A Lean Systems Engineering Approach for the Development of Safety-critical Avionic Systems

Ralf Bogusch, Sabine Ehrich, Roland Scherer, Tobias Sorg, Robert Wöhler
Airbus Defence and Space, Claude-Dornier-Straße, 88039 Friedrichshafen, Germany
{ralf.bogusch, sabine.ehrich, roland.scherer, tobias.sorg, robert.woehler}@airbus.com

***Abstract* — The strong cost pressure of the market and rigorous safety regulations affect the development of avionic systems. Safety standards like SAE ARP4754A and RTCA DO-178C require high efforts for assuring compliance with applicable airworthiness requirements. Hence, industry is forced to continuously optimize their lifecycle processes and tool environments to facilitate the development of safety-critical systems. In this paper, we report on our experience of adopting lean enablers to systems engineering. The approach covers requirements quality analysis, model-based systems engineering, model-based testing, product family engineering and safety analysis. The experiences are gained from an industrial case study in the aerospace domain.**

***Keywords* — Application lifecycle management, domain ontologies, end-to-end traceability, functional analysis, lean systems engineering, model-based systems engineering, model-based testing, product family engineering, requirements patterns, requirements quality analysis, safety analysis, variants.**

1. INTRODUCTION

Safety-relevant avionic systems are becoming more and more complex. Additionally, safety standards like SAE ARP4754A and RTCA DO-178C require rigorous development and assurance activities to demonstrate compliance with applicable airworthiness requirements. However, defects introduced in system specifications such as missing or unclear requirements may cause rework in the downstream activities leading to unnecessary costs. In addition, product variants sharing similar features are often developed by different teams hindering the reuse of existing solutions. Due to the strong cost pressure in the aerospace market, industry is forced to optimise their lifecycle processes and tool environments.

In order to address these challenges, we propose a Lean Systems Engineering (LSE) approach. *Lean Thinking* is a methodology which aims to deliver value to the customer while cutting out waste inadvertently generated by the process, see **Figure 1**.

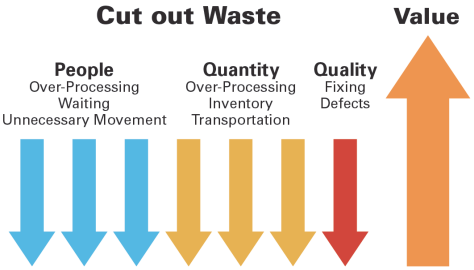

Figure 1. Lean thinking [17].

The lean methodology can be traced back to the success of the Toyota Production System (TPS) and Taiichi Ohno who is considered as the father of the TPS. J.P. Womack and D.T. Jones [18] published the basic principles in their book on Lean Thinking:

- Value,
- Value streams,
- Flow,
- Pull,
- Perfection.

In order to advance lean thinking in systems engineering, INCOSE has established the LSE Working Group. One significant result of this working group is a list of 194 practices and recommendations of systems engineering based on lean thinking, the so-called lean enablers [1]. In this paper we refer to the following lean enablers:

Map the value stream:

- Map the systems engineering value stream: have *cross-functional* stakeholders work together to build the agreed value stream.
- Plan for frontloading: anticipate and plan to resolve as many downstream issues and risks as early as possible to prevent downstream problems. Plan early for consistent robustness and "*first time right*".
- Plan to develop only what needs developing: *promote reuse* and sharing of program assets.
- Plan leading indicators and metrics to manage the program: *use metrics* structured to motivate the right behaviour.

Make value flow continuously along the value stream:

- Clarify, derive, prioritize requirements early: *use architectural methods and modelling* for system representations (prototypes, models, simulations) that allow interactions with customers.
- Promote smooth systems engineering flow: *minimize handoffs* to avoid rework.
- Make program progress visible to all: *make work progress visible* and easy to understand to all, including external customer.
- Use lean tools: use lean tools to *promote the flow of information* and minimize handoffs.

