and hardware high-level requirements have their own 'work-item' type in Polarion which helps in creating links between them. The traceability of requirements is explained in the later chapter.

2.3. Software Design Process

A tool dBricks is used for managing the interfaces used in the component. It provides an interface to Simulink which can import the data buses defined in dBricks to the Simulink environment by creating a data dictionary that can be easily referenced. Fig. 5 shows a snapshot of dBricks tool containing a signal '`can_att_euler`' that can be imported in Simulink as 'bus'.

The system requirements as described in ARP4754A [26] are further churned down to item level. Item level requirements can be software, hardware or even both. In the case of embedded products, both the software and hardware requirements are to be well documented. This research emphasizes more on the software part and hence the hardware development is less focused on.

As mentioned earlier, the software is divided into functional and application parts. Low-level software requirements for the functional part are represented by the design models in Simulink which offers a direct route to generate code using the Embedded Coder.

The design models explain not only the functionality of the software but also the data structures and how different functions are related to each other. As the models merely explain the control laws, a simulation of the low-level requirements is possible followed by verification and validation. The DO-331 [8] offers several possible use-cases of models in software development and in the concerned research, the design models are used as low-level requirements which are derived from the high-level requirements (see table MB. 1-1 of [8]). Specification models are not used, and the high-level requirements are represented in textual form.

Design models that represent functional low-level requirements provide several advantages like automatic code generation, deterministic code, readable code, traceable code, configurable in terms of coding guidelines, verification of models against modeling and coding guidelines, requirements verification, and object code verification against requirements [27].

A process-oriented build tool, Mrails, is used to design the models in Simulink providing significant advantages in terms of efficiency. Advantages of this model-based process-oriented build tool is mentioned in several research papers - [10–15]. In a nutshell, the tool provides a development framework in MATLAB/Simulink with several automated tasks for designing and code generation. It also provides process-oriented features like custom library blocks, design and traceability review checklists and custom Model Advisor checks. The tool also provides an HTML based status report that houses all the results of the design and code verification tasks. These automated jobs are shown in Fig. 8.

Application software in this research is not developed using model-based technology and hence the low-level requirements are stored in textual form in Polarion. The application software is designed in the Eclipse IDE, specifically like STM32CubeIDE [28] and NXP S32DS [29].

2.4. Code Generation

```
>> mrails jobs
[INFO]: -- design-verification::do331.jobs.StaticModuleAnalysis
[INFO]: -- design-verification::do331.jobs.StaticModelAnalysis
[INFO]: -- design-verification::do331.jobs.DesignErrorDetection
[INFO]: -- design-verification::do331.jobs.TraceabilityReview
[INFO]: -- design-verification::do331.jobs.ModelReview
[INFO]: -- design-verification::do331.jobs.ModelMetrics
[INFO]: -- simulation-testing::do331.jobs.SimulationCaseProcedureReview
[INFO]: -- simulation-testing::do331.jobs.SimulationCaseExecution
[INFO]: -- simulation-testing::do331.jobs.SimulationResultReview
[INFO]: -- simulation-testing::do331.jobs.ModelCovAggregation
[INFO]: -- simulation-testing::do331.jobs.ModelCoverageAnalysis
[INFO]: -- simulation-testing::do331.jobs.ModelCallCoverageAnalysis
[INFO]: -- code::do331.jobs.GenerateSharedCode
[INFO]: -- code::do331.jobs.GenerateCode
[INFO]: -- code::do331.jobs.PackageCode
[INFO]: -- code-verification::do331.jobs.InspectCode
[INFO]: -- code-verification::do331.jobs.CheckCodeCompliance
[INFO]: -- code-verification::do331.jobs.CodeDefectAnalysis
[INFO]: -- code-verification::do331.jobs.CodeProving
[INFO]: -- SIL-testing::do331.jobs.SILTestCaseExecution
[INFO]: -- SIL-testing::do331.jobs.SILCovAggregation
[INFO]: -- SIL-testing::do331.jobs.SILCoverageAnalysis
[INFO]: -- documentation::do331.jobs.GenerateDocumentation
```

Fig. 6. Automated model and code development jobs provided by the process-oriented build tool

The model-based functional code is then integrated into the application code written manually as shown in Fig. 7.

