S. Götz, L. Linsbauer, I. Schaefer, A. Wortmann (Hrsg.): Software Engineering 2021 Satellite Events,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2021 1

A Requirements Management Template in Polarion for Model-Based Development of Airborne Systems

Kevin Schmiechen[1], Shanza Ali Zafar[2], Konstantin Dmitriev[3], Christoph Krammer[4], Markus Maly[5], Florian Holzapfel[6]

Abstract: With numerous startups and small companies entering the market of air mobility, safety-critical system development has become relevant for an increasing number of project teams with little or no experience in that field. In a previous publication, we presented a custom software development workflow for research and prototype development based on objectives from industry standards DO-178C and DO-331. An important part of this custom workflow is requirements management since poor requirements and a lack of proper traceability has been attributed as a major source of project delays and increased costs. This paper presents the setup of an extended data model in the application lifecycle management platform *Polarion* including the artifact types and their attributes. Exemplary key workflows are demonstrated. It also provides more details regarding the linking strategy between the textual requirements and model artifacts in the *MathWorks* toolchain, as well as the trace report for the requirements including links to model artifacts and source code. The setup has proven its functionality in multiple projects at our institute without significant rework.

Keywords: lean software development, safety-critical systems, model-based development, requirements management, requirements engineering, data model, traceability

1 Introduction

Requirements engineering is undoubtedly an important part for the successful development of safety-critical systems and software. Many projects cite poorly defined requirements as the source of project delays and increased costs [FM15; LH16; Ri17]. As the complexity and criticality of the systems increase, it is extremely difficult to fulfill the goals of the development project without properly defined and managed requirements along with a

[1] Technical University of Munich, Institute of Flight System Dynamics,
Boltzmannstr. 15, 85748 Garching, Germany, kevin.schmiechen@tum.de
[2] Technical University of Munich, Institute of Flight System Dynamics,
Boltzmannstr. 15, 85748 Garching, Germany, shanza.zafar@tum.de
[3] Technical University of Munich, Institute of Flight System Dynamics,
Boltzmannstr. 15, 85748 Garching, Germany, konstantin.dmitriev@tum.de
[4] Technical University of Munich, Institute of Flight System Dynamics,
Boltzmannstr. 15, 85748 Garching, Germany, christoph.krammer@tum.de
[5] Technical University of Munich, Institute of Flight System Dynamics,
Boltzmannstr. 15, 85748 Garching, Germany, markus.maly@tum.de
[6] Technical University of Munich, Institute of Flight System Dynamics,
Boltzmannstr. 15, 85748 Garching, Germany, florian.holzapfel@tum.de

user-friendly platform to work with the artifacts. For the aircraft development process, there are multiple standards about system and software requirements engineering and how to implement them in the overall process (e.g., IEEE 29148:2018 [In18a], ARP-4754A [So10], DO-178C [Ra11a]). However, full compliance with those standards is expensive and complex; not only for the setup but also during the development process itself. Our institute, the Institute of Flight System Dynamics at the Technical University of Munich, developed "A Lean and Highly-Automated Model-Based Software Development Process Based on DO-178C/DO-331" [Dm20] to simplify the airborne software development process for research and prototype aircraft, where full compliance is not necessary, but a certain quality of software and basis for future certification may be needed. This process also includes requirements management due to the importance of requirement related objectives. The current paper provides a more detailed and extended view on the setup of the requirements management, the artifact types, their attributes and workflows in the context of the process in [Dm20]. It also further explains the linking strategy between requirements and model elements as well as automatic trace reports. However, this paper does not describe the process of requirement elicitation, meaning how to identify necessary requirements and how to write them. There are already numerous sources (e.g., [FM15; LM09; LT08; Po10]) available that cover this topic.

Unlike [KK19], we decided not to develop our own requirements management tool but to use the commercial and widely adopted application lifecycle management tool *Polarion*[7] by *Siemens Digital Industry Software*. It is a web-based platform that runs on a server, thus only needs to be setup once for a company and can be used by all developers simply from their web browser. There is also a tool qualification kit for ISO-26262-8 [In18b] which can be modified for DO-178C and applied to qualify the use of *Polarion* in one's toolchain.[8] This is only one benefit over a self-developed management tool. *Polarion* also fulfills all high priority requirements for requirements management tools from [Ho04] except for data encryption on the server, the ability to check-out artifacts for offline editing and automatic content updates in the user interface when multiple users are simultaneously working on the same object. *Polarion* has been previously used by our institute [MSH19; Sc19]. The tool is shipped with several template and demo projects (e.g., V-model, CMMI, agile) and further templates are available from the extensions portal[9]. However, in order to use the project templates for our purposes, we modified them heavily. These adaptations are described in this paper, thereby distinguishing it from a straightforward tool application report. This includes modification of the data model (e.g., requirement types, attributes and workflows; link roles; document types) as well as automatic processing of the contained information. At the same time, we see the possibility of modifying the project templates as a major advantage of this tool. *Polarion* allows the developers to set up the implementation according to their needs. A data model tailored to *Polarion* for conformity assessment in the context of nuclear power plants has been presented in [LA19].