Pursue perfection:

- Strive for excellence of systems engineering processes: build in robust quality at each step of the process, and resolve and do not pass along problems. *Apply basic PDCA (Plan-Do-Check-Act) method* to problem solving.
- Develop perfect communication, coordination and collaboration policy across people and processes: *ensure timely and efficient access to centralized data*.

In the remainder of this chapter we will introduce four aims of the proposed lean systems engineering approach that allow to cover the lean enablers mentioned above.

Our first aim is to significantly *improve the quality of system specifications*, since reducing defects in requirements is the earliest opportunity in the lifecycle to save money [8]. This is accomplished by the following intertwined activities:

- Perform linguistic analysis of requirements supported by domain ontologies to obtain well-formed requirements which fulfil given quality characteristics.
- Conduct system modelling guided by model-based systems engineering (MBSE) best practice to ensure consistency and completeness of the requirements.
- Define requirements-based test cases to guarantee verifiability of requirements.
- Perform simulations of the system model and execute model-based tests to conclude the validation of the system specification.

Our second aim is to *standardize the development of variants* that share commonalities and *promote reuse*. In past projects substantial effort was spent for the development of each single variant. Thus, we introduce product family engineering practices leveraging reuse, allowing shorter time-to-market, reducing development cost, while taking into account the variability and diversity of users and customers [2].

Our third aim is to *integrate model-based system engineering with functional safety analysis* and *foster cross-functional work*. Thus, safety aspects are considered early in the system development process. Furthermore, the approach allows safety and system engineers sharing common artefacts. As a consequence, there is substantial potential for cost savings and quality improvements [14].

Our fourth aim is to *establish a tightly integrated systems engineering environment (SEE)* with interoperable tools providing requirements management, requirements quality analysis, modelling, safety analysis, simulation and test capabilities. This SEE shall support end-to-end traceability across different tools, enable efficient access to lifecycle data, support team collaboration and communication, and provide visual dashboards for process measurements. The tool integration relies on the Linked Data approach [15] and leverages interoperability standards such as Open Services for Lifecycle Collaboration (OSLC) [16].

In this paper, we present a demonstrator based on an open platform for lifecycle tool integration and an experimental case study which was performed by Airbus Defence and Space in the frame of the ARTEMIS Joint Undertaking research project CRYSTAL[1] in order to validate the approach from an industrial point of view.

The remainder of the paper is organized as follows. Section 2 introduces the industrial case study and the envisioned SEE. Sections 3 to 9 exemplify the approach. Section 10 concludes the paper and provides an outlook to future work.

2. INDUSTRIAL CASE STUDY

Airbus Defence and Space develops avionic systems that support helicopter pilots in degraded visual environments (DVE) which can be caused by e.g. rain, fog, sand and snow. Many accidents can directly be attributed to such DVE where pilots often loose spatial and environmental orientation (see **Figure 2** on the left side). In this case study we employ the landing symbology function which is part of the "Pilot Assistance Landing" capabilities of the situational awareness suite Sferion™. Other Sferion™ capabilities are "Pilot Assistance In-flight" or "Obstacle Warning".

The landing symbology function supports helicopter pilots during the landing approach. It enables the pilot to mark the intended landing position on ground using a head-tracked HMS/D (Helmet Mounted Sight and Display) and HOCAS (Hands on Collective and Stick). During the final landing approach the landing symbology function enhances the spatial awareness of flying crews by displaying 3D conformal visual cues on the HMS/D (see **Figure 2** on the right side). Additionally, obstacles residing in the landing zone can be detected.

Figure 2. Landing aid in degraded visual environments.

[1] Critical System Engineering Acceleration, http://www.crystal-artemis.eu/

The situational awareness suite Sferion™ constitutes a product family to be deployed on different helicopter platforms. The reuse of development assets, e.g. requirements, design or test cases, can be significantly improved by the application of product family engineering practices. In this case study we applied the product family engineering practice "Scoping". The practices "Domain Requirements Engineering" and "Domain Design" have been applied on system level to develop system requirements and the system design for the product family [2].