Example of a model-based functional code integrated into application framework in Eclipse IDE The model 'initialize' and 'step' function is called in the main function and the inputs and outputs of the step function are represented by *<model-name>_U* and *<model-name>_Y* respectively.

The Simulink code is packed and added to the path of the Eclipse project which takes care of adding the includes and source. The process-oriented build tool provides automated job for code generation which follows a bottom-up code generation process where the regeneration of shared code is avoided. The code generation process is explained in detailed in [12].

Advantages of automatic code generation are advanced verification and validation techniques, eliminating human errors, improves readability and the traceability to the

```
int main(void) {

dcu_initialize();
dcu_U.can_dcu_en_Bus.V_flag = 1;
dcu_U.can_dcu_en_Bus.U_flag = 1;
dcu_U.can_dcu_en_Bus.EN = 1;
dcu_U.can_dcu_en_Bus.conv_f = 1;
dcu_U.can_att_euler_Bus.V_flag = 1;
dcu_U.can_att_euler_Bus.U_flag = 1;
dcu_U.can_att_euler_Bus.conv_f = 1000;

  for (;;)
  {

    dcu_U.can_att_euler_Bus.att_euler_rad.phi_rad = 500;
    dcu_U.can_att_euler_Bus.att_euler_rad.theta_rad = 500;
    dcu_U.can_att_euler_Bus.att_euler_rad.psi_rad = 500;
    dcu_step();
    M11 = dcu_Y.fc_ctrl_cmd_k.M_OB.M1.x1;
    M12 = dcu_Y.fc_ctrl_cmd_k.M_OB.M1.x2;
    M13 = dcu_Y.fc_ctrl_cmd_k.M_OB.M1.x3;
    M21 = dcu_Y.fc_ctrl_cmd_k.M_OB.M2.x1;
    M22 = dcu_Y.fc_ctrl_cmd_k.M_OB.M2.x2;
    M23 = dcu_Y.fc_ctrl_cmd_k.M_OB.M2.x3;
    M31 = dcu_Y.fc_ctrl_cmd_k.M_OB.M3.x1;
    M32 = dcu_Y.fc_ctrl_cmd_k.M_OB.M3.x2;
    M33 = dcu_Y.fc_ctrl_cmd_k.M_OB.M3.x3;
    __asm volatile ("svc 0");
  }
  return 0;
}
```

Fig. 7. Example of a model-based functional code integrated into application framework in Eclipse IDE

requirements.

2.5. Verification and Validation

Validation of the software requirements is not explained in detail in DO-178C as it occurs at system level. However, incomplete requirements could lead to validation failures. To understand the difference between validation and verification, one can resemble it with two questions: is the right aircraft being built? – (*validation*); is the aircraft being built correctly? – (*verification*).

Verification of the software is essentially the right part of the software development V-model. Since the functional code is developed using a model-based approach, it can utilize the powerful verification mechanisms provided by the tools like MATLAB for XIL (x-in-the-loop) simulations for performing requirements-based testing as needed by the DO-178C. Model verification includes checks, compliance against standards, and model coverage. The process-oriented build tool provides jobs to automate the static analysis and simulation testing of the functional part of the software using the required tools in MATLAB/Simulink,