[7] https://polarion.plm.automation.siemens.com/
[8] https://extensions.polarion.com/extensions/315-tool-qualification-kit
[9] https://extensions.polarion.com/

The other tools we used in conjunction with the requirements management process are *MATLAB*[10]*/Simulink*[11] by *The MathWorks* for model-based development, and *SimPol*[12], which was developed at our institute and offers bidirectional traceability between requirements in *Polarion* and models and test cases in *MATLAB/Simulink* [HMH18; MSH19; Sc19]. We also wrote a user guide that provides step-by-step instructions for our requirements management workflows. However, the user guide will not be described in this paper.

The remaining sections of the paper are structured as follows. In section 2, the artifact data model consisting of the artifact types, the corresponding attributes and the linking between the different artifact types is described. Based on that, key workflows of the requirements themselves, certification references and question lists for the customer or project partner are shown in section 3. Section 4 gives a detailed description of the linking strategy between the requirements in *Polarion* and the different types of models in *MATLAB/Simulink*. The features of the automatic trace reports are presented in section 5. Finally, section 6 concludes the presented setup and section 7 gives an outlook of future work.

2 Data Model

Artifacts in *Polarion* are called work items. By specifying the name of a work item type, its attributes and workflow, work items can be configured to represent almost anything in the development process. This includes requirement types on different levels, configuration indexes, tasks, test cases, issues etc. The custom link roles define the relationship between different work item types and are used for traceability. The work items can be edited individually or as part of a document in *Polarion* that contains multiple work items. This allows users to create, e.g., a requirement specification document that is considered as one unit and can easily be shared for collaboration or exported as a PDF file. Documents have workflows and attributes as well.

2.1 Work Item Types and Link Roles

Fig. 1 shows the setup of the work item types and their link roles. The bold framed boxes indicate work item types that were already presented in [Dm20]. The thin framed boxes indicate work item types of an extended setup that may be optional depending on the project. Dotted boxes are used to cluster work item types and thus reduce the number of links in Fig. 1. We followed the classical V-model approach, structuring high-level customer requirements by breaking them down into aircraft, system, component and software requirements where each level refines the previous level (indicated by blue arrows). Independent of the development project scope, one requirement type always addresses only one level. Aircraft requirements

[10] https://www.mathworks.com/products/matlab.html
[11] https://www.mathworks.com/products/simulink.html
[12] https://www.fsd.lrg.tum.de/software/simpol/

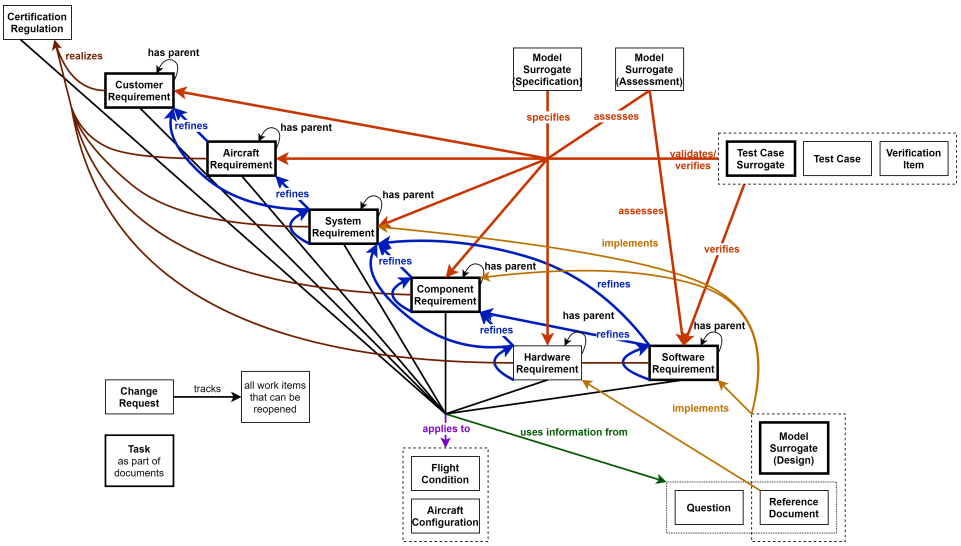

Fig. 1: Work Item Types and Link Roles

address the aircraft as a whole. System requirements relate to specific systems of the aircraft (e.g., flight control system, propulsion system, high-lift system). Component requirements consider subsystems (e.g., flight control computer of flight control system, fan of propulsion system, flap of high-lift system).

A different approach was realized in the *System of Systems*[13] project template for *Polarion*, where everything is considered a system. To identify the layout of the different systems, they are linked in a tree structure. This approach requires less work item types and thus results in a lean *Polarion* project setup, with the disadvantage that it may be more difficult to identify the requirement level one is currently working on.

As a simplification in our setup, the aircraft and component requirements can be skipped in the traceability chain. This is the case when system requirements directly refine customer requirements or software requirements refine system requirements. Furthermore, requirements up until system level can refine other requirements from the same level to provide more detail. Software requirements can also directly implement system requirements.