The envisaged SEE includes tools and databases to support Application Life Cycle Management (ALM) and Product Lifecycle Management (PLM), see **Figure 3**. For interoperability, each tool has to provide a connector that is based on open interoperability standards (IOS) such as OSLC[2]. The communication between the tools (e.g. sending of requests to other tools, receiving data from tools) is realised by well-established web protocols.

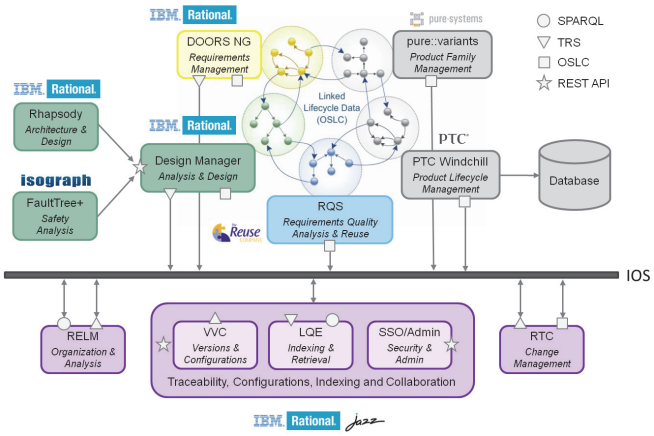

Figure 3. Envisioned seamlessly integrated SEE.

The following sections summarize the steps performed in the case study and describe the related tool chain:

- Define product family scope.
- Develop system requirements.
- Analyse and improve requirements quality.
- Develop system model.
- Perform safety analysis.
- Develop and perform model-based tests.
- Create product variants.

3. PRODUCT FAMILY SCOPING

First, the characteristics of potential products (e.g. low-cost or high-end variants) are categorized. In the Sferion™ product family three main configurations have been identified: class 2, 3 and 4.

[2] Open Services for Lifecycle Collaboration, http://open-services.net /

Then, features of the product family are identified by [7]:

- Analysing the capability in terms of end user visible services, internal operations needed to provide the end user visible services and definition of non-functional properties, e.g. performance.
- Analysing the operational environment of the product family to define the context, e.g. external systems and interfaces.
- Analysing domain technology required to implement services or operations, e.g. development approaches or tools.
- Analysing implementation technique to indicate key design and implementation decisions used to implement other features.

Features are refined in terms of commonality and variability rather than describing all details. Common features are described on top level. Feature variations are refined until no variation is exposed among products. An initial classification of each feature is performed (e.g. feature is optional; feature is refined into a range of alternatives), see **Figure 4**. Then, each feature is allocated to the intended product variant.

Finally, each feature is assessed by the stakeholders with respect to customer value, development risk and cost yielding a relevance indicator. A threshold for the relevance is defined in order to get the list of features which should be in the scope of the product family. The results of the scoping process are recorded in an Excel sheet – the Product Feature Matrix (PFM).