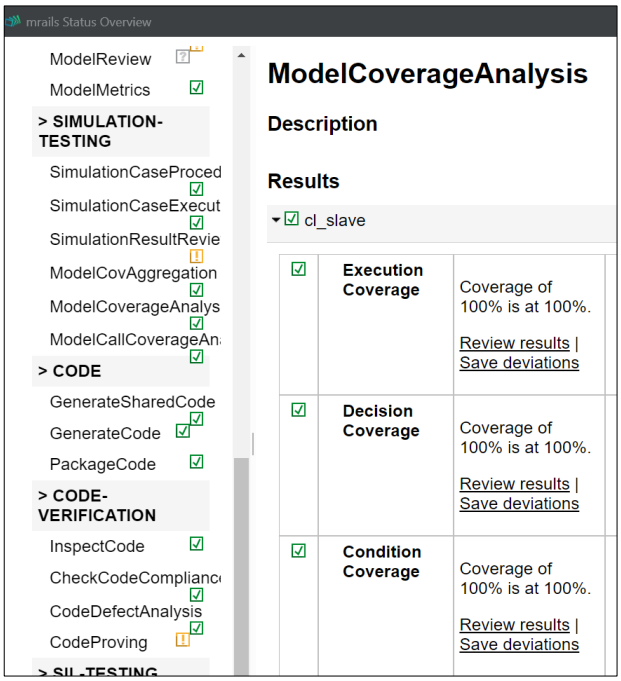


Fig. 8. Status report of the process-oriented build tool

covering the model-in-the-loop (MIL) and software-in-the-loop (SIL) aspects. The source-code is verified using the Polyspace Code Prover, Bug Finder and simulation testing like SIL and PIL.

PIL is performed with the Trace32 debugger toolchain provided by Lauterbach GmbH. The debugger has an integration package for Simulink which basically provides a toolchain for the Embedded Coder to generate the PIL code and then executes it on the target using its own setup. This reduces the effort of setting up the target hardware since the startup script required by the Trace32 software can be reused from the debugging environment.

Fig. 8 shows a snapshot of the HTML status window of the process-oriented build tool showing accumulated model coverage result. Static analysis and unit testing of the application framework of the software is performed using the Polyspace and VectorCAST tools.

For the object code verification, the real-time testing of the software is done using hardware-in-the-loop simulations. For this the integrated object code is generated and loaded on the hardware and then Simulink real-time capabilities is used to perform HIL simulations.

In this work, Simulink baseline performance machine and TechSAT real-time PC with ADS tools are used in conjunction with each other. A modular HIL testbench is created that can be used to verify software in real-time as shown in Fig. 9. Initial HIL results of the battery cell controller application are discussed in [20]. The nonlinear flight dynamics model of the electric aircraft [24] is simulated on the real-time system RTPC-1 which is connected to a Speedgoat machine RTPC-2 which simulates the power electronics, motor and battery components. Communication protocols are shown in the legend of the Fig. 9.While RTPC-1 is suitable for achieving simulation frame rates of up to 2 KHz, RTPC-2 is