The direction of the arrows indicates with which work item the link information is stored and to which other requirement the link is pointing to. Nevertheless, in the user interface and when accessed via the application programming interface (API), the link is still bidirectional (e.g., forward trace: refines / backward trace: is refined by). The links represent the trace data as required in paragraphs a. to d. in section MB.11.21 of DO-331 [Ra11b]. Paragraph e., linking between test cases and test procedures, is not included, since we do not distinguish

[13] https://extensions.polarion.com/extensions/364-system-of-systems-template

between test cases and test procedures. Linking of test cases and test results in paragraph f. is established when uploading the test results as described in section 4.

In addition to the published workflow [Dm20] for software development, the extended setup includes hardware requirements. We also introduced the new work item type *Certification Regulation*, which is used to track requirements from certification authorities like CS-23 [Eu20] from the European Union Aviation Safety Agency (EASA) or development standards like SAE AS94900A [So18] in *Polarion*. A similar approach has been published [WZ20], however, our workflow presented in section 3.1 provides more implementation details.

The four surrogate work item types in Fig. 1 (i.e., specification, assessment and design model, as well as test case) are *Polarion* representations of corresponding elements in the *MathWorks* toolchain. The complete context of the surrogate work items will be described in section 4. In addition to the surrogate test cases, the setup now also includes test cases for functional tests which are not model-based and verification items which contain the description of non-functional tests. Both the surrogate and functional test cases are used for validation of the requirements and verification of the implementation against the requirements, while verification items are, as the name implies, only used for verification.

With reference document work items, reference management is setup in *Polarion*. They can represent any type of document (e.g., aircraft manuals, data sheets, emails, meeting minutes, scientific papers, reports, etc.) and be linked with the requirements. That way, it can be traced where a specific requirement or an aspect of it originates from. For lower level requirements, reference documents can also serve as implementation trace. A common example for this are data sheets of commercial off-the-shelf (COTS) products.

We also implemented questions as work item type since we found it useful to be able to trace statements from the customer or project partners to requirements within the same platform. More details about this can be found in section 3.3. A central list of flight conditions (e.g., altitude, speed) and aircraft configurations (e.g., flap and gear setting) makes it consistent and easier for the user to specify which requirements apply to which of these conditions or configurations.

Change request work items are used to reopen already accepted work items for modification. Only after the change request has been accepted, the e.g., requirement can be reopened to the status *Draft* and modified again. This prevents quick fixes by users without considering extensive impact on other work items. The change request will also appear in the link list of the modified work item which makes it easier to track changes afterwards.

As a modification to the published review checklists [Dm20], we now use the more general task work items. In the case of requirement specification documents, a review section is appended to the document template containing multiple task work items, one for each task of the requirement review. This way, the relevant information is directly stored with the document and the content of the review does not have to be implemented as one work item type but can be assembled by multiple task work items. This provides more flexibility for

the setup of the requirement review and the status of the review tasks can be easily tracked individually.

2.2 Work Item Attributes

Besides the work item types and the linking between them, the second important part of the data model are the attributes of each work item type. By default, every work item on *Polarion* already has a set of fields. This includes the unique work item ID, a title, the status (e.g. draft, in review, accepted, rejected), an assignee field, the name of the original author, a description field (e.g., the requirement text), a severity field (i.e., must, shall or should), a comment section, hyperlinks, work item links and many more. However, we only use the explicitly named fields above (severity only for requirement work items) and additional custom fields that will be described later in this subsection.

The status field shows the current status within the workflow of the work item, and via a drop-down menu, the user can choose from the permissible transitions to other statuses. The workflow can be freely modified for every work item type in the administration panel. However, from our experience, it is advisable to reduce the complexity and strictness of the workflow to a minimum. Otherwise, especially users new to requirements management processes may get confused and not use the workflow at all. To encourage users to use the status field, we have implemented very basic work item workflows. Fig. 2 shows an exemplary requirement work item workflow. It is limited to the very basic states *Draft*, *In Review*, *Accepted* and *Rejected*. However, the workflow can be extended according to the needs of the project, even for an ongoing project. Instead of introducing a new status for requirements that are released to the implementation engineers or external parties, we recommend to use the baseline feature of *Polarion*. It enables the user to highlight specific versions of the document and provide links to these. We also discourage using the status field to track the validation and verification status of requirements. Otherwise, the requirement revision would change every time the verification status changes. Rather, automatic report pages should be used that determine the validation and verification status based on the linked requirements, test cases and the latest test results.

The built-in description field is a rich text field with basic HTML formatting where the user can also add equations, tables and figures. For the requirement description, we recommend to use formal templates like *FRET* [Gi20] or basic wording templates [ERZ14, Ch. 3].

Furthermore, from the list of built-in fields, we also use linked work items and hyperlinks. The first one is used to link *Polarion*-internal items according to the schema in Fig. 1. The second one is used to store links to external elements such as online sources for reference document work items.