Feature Name	Classification	Value	Readiness	Cost	Relevance	SFERION Class 4	SFERION Class 3	SFERION Class 2
SFERION Subsystem	Dom1							
Capability Features	Dom2					✓	✓	✓
Services	Dom3					✓	✓	✓
Pilot Assistance In-Flight	Ftr					✓	✓	✓
Pilot Assistance Landing	Ftr					✓	✓	✓
Mark landing position	Ftr					✓	✓	✓
Both pilots	Alt	3	3	2	✓2,3		✓	✓
Handling pilot only	Alt	1	3	1	✓2,0	✓		
Check landing point	oFtr						✓	✓
Check No Ground	oFtr	2	3	1	✓2,3		✓	✓
Check Obstacles	oFtr	3	3	3	✓2,0			✓
Operations	Dom3					✓	✓	✓
Terrain Data Fusion	oFtr	1	3	1	✓2,0			✓
NonFunctional Properties	Dom3					✓	✓	✓
Performance	Dom4					✓	✓	✓
500km Radius Terrain Data	Ftr	3	3	2	✓2,3	✓	✓	✓
Safety	Dom4					✓	✓	✓
Frozen Image detection	oFtr					✓	✓	✓
HMD	oFtr	2	2	2	!1,7		✓	✓
HDD	Ftr	2	3	1	✓2,3	✓	✓	✓
Operational Environment	Dom2					✓	✓	✓
OWS	oFtr							✓
SferiSense	Alt	3	3	2	✓2,3			✓
ELOP	Alt	2	3	2	✓2,0			
Domain Technology	Dom2					✓	✓	✓
OpenGL 1.0.1 SC	Ftr	2	3	2	✓2,0	✓	✓	✓
Implementation Technique	Dom2					✓	✓	✓
Digital Map Projections on HDD	oFtr					✓	✓	✓
WGS84	Alt	2	3	1	✓2,3	✓	✓	✓
Orthographic	Alt	2	1	3	!1,0			

Figure 4. Extract of a Product Feature Matrix (PFM).

Important descriptive goals of the feature model are:

- Understanding of the variability among products.
- Communication of requirements in terms of features.
- Specification of product variants.

Additionally, prescriptive goals of the feature model are important for this case study:

- Guidance for the development or transformation of core assets which are in our case study the product family system requirements and product family architecture.
- Guidance for variant derivation of the system requirements and the system architecture model.

The variability modelling tool pure::variants was selected to support the prescriptive feature modelling goals. Based on the PFM, a variability model has been set up with pure::variants in order to formalize the results of the product family scoping in a feature tree, see **Figure 5**.

Subsequently, features, commonalities, variabilities and structural relationships have been added to the variability model.

The feature tree was manually optimized to represent variabilities and commonalities rather than functional dependencies like a function call hierarchy.

For example, the mandatory feature "Mark landing position" can be realized in two ways: either the handling pilot only or both pilots are allowed to set the landing position. All product variants provide the safety feature "FrozenImageDetection". However, the frozen image detection on "HMD" is optional.

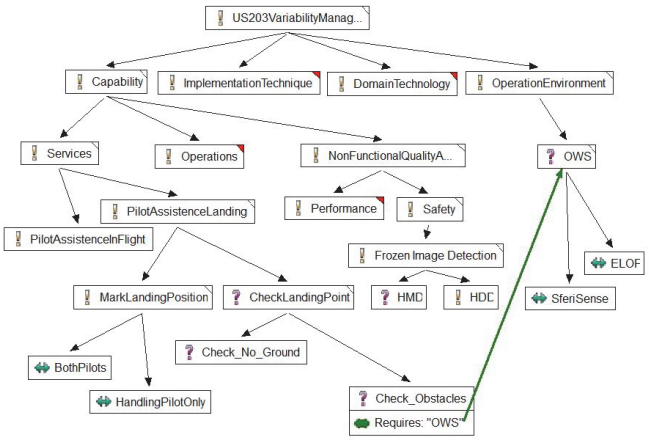

Figure 5. Feature model describing the variabilities.

The main advantage of the feature modelling approach is that based on the feature model, validation mechanisms assure that only valid product configurations are defined.

In order to provide guidance for variant derivation it is important to add composition rules to optional and alternative features. Essential composition rules are:

- Mutual dependency relationship: features that shall be selected along with the designated one.
- Mutual exclusion relationship: features that shall not be selected along with the designated one.

For example, if "Check_Obstacles" is selected, an Obstacle Warning System (OWS) is required. In this case, one of the sensor equipments "ELOP" or "SferiSense" has to be selected.

4. REQUIREMENTS DEVELOPMENT

After the definition of the product family scope, a first set of system requirements covering all features of the product family and derived from stakeholder needs is defined using the requirements management tool DOORS NG. The requirement type is assigned. This enables filtering requirements according to their type, e.g. gathering all functional requirements as input for the subsequent functional analysis.