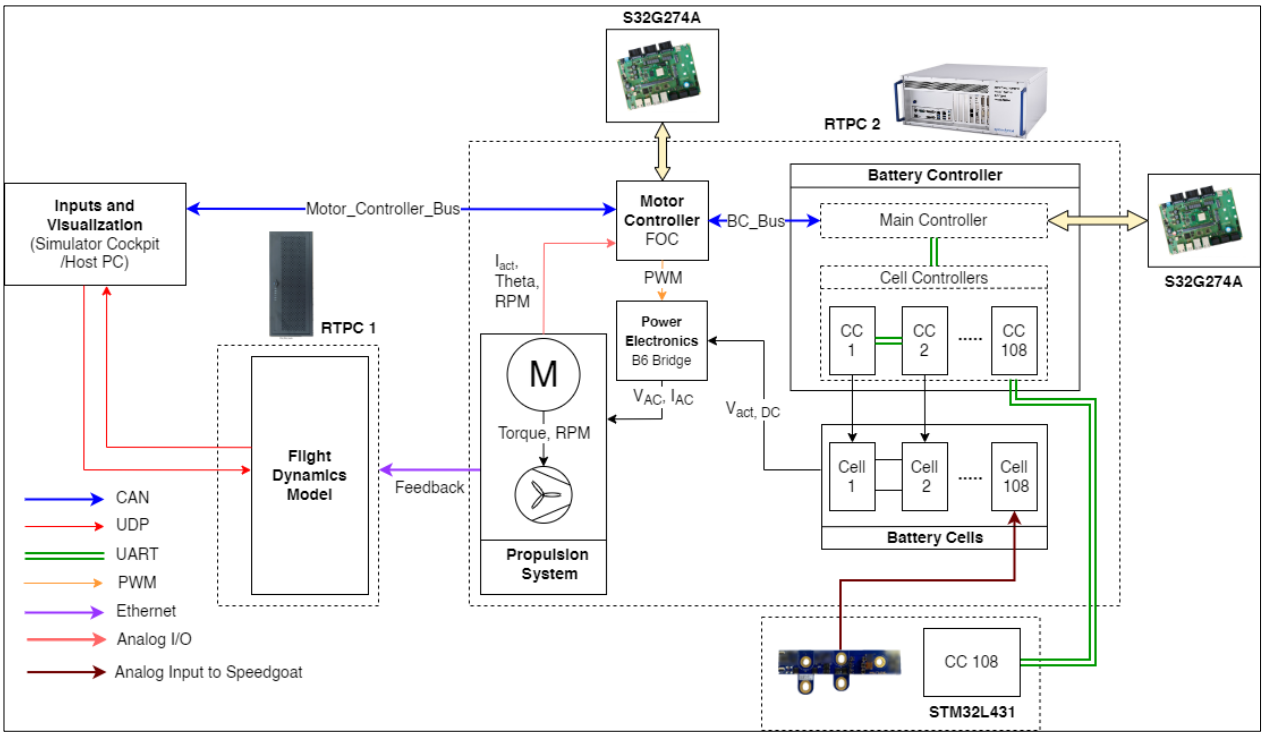


Fig. 9. Modular Hardware-in-the-Loop Testbench for a Battery Controller Software Application

specifically designed to simulate electrical power systems with up to 200 KHz. This is achieved by an integrated FPGA, which can be targeted directly from Simulink. This modular HIL testbench allows for easy switching between non-real-time and real-time simulation using a variant subsystem block in the integration model. The variant subsystem block contains the interface subsystem for real-time simulation and the model subsystem for non-real-time simulation. The test cases are derived from requirements and are linked to the model and code as discussed in the next chapter. This ensures that requirements-based testing is achieved.

Requirements-based simulation testing is validated directly from its specification document in Polarion. To facilitate this process, a Continuous Integration (CI) server based on Jenkins is set up in conjunction with GitLab for version control. Polarion offers seamless integration with Jenkins,