In addition to the aforementioned fields, the administrator of the Polarion project is able to create custom fields with different data types like string, rich text HTML formatting,

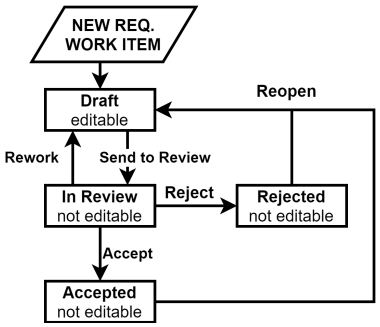

Fig. 2: Requirement Work Item Workflow with Optional Extension

Boolean, enumeration, integer, date and time. For the scope of this paper, we present the implemented custom fields only of some of the work items in the following.

For requirement work items, we added an enumeration for the type and sub-type of requirements. The enumerations and their mapping are based on ARP-4754A [So10]. A Boolean field *Derived* indicates whether the requirement is derived, meaning it is an "additional requirement[] resulting from design or implementation decisions during the development process which [...] [is] not directly traceable to higher-level requirements" [So10, p. 11]. Another Boolean field is used to identify requirements where no downstream trace (i.e., to a lower level requirement or model) is necessary. Refer to section 5 for details. A text field is used to log the justification why no downstream trace is necessary. A rationale field holds additional information about the requirement, e.g., a justification. An enumeration consisting of the means of compliance (MoC) codes from [Eu12] indicates the verification strategy for the requirements. More MoC details are provided in a rich text description field.

Besides the basic set of fields, certification regulation work items also have the already mentioned fields MoC code and description. Additionally, they have an enumeration called applicability with the values yes, no or discussion. A field for a translation of the description is provided in case the original text is not available in English, which is the working language at our institute. The output of the internal review regarding the applicability of the certification requirement or standard is logged in another text field.

The different types of surrogate work items only have the basic set of fields. *SimPol* stores a screenshot of the linked model in the description field if specified by the user. The tool also adds a hyperlink to the linked element in the *MathWorks* toolchain. More details are provided in section 4.

The description field of the question work items contains the question or proposal to the customer or project partner. Furthermore, a field for the rationale is used to provide reasoning for the question or potential implications. In case of a yes/no question or a proposal, an

enumeration field tracks the confirmation or rejection by the customer or project partner. A detailed answer can be provided in a separate rich text field.

In our implementation, documents only have the status field. The version of the document is identified via the *Polarion* repository revision number which is maintained automatically.

3 Work Item and Document Workflows

Since describing all workflows would be beyond the scope of this paper, only the following key workflows are presented. They are a combination of work item and document workflows and illustrate the steps the users have to perform.

3.1 Certification Regulation

The results of a development project not only need to fulfill customer requirements but also regulatory requirements and comply with applicable standards. Examples like EASA CS-23 [Eu20] or SAE AS94900A [So18] have already been named in section 2.1. To ensure that all relevant requirements of these standards are addressed, we implemented the following workflow.

In a first step, the original document is parsed into a spreadsheet where each row represents either a headline or a paragraph of the document. This sheet can be imported as a *Polarion* document. During the import, the headline rows will be converted to document headlines and the paragraph rows will be converted to certification regulation work items.

The resulting document is then used in a succeeding review process to evaluate the applicability of each paragraph. In case multiple systems or functions are to be developed, it is advisable to add an enumeration to the work item configuration with the different systems or functions. Once the paragraph is applicable, requirement engineers can also log to which part of the project it is applicable. As stated in section 2.2, further details about the review can be stored in a specific text field of each work item.

Once the document has been completely processed by the users, it can easily be identified which parts of the certification document or standard are relevant for the project. The created requirements based on these paragraphs are then linked to enable thorough traceability and automatic coverage reports.

3.2 Requirements

Instead of creating new requirement documents, e.g., a system requirement specification from blank, we specified document templates for each document type which are then used in

the projects. The templates are already preconfigured to only hold the appropriate work item types and the visibility settings of the work item fields for both the document itself and the editing sidebar. Work item fields visible in the document will also be included in a PDF or Word export. Fields only visible in the sidebar are only available in *Polarion*. Examples for fields which are important for requirements engineering and system development but we do not necessarily consider relevant for the exported version of the document to the customer or project partner are the requirement type, rationale and information regarding individual traces. Having this information in the document would make the specification harder to read and may distract the reader from the more important information. The document templates are already filled with a table of contents and numerous headings providing guidance for which aspects requirements have to be created. The last section in the requirement document includes the review tasks as explained in section 2.1.

Once the document is created, the users will simultaneously work on the document and add the requirements. The comments functionality of the document or the work items themselves can be used for internal discussions. This approach has the benefit that information does not get lost in endless email conversations or various messenger apps. The basic workflow of Fig. 2 also applies to the documents, except that the document itself and the task work items can still be edited when the document has the status *In Review*. However, the requirement work items will inherit the document status and can thus not be edited. If the review process determines that requirements need to be modified, they need to transition back to the status *Draft* in order to be editable again. This process is iterated until the requirement specification document is accepted. During the review, the review tasks from the end of the document need to be performed. For the custom workflow of [Dm20], the tasks only consist of checking the trace reports, which are described in section 5. Bidirectional traceability is required by ARP-4754A (section 5.4.6 a.) and DO-178C/DO-331 (objectives 1, 4, 6 of table MB.A-2) and creating these traces in later stages of development can result in higher cost of the project. We found that requiring all requirements having proper upstream and downstream traces aids in achieving a better quality of requirements, since it necessitates a thorough analysis and consideration of the requirements. In a subsequent step, once the test cases for the requirements are created, the verification coverage review is necessary to ensure that all requirements are associated with at least one test case.