In order to support the requirement authoring activity, we use two additional tools from the Requirements Quality Suite (RQS) [11]:

- Knowledge Manager (kM), which is used to create and maintain the domain ontology and requirements patterns.
- Requirements Authoring Tool (RAT), which provides on-the-fly guidance for the requirements specification activity [9].

First, we developed a domain ontology by defining terms, abbreviations, agreed-upon concepts and relations that hold in our application domain. Then, we defined requirements patterns by defining sequences of fixed syntax elements for each type of pattern to cover the relevant requirements statements [10]. Requirements patterns can be specified in an elementary way and then concatenated to cover more complex sentence structures, allowing keeping the number of required patterns low while at the same time having a high flexibility. The majority of the functional requirements of our case study could be covered with a few set of patterns.

Figure 6 shows how a Natural Language (NL) requirement is represented by a pattern. The pattern consists of fixed syntactic elements and variable elements. The latter are mapped to semantic concepts in the domain ontology such that the requirement can formalized by a semantic graph.

Using requirement patterns is an effective way to mitigate many types of ambiguity in NL requirements and to enable advanced formal analysis of these requirements, see Section 5. For instance, we can detect requirements formulated in passive voice or any requirement structure that is not well-formed like "It shall be possible..." which lacks an actor.

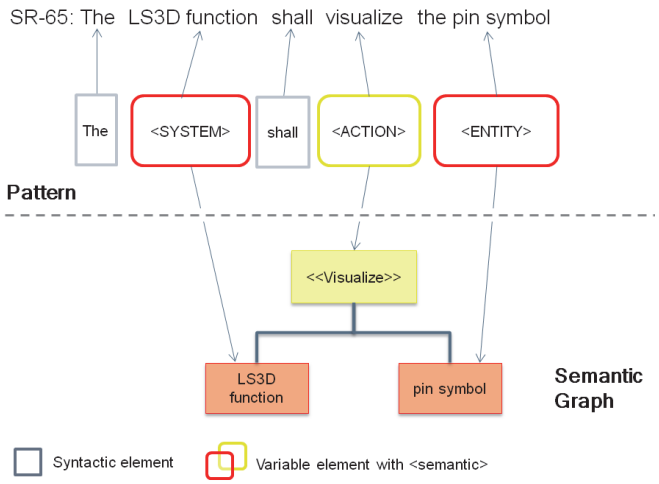

Figure 6. Semantic analysis.

The RAT tool uses the concepts of the domain ontology together with the set of pre-defined requirement patterns managed in the common asset repository to provide a list of suggestions the requirements author can directly build on when defining textual requirements. If done manually, pattern conformance checking can be cumbersome, particularly when requirements change frequently. The tool gives real-time feedback whether a selected pattern is matched or not and also on the quality. Additionally, terms of the controlled vocabulary fitting with the next matched pattern element are suggested, which force the authors to use consistently agreed-upon domain terms.

Having a proactive and interactive guidance that tries to improve requirements quality while actually writing requirements, is a real benefit. It is essential that errors in requirements are found early to avoid cost and error propagation later in the project, as well as reducing rework and modification loops. According to Boehm's law [6], revealing a defect in the requirements stage is 100 times cheaper than fixing one in coding. Additionally, using a domain ontology allows a better know-how transfer between domain experts and requirements authors in order to achieve a common understanding on the set of requirements between all project stakeholders. Dealing with all these new tools beside DOORS NG as requirements management tool bring new challenges and a systematic process for ontology creation and maintenance is needed as well as a new engineering role: the knowledge manager.

In order to integrate the set of system requirements with the variability model, formal relationships between individual system requirements and features need to be established. The OSLC connector of pure::variants reads all requirements including the allocated features from DOORS NG and initializes the family model defining the reusable artefacts accordingly. In case of changes, the family model can be synchronized again with changes performed in DOORS NG.