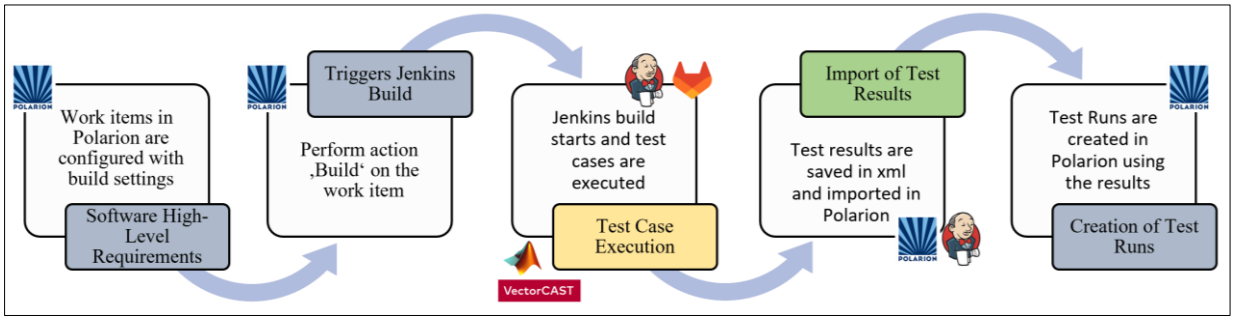


Fig. 10. Software Requirements Validation Worklfow with CI

Fig. 11. Test Runs in Polarion Imported from Jenkins Build

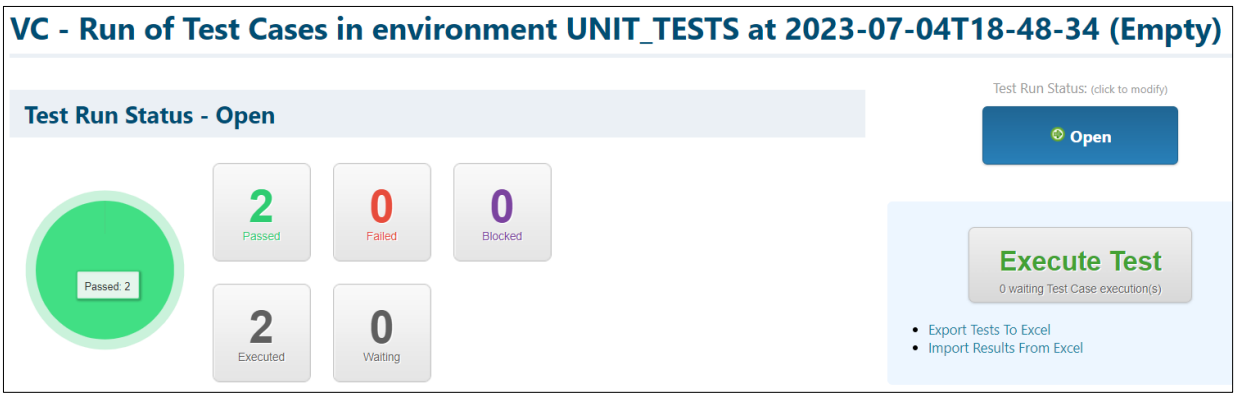


Fig. 12. Test Runs in Polarion Imported from VectorCAST

allowing builds to be triggered directly from requirements and importing the test results.

Fig. 10 illustrates this workflow, where the action 'build' is performed on the requirement work item, triggering the Jenkins build. The Jenkins build is configured to execute simulation test cases in MATLAB and generate an XML-based result artifact. Once the build is completed, the artifact is retrieved by the test run workflow in Polarion, automatically creating a test run using the predefined template in the configuration. Fig. 11 showcases the Jenkins built results imported in Polarion. Jenkins also supports integration with VectorCAST, enabling the execution of unit tests for the handwritten source code in the application framework. This integration allows Jenkins to be triggered from Polarion, and the results can be imported back into Polarion.

VectorCAST also provides seamless integration with Polarion, and the test runs from VectorCAST can be exported to Polarion as shown in Fig. 12 which shows an exemplary test run.

2.6. Artifacts Traceability

According to DO-178C, trace data must be established between system requirements allocated to software (SRATS) and high-level requirements. The traceability concept implemented along the V-Model in the software development approach is shown in Fig. 13. The figure emphasizes the software part related to DO-178C. The subsystem requirements allocated to software high-level requirements are linked bi-directionally using the built-in linkage feature in Polarion. The linkage type is indicated in the figure's legend. The low-level software requirements for the application framework are also stored in Polarion and utilize its linkage for maintaining bidirectional traceability.