Once the requirements have the status *Accepted*, they are considered final and ready to serve as a basis for further refinement or ultimately for implementation. To prevent both intentional and unintentional changes that remain undetected for other project members, already accepted requirements cannot be directly reopened to implement changes. First, a work item of the type *Change Request* has to be created and linked to the requirement providing reasoning and a proposal for the necessary change. It can be specified that only a certain user type can accept change requests, e.g., project responsibles. Only after this change request has been accepted, the requirement can be reopened into the drafting stage. Once the updated requirement was reviewed and accepted, the linked change request is closed automatically.

Following the specified workflow, objectives 1, 3, 4 of table MB.A-2 of DO-331 [Ra11b] are completely fulfilled. Objectives 2 and 5 are only partially fulfilled, since a safety assessment process is not part of the workflow. As we only included a review of the traces to reduce manual effort in our workflow, only objective 6 of tables MB.A-3 and MB.A-4 of DO-331 [Ra11b] is fulfilled. However, to improve the compliance with tables MB.A-3 and MB.A-4, further review tasks can be included in the review section of the requirement document at any time (see section 2.1). The other tables are not relevant for requirements management and are thus out of the scope of this paper. In addition, our project template also supports requirements capture and validation as described in section 5.3 and 5.4 of ARP-4754 [So10].

For more complex projects, *Polarion* also supports branching of documents. However, since we did not have a use-case for branching, we did not consider using this function. The more powerful feature of variant management was also not considered since it requires a separate commercial license.

3.3 Questions

Top-level project requirements are defined in collaboration with the customer or partner and collected in a customer requirements document, which ensures a common understanding among all parties. For example, customer requirements may define use-cases or mission requirements, desired operational features of the aircraft, or serve as a basis for the derivation of applicable standards and certification requirements [Na20]. Customer requirements provide the basis for the further development of the system and therefore are crucial for staying within the project scope, schedule and budget plan [Ri17]. From our experience, customer requirements are often ambiguous and thereby may cause additional iterations in the development process, which increases development time and costs.

Therefore, in order to obtain unambiguous customer requirements, we implemented the aforementioned question work items, which can serve as the basis for the communication with the customer and for iterating the customer requirements. For example, if a requirement seems ambiguous to us or contradicts other requirements, we create new question work items and reference them to the requirements to serve as rationale. Optionally, we describe the implications of the answer in the rationale field of the question. In general, the question work items are collected within a question list and reviewed internally before they are exported from *Polarion* (e.g., as *Word Round-trip* file) and sent to the customer or project partner. Using the *Word Round-trip* feature of *Polarion*, external parties can only modify specified fields of the work items (i.e., the ones to provide the answer), and the modified question work items can be directly imported back to *Polarion*. This can be iterated until all parties have come to an agreement. Referencing the question work items allows tracing the answer to the affected requirements and tracking of agreements directly within the requirements management platform. Because of the *Word Round-trip* feature, it is not necessary to provide the external party access to the own requirements management platform. Although the focus

is on improving customer requirements, question work items can be linked to all types of requirements and may also be utilized for internal questions or discussions.

4 Requirements to Model and Test Case Linking

Combining requirements management in *Polarion* with *Simulink* models and test cases in the *Simulink Test Manager* has already been introduced in the context of model-based requirement validation [MSH19]. Early steps in requirement and model linking habe been presented in [Sc19]. In addition to that, linking between requirements, design models and model test cases was already shown as part of our custom workflow [Dm20]. In the current paper, we present the combined and refined approach to link textual requirements in *Polarion* and modeling artifacts in *MATLAB/Simulink*.

The overview of the linking workflow is shown in Fig. 3. The left hand-side represents the *Polarion* domain while the right hand-side represents the *MATLAB/Simulink* domain. As stated in DO-331 [Ra11b], the specification models provide an abstract unambiguous representation of the textual requirement to support the understanding of the functionality. We used them for all higher-level requirements up until customer level. The output of this model is the desired behavior or response according to the requirement. In contrast to the specification models, the assessment models check whether the output of a specification or design model satisfies the requirements. In model-based design, design models represent the low-level requirement from which source code can be directly generated. The combination of specification and assessment models will result in validation test cases. After the implementation, the same assessment models can be combined with the design models for the verification test cases.

When the user links the model artifact (right side of Fig. 3) and the textual requirement (left side of Fig. 3) via the *SimPol* user interface, *SimPol* creates the model and test case surrogate work items (center of Fig. 3) in *Polarion*, which have been introduced in Fig. 1. The different types of models and test case surrogates, differentiated through link roles, represent the models and test cases from *MATLAB/Simulink*. This intermediate item is necessary to make sure that the revision of the *Polarion* requirement work item is not updated when creating or updating the link. With the auto-suspect mode active, *SimPol* can suspect and thus highlight links between requirements and models after changes have been made on the requirement side. This allows the user to identify the impact of envisaged changes to other work items, and it enables an impact analysis after changes have been made. Suspecting those links between requirements and models directly on *Polarion* is only possible with the surrogate work item setup. Another option would be to have *SimPol* create direct links between requirements and the models. However, in that case, *SimPol* would have to determine outdated links based on the revisions of the work items and models.