However, the low-level software requirements of the functional part are presented through design models in Simulink. Therefore, these models are linked to their parent high-level requirements using the SimPol tool. SimPol

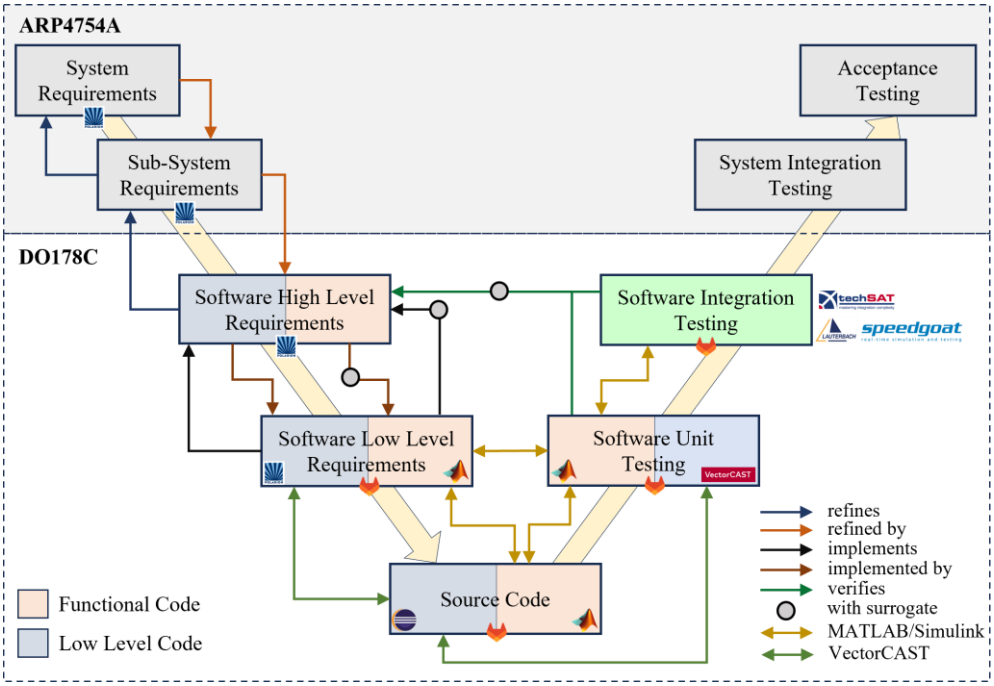


Fig. 13. Traceability Concept along the V-Model Based Software Development

creates a surrogate item in Polarion, which contains hyperlinks to the Simulink model and the Git repository. Bi-directional traceability between the automated code generated by Embedded Coder and the design model is ensured by Simulink itself. For hand-written code, the VectorCAST tool is used to link the test cases for the source code to their low-level requirements in Polarion.

The test cases for functional parts are created and executed using MATLAB Tests, which leverage the same infrastructure for maintaining traceability to low-level requirements. The MATLAB test cases are also linked to the software high-level requirements using the 'verifies' linkage type, again by creating a surrogate item using SimPol. The functional part of the battery management system is integrated and tested in real-time using Simulink Real-Time and a Speedgoat machine, also utilizing MATLAB Tests. The trace is then established using SimPol and surrogates between the system integration testing and software high-level requirements in Polarion.

3. FUTURE WORK

The software development toolchain presented in this paper is currently being applied to develop the battery controller for the project ELAPSED and will also be used to develop the motor controller in future. The development of the battery controller is under progress and hence not all the results have been achieved. After the completion of the battery controller software, the toolchain will be further concretized, and the advantages can be seen more evidently. Future work of this research also includes creating templates of projects in tools like Jenkins, VectorCAST, Polarion, and dBricks. This will help as a starting point for developing new software.

4. CONCLUSIONS

The research presents a complete software development toolchain used at the Institute for Aeronautical Engineering in the University of the Bundeswehr Munich. This toolchain is used for developing safety-critical software like flight controller, battery, and motor controller. The research is funded by the project ELAPSED in which a novel propulsion system is being developed with a multilevel battery system. Hence, the toolchain is used to develop the battery and motor controller for the propulsion system.

The paper presents the required tools and emphasizes the software development process and its different stages. The process starts with the requirements management which is managed in Polarion tool. The functional part of the software is developed using a model-based approach. The low-level functional software requirements are represented using the design models and the low-level application software requirements are represented in textual form.

MATLAB/Simulink is used for developing model-based software and automatic code generation followed by its verification whereas Eclipse IDE is used for the development of application software. Polyspace and VectorCAST are used for unit and static tests. Real-time testing is performed using Simulink real-time environment with TechSAT real-time system. Interfaces are managed separately using a tool, dBricks, which can create models for interfaces, integrate into Simulink and provides input-output mapping for real-time simulations.

Bidirectional traceability of most of the artifacts are handled within these tools. For traceability between software low-level requirements and high-level requirements is ensured using additional tools like SimPol and VectorCAST. The entire software testing process is based on requirements-based testing as it is mandatory by DO-178C. This holistic software development process not only provides consistency during the development but can also accommodate changes in requirements with the help of agile tools like Git and Jenkins.

This research is funded by dtec.bw – Digitization and Technology Research Center of the Bundeswehr [3].

5. REFERENCES

[1] L. Rierson, *Developing safety-critical software: A practical guide for aviation software and DO-178C compliance / Leanna Rierson*. Place of publication not identified: CRC Press, 2013.

[2] J. Cleland-Huang, A. Agrawal, M. Vierhauser, and C. Mayr-Dorn, “Visualizing Change in Agile Safety-Critical Systems,” *IEEE Softw.*, vol. 38, no. 3, pp. 43–51, 2021, doi: 10.1109/MS.2020.3000104.

[3] dtec.bw. “Electric Aircraft Propulsion – die Zukunft der Flugzeugantriebe.” https://dtecbw.de/home/forschung/unibw-m/projekt-elapsed

[4] Manuel Kuder, Julian Schneider, Anton Kersten, Torbjörn Thiringer, Richard Eckerle, Thomas Weyh, “Battery Modular Multilevel Management (BM3) Converter applied at Battery Cell Level for Electric Vehicles and Energy Storages,” 2020.

[5] N. Sorokina *et al.,* “Inverter and Battery Drive Cycle Efficiency Comparisons of Multilevel and Two-Level Traction Inverters for Battery Electric Vehicles,” in *2021 IEEE International Conference on Environment and Electrical Engineering and 2021 IEEE Industrial and Commercial Power Systems Europe (EEEIC / I&CPS Europe)*, Bari, Italy, 2021, pp. 1–8, doi: 10.1109/EEEIC/ICPSEurope51590.2021.9584705.

[6] J. Buberger *et al.,* “Charging Strategy for Battery Electric Vehicles with a Battery Modular Multilevel Management (BM3) Converter System using a PR controller,” in *2021 23rd European Conference on Power Electronics and Applications (EPE'21 ECCE Europe)*, Ghent, Belgium, uuuu-uuuu, P.1-P.10, doi: 10.23919/EPE21ECCEEurope50061.2021.9570669.

[7] *DO-178C - Software Considerations in Airborne Systems and Equipment Certification,* RTCA, 2011. [Online]. Available: https://my.rtca.org/productdetails?id=a1B36000001IcmqEAC

[8] *DO-331 - Model-Based Development and Verification Supplement to DO-178C and DO-278A,* RTCA, 2011. [Online]. Available: https://my.rtca.org/nc__store?search=331

[9] Siemens, *Polarion PLM Automation*. https://polarion.plm.automation.siemens.com/: Siemens. [Online]. Available: https://polarion.plm.automation.siemens.com/

[10] M. Hochstrasser, S. Myschik, and F. Holzapfel, “A Process-oriented Build Tool for Safety-critical Model-based Software Development,” in *Proceedings of the 6th International Conference on Model-Driven Engineering and Software Development*, Funchal,