The *Design Models* have the link role *implements* to software requirements and to those component and system requirements that are allocated to software. Specification and

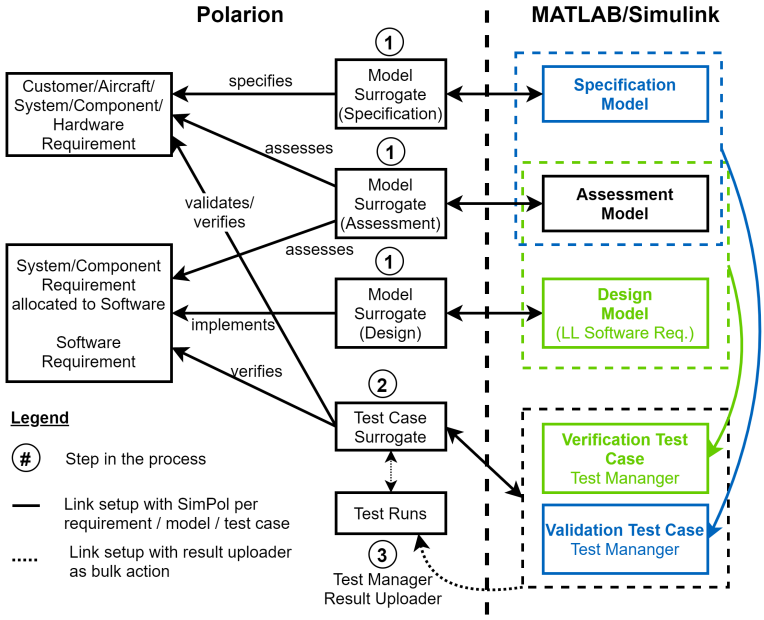

Fig. 3: Linking of Requirements, Models and Test Cases

assessment models use the link roles *specifies* and *assesses*, respectively. The model links are established in the first step of the process shown in Fig. 3. In the second step, the model-based validation and verification test cases, which are linked to the different model types along with the rest of the testing framework in the *Simulink Test Manager*, are linked to requirements in *Polarion* with the roles *validates* or *verifies*.

The third step is to run the tests and upload the records to *Polarion*. Therefore, a custom routine processes the records from the test run of the *Simulink Test Manager* containing a single or multiple test cases and saves them into an *XML* file in the *xUnit* format. A custom extension on the *Polarion* server processes this *XML* file, creates the test run records and links them to the test case surrogates. The last step is not mandatory, because the verification results are already available in *MATLAB/Simulink*, but we found it useful to also have a complete overview of the test results in *Polarion*. The process is already proven functional, however, obtaining the *XML* file currently requires manual effort. Full automation of this process is under development. The overview on *Polarion* is useful because test results will not only be generated in the *MathWorks* toolchain but also from the *Polarion* test case and verification item work items. Furthermore, the overall validation and verification status of the requirements can only be displayed on *Polarion*.

5 Requirement Traceability Report

As mentioned in section 3.2, we identified traceability between artifacts to be a key factor to be considered from early development. To provide an overview of the trace status and to identify traceability gaps, a so-called *LiveReport* page was created in *Polarion*. Based on the widgets of the page that contain the reporting code, these *LiveReport* pages retrieve data from the projects and display them in a formatted way. Since the already available widgets for traceability reports did not provide the capability and flexibility we needed, we created a custom widget. This custom widget is loosely based on the general traceability table widget and the test case coverage traceability table widget available in *Polarion*. The first widget supports checking traces in both directions (e.g., from higher to lower levels and vice versa), but since there are no colored icons indicating the trace status, it is hard to identify trace gaps. The test case coverage traceability table widget has this kind of icons included but is not capable of showing upstream traces. Both widgets also do not account for derived requirements, which per definition do not have an upstream trace and should thus not be flagged as not having a trace in the report. After almost completely rewriting the Velocity[14] code of the two aforementioned examples, the custom widget can be configured for trace reports between the following exemplary artifact types:

- All requirement types
- Requirements and specification / assessment / design surrogate models
- Requirements to source code
- Requirements and test cases

For traces to models and source code, the tool can only provide a complete report from the requirements to the models and source code. The report can show the trace status for all surrogate models but not for those model elements where no surrogate work item has been created in *Polarion*. However, this is covered by a *Simulink Model Advisor* check[15] in our custom software development process. The trace report from the source code to the requirements is not possible because an automatic analysis of the source code regarding which lines need to be linked would be required. However, this can of course be implemented as an extension in the future.