Madeira, Portugal, 2018, pp. 191–202, doi: 10.5220/0006605301910202.
[11] 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* (Communications in Computer and Information Science), S. Hammoudi, L. F. Pires, and B. Selic, Eds., Cham: Springer International Publishing, 2019, pp. 380–405.
[12] P. Panchal, S. Myschik, K. Dmitriev, P. Bhardwaj, and F. Holzapfel, Eds., *Handling Complex System Architectures with a DO-178C/DO-331 Process-Oriented Build Tool*. 2022, 2022.
[13] P. Panchal, S. Myschik, K. Dmitriev, and F. Holzapfel, “Application of a Process-Oriented Build Tool to an INDI-Based Flight Control Algorithm,” in *AIAA AVIATION 2022 Forum*, Chicago, IL & Virtual, 2022, doi: 10.2514/6.2022-4092.
[14] P. Panchal, N. Sorokina, S. Myschik, K. Dmitriev, and F. Holzapfel, “Application of a Process-Oriented Build Tool to the Development of a BM3 Slave Controller Software Module,” 2021. doi: 10.25967/570308. [Online]. Available: https://doi.org/10.25967/570308
[15] P. Panchal, N. Sorokina, M. Kuder, S. Myschik, K. Dmitriev, and F. Holzapfel, “Application of a Process-Oriented Build Tool for Verification and Validation of a Battery Slave Controller for a Battery Modular Multilevel Management System Along the DO-178C/DO-331 Process,” in *Proceedings of the 11th International Conference on Model-Based Software and Systems Engineering*, Lisbon, Portugal, 2023, pp. 184–193, doi: 10.5220/0011696100003402.
[16] *MathWorks - Entwickler von MATLAB und Simulink.* Accessed: Nov. 8, 2022. [Online]. Available: https://de.mathworks.com/?s_tid=gn_logo
[17] The Eclipse Foundation. “Eclipse Desktop & Web IDEs.” https://www.eclipse.org/ide/ (accessed Mar. 30, 2023).
[18] MathWorks, *Polyspace*. MathWorks. [Online]. Available: https://www.mathworks.com/products/polyspace.html
[19] Vector. “VectorCAST.” https://www.vector.com/int/en/products/products-a-z/software/vectorcast/ (accessed Mar. 30, 2023).
[20] P. Panchal, W. Bliemetsrieder, N. Sorokina, and S. Myschik, “Real-Time Verification of A Battery Slave Controller Developed Using a DO-178C/DO-331 Based Process-Oriented Build Tool,” in *AIAA AVIATION 2023 Forum*, San Diego, CA and Online, 2023, doi: 10.2514/6.2023-3992.
[21] Lauterbach. “Microprocessor Development Tools.” https://www.lauterbach.com/frames.html?home.html (accessed Mar. 30, 2023).
[22] FSD, *SimPol - Simulink® – Polarion® Connector*. https://www.fsd.lrg.tum.de/software/simpol/: TUM. [Online]. Available: https://www.fsd.lrg.tum.de/software/simpol/
[23] CloudBees. “Jenkins - Build great things at any scale.” https://www.jenkins.io/ (accessed Mar. 30, 2023).
[24] L. Hein and S. Myschik, “Simulation of an Electric Powered Aircraft for Flight & Mission Performance Evaluation,” in *AIAA AVIATION 2022 Forum*, 06272022, doi: 10.2514/6.2022-3572.
[25] L. Hein, P. Panchal, and S. Myschik, Eds., *Certification Compliant Performance Analysis and Requirements Management of an Electrically Powered General Aviation Aircraft*, 2023.
[26] S. A. ARP4754A, *Guidelines for Development of Civil Aircraft and Systems. 2010*. RAS.
[27] Ulrich Eisemann, “Applying Model-Based Techniques for Aerospace Projects in Accordance with DO-178C, DO-331, and DO-333,” in 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:201691706
[28] STMicroelectronics, *STM32CubeIDE - Integrated Development Environment for STM32 - STMicroelectronics.* [Online]. Available: https://www.st.com/en/development-tools/stm32cubeide.html
[29] NXP, *S32 Design Studio IDE*. [Online]. Available: www.nxp.com/design/software/development-software/s32-design-studio-ide:S32-DESIGN-STUDIO-IDE