Tab. 1 shows the different types of statuses our custom widget can distinguish. A software requirement regarding the maximum allowable size of the compiled code is one example where it is justified not to have a trace to a lower level requirement or code. The trace report creates a pie chart, as shown in Fig. 4, with the distribution of the trace status types listed in Tab. 1. This serves as an overview to quickly perceive the traceability status. The detailed trace information is provided in multiple tables. Since we follow the document-centric

[14] https://velocity.apache.org/
[15] https://www.mathworks.com/help/slcheck/ref/hism-checks_hism_checks.html#hisl_0070

Tab. 1: Requirement Trace Status Types

✔	The work item has a trace of the specified link role.
✔	The work item has a trace, but the link is suspected.
✔	A justification is provided why the work item does not have a trace.
✔	The work item is marked as derived and thus does not require an upstream trace.
✖	The work item does not have a trace matching the report configuration.
!	The work has a trace and concurrently is marked as derived or justified as not having a trace.
⊘	The work item has a link to another work item which cannot be resolved (i.e., was deleted).

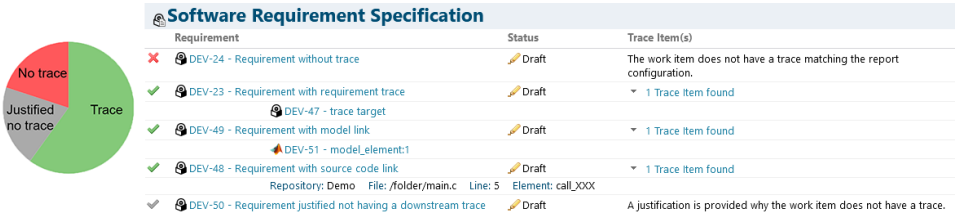

Fig. 4: Example of Pie-Chart Overview (With Enlarged Trace Labels) and Trace Report Table

approach for requirements management, we also decided to structure the report tables by documents instead of one large table. This makes it easier to understand the trace status per document and thus per aspect of the project.

A filter and customization panel allows even further report modifications from the front-end. First of all, the trace direction can be specified, meaning either from higher to lower level (downstream) or from lower to higher level (upstream). Consequently, it is not only possible to identify requirements with missing child elements but also requirements with missing parent elements. In that context, test cases and surrogate models are considered to be on a lower level than requirements. Furthermore, the report and thus the pie chart can be filtered to only show specific documents. Similarly, the report can be filtered by specific work item types (e.g., system requirements). It is also possible to modify the report to only show work items with a specific trace status, for example, create a list of all work items still missing traces. Lastly, if the aforementioned options are not sufficient, a Lucene[16] query can be provided for further filtering. This is the same query language that is also used for the work item tracker of *Polarion*.

An example of a pie-chart serving as an overview and a trace table is shown in Fig. 4. Each line of the trace report consists of a work item identified by ID and title, the corresponding trace icon from Tab. 1, and either the ID and title of the linked work item (i.e., the trace) or a description which of the other trace statuses is true.

[16] https://lucene.apache.org/core/

6 Conclusion

In this paper, we presented a requirements management template in *Polarion* and the linking strategy to model artifacts for airborne systems in the context of industry standards. The data model of the template not only consists of different types of requirements, but also contains associated artifacts that enable the developer to trace the requirements to certification references, customer or project partner decisions, flight condition and aircraft configuration indexes, test cases, and (review) tasks. We described the fields of the work item types and their purpose. Exemplarily, we also presented the workflows to manage certification references, requirements and questions. The different model types and the linking strategy to bridge the gap between the two platforms, i.e. *Polarion* and *MathWorks*, were explained. Lastly, we presented the trace report between the different requirement types, the design models and the source code. This report helps to automate the review process.

The presented setup has already proven its applicability and benefits in multiple research projects at our institute. The template provides the aforementioned features to the project without requiring project-specific modifications. Furthermore, a user guide was written to provide the users with step-by-step instructions.

7 Future Work

As already mentioned in [Dm20], we are planning to integrate *FRET* [Gi20] into Polarion as an extension, either directly as a frame in the user interface or as a pop-up window where the user can construct the requirement text. This text is then stored in the description field of the *Polarion* work item. This approach would guarantee that the requirement text is strictly following the wording template.

We are also planning to automate the upload of test results from the model domain to *Polarion*. Besides verification coverage reports (i.e., is there at least one test case per requirement) we will then also be able to provide validation and verification status reports from these test results (i.e., can a requirement be considered validated or the implementation verified because the associated validation or verification test passed). Ultimately, this automatic test result upload shall be integrated in a continuous integration (CI) process.

The aim of a current student thesis is to develop a connector to *GitLab* (instead of cloning the git repository on the *Polarion* server) for source code traceability, and an *Eclipse* user interface to manage the links from the source code to the *Polarion* work items similar to *SimPol*.

Since we are planning to extend our custom development workflow to full DO-178C development assurance level (DAL) A compliance [Dm20], this template will likewise be extended for full support. One task of this effort will be to apply the tool qualification kit to the trace reports.

References

[Dm20] Dmitriev, K.; Zafar, S. A.; Schmiechen, K.; Lai, Y.; Saleab, M.; Nagarajan, P.; Dollinger, D.; Hochstrasser, M.; Holzapfel, F.; Myschik, S.: A Lean and Highly-automated Model-Based Software Development Process Based on DO-178C/DO-331. In: 39th Digital Avionics System Conference. San Antonio, TX, USA, 2020.

[ERZ14] Eigner, M.; Roubanov, D.; Zafirov, R., eds.: Modellbasierte virtuelle Produktentwicklung. Springer Verlag, Berlin, Heidelberg, 2014, ISBN: 978-3-662-43816-9.

[Eu12] European Aviation Safety Agency: AMC and GM to Part 21 Acceptable Means of Compliance and Guidance Material, 2012, visited on: 06/28/2017.

[Eu20] European Union Aviation Safety Agency: Certification Specifications for Normal-Category Aeroplanes (CS-23), 2020.

[FM15] Fernandes, J. M.; Machado, R. J.: Requirements in engineering projects. Springer, Cham, 2015, ISBN: 9783319185972.

[Gi20] Giannakopoulou, D.; Pressburger, T.; Mavridou, A.; Rhein, J.; Schumann, J.; Shi, N.: Formal requirements elicitation with FRET. In: CEUR Workshop Proceedings. Vol. 2584, 2020.

[HMH18] Hochstrasser, M.; Myschik, S.; Holzapfel, F.: A modular model-based DO-178C software life cycle - Planning, realization, and preservation. In: DGLR Workshop. Munich, Germany, 2018.

[Ho04] Hoffmann, M.; Kuhn, N.; Weber, M.; Bittner, M.: Requirements for requirements management tools. In: Proceedings. 12th IEEE International Requirements Engineering Conference. Kyoto, Japan, pp. 301–308, 2004, visited on: 10/07/2020.

[In18a] Institute of Electrical and Electronics Engineers: ISO/IEC/IEEE 29148:2018(E): Systems and software engineering – Life cycle processes –Requirements engineering, 2018.

[In18b] International Organization for Standardization: ISO 26262-8:2018: Road vehicles — Functional safety — Part 8: Supporting processes, Geneva, Switzerland, 2018.

[KK19] Kildishev, D.; Khoroshilov, A.: Developing Requirements Management Tool for Safety-Critical Systems. In: 2019 Actual Problems of Systems and Software Engineering (APSSE). Pp. 50–57, 2019.

[LA19] Linnosmaa, J.; Alanen, J.: Demonstration of a conformity assessment data model. In: 2019 IEEE 17th International Conference on Industrial Informatics (INDIN). Helsinki, Finland, pp. 369–373, 2019.

[LH16] Lineberger, R. S.; Hussain, A.: Program management in aerospace and defense: Still late and over budget, Oct. 2016.

[LM09] Lempia, D. L.; Miller, S. P.: REQUIREMENTS ENGINEERING MANAGEMENT HANDBOOK, Springfield, VA, USA, June 2009, visited on: 02/18/2020.

[LT08] Langer, B.; Tautschnig, M.: Navigating the requirements jungle. In: International symposium on leveraging applications of formal methods, verification and validation. Springer Berlin, Heidelberg, pp. 354–368, 2008.

[MSH19] Meidinger, V.; Schmiechen, K.; Holzapfel, F.: 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, Darmstadt, Germany, 2019.

[Na20] Nagarajan, P.; Maly, M.; Jaisle, J.; Gesting, P.; Steffensen, R.; Wechner, M.; Gierszewski, D.; Krammer, C.; Holzapfel, F.: Taking Autonomy Out-of-the-Loop – Proposal of a Novel Methodology for the Development and Automated Operation of UAS in Integrated Airspace. In: 2020 IEEE/AIAA 39th Digital Avionics Systems Conference (DASC). San Antonio, TX, USA, 2020.

[Po10] Pohl, K.: Requirements engineering: Fundamentals, principles, and techniques. Springer, Heidelberg, 2010.

[Ra11a] Radio Technical Commission for Aeronautics: DO-178C: Software Considerations in Airborne Systems and Equipment Certification, Washington DC, USA, 2011.

[Ra11b] Radio Technical Commission for Aeronautics: DO-331: Model-Based Development and Verification Supplement to DO-178C and DO-278A, Washington DC, USA, 2011.

[Ri17] Rierson, L.: Developing safety-critical software: a practical guide for aviation software and DO-178C compliance. CRC Press, 2017.

[Sc19] Schmiechen, K.; Hochstrasser, M.; Rhein, J.; Schropp, C.; Holzapfel, F.: Traceable and Model-Based Requirements Derivation, Simulation, and Validation Using MATLAB Simulink and Polarion Requirements. In: AIAA Scitech 2019 Forum. San Diego, CA, USA, 2019.

[So10] Society of Automotive Engineers Aerospace: Guidelines for development of civil aircraft and systems: SAE ARP 4754 rev. A, 2010.

[So18] Society of Automotive Engineers Aerospace: AS94900A: Vehicle Management Systems - Flight Control Function, Design, Installation and Test of Piloted Military Aircraft, General Specification For, 2018.

[WZ20] Wang, Y.; Zhang, X.: Requirements Management Applied in Airworthiness Certification in the Civil Aircraft. IOP Conference Series: Materials Science and Engineering 751/, 2020.