5. REQUIREMENTS QUALITY ANALYSIS

The quality of requirements has a major impact on project success. Badly written requirements are a well-known source of project failure. Moreover, the quality of requirements should be secured before reusing them in different product variants. For this reason, quality metrics are employed to measure the requirements quality and to identify language defects, see **Figure 7**. Some quality metrics are: readability, use of passive voice, ambiguous terms, negations, abuse of connectors, undefined acronyms, and inconsistent use of measurement units. The INCOSE Requirements Working Group [4] and ISO/IEC 29148 [5] provide an exhaustive definition and justification of requirements quality characteristics.

Quality analysis can be done on different levels as depicted in **Figure 7**:

- Textual analysis: e.g. size or readability.
- Lexical and syntactic analysis: e.g. ambiguous terms and passive voice.
- Semantic analysis based on domain ontologies: e.g. overlapping requirements and completeness.

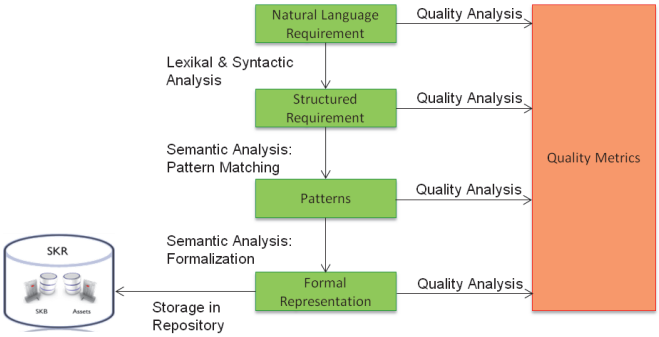

Figure 7. Derivation of requirement quality metrics.

We use the Requirements Quality Analyser (RQA) for DOORS NG that provides an automatic quality evaluation of NL requirements by performing lexical and linguistic checks as well as semantic analysis based on the developed domain ontology and requirement patterns, see Section 4.

RQA comprises more than 60 pre-defined metrics, from which a subset may be selected for quality analysis. Every single requirement is analysed one by one and a series of indicators (e.g. readability, ambiguous phrases, and traceability) is determined for every requirement. Every indicator is then transformed into a qualitative value by the associated quality function. During the evaluation, every quality metric rated as medium or low will generate some hints and suggestions for improvement. **Figure 8** depicts measurement results for one requirement. Most relevant findings are:

- Ambiguous sentences: “be capable of”
- Ambiguous use of connectors: “and/or”
- Missing traceability: outlink of type “satisfies” is missing

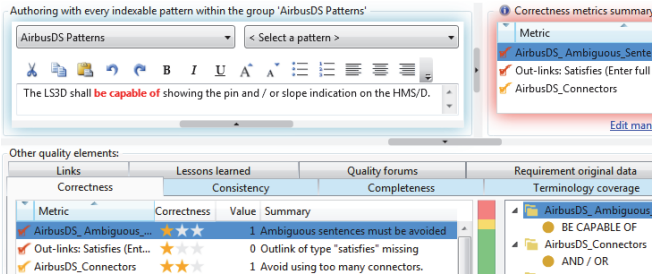

Figure 8. RQA quality report.

There is also another view for the whole set of requirements in a DOORS NG module, which provides a quick overview of the requirement quality in this module. Moreover, the most frequent quality issues are displayed which allows identifying focused corrective actions.

In addition, there exist several ontology-based metrics which give feedback whether concepts are properly used at the right level of abstraction. For instance we can make use of metrics related to the Product Breakdown Structure (PBS), which is represented within the ontology. In lower level documents, we activate the detection of "compound" terms to force the authors to specify the concrete sub-elements as defined in PBS. On the other hand for high-level documents, we can check whether "part" terms have been used to avoid that there exist too specific terms there.

Some additional metrics make use of the categorization of terms within a specific semantic cluster. Terms which represent a system, component or a sub-component of the PBS are assigned to e.g. the semantic cluster "SYSTEM". These clusters are used within the pattern definition to restrict the allowed terms within the defined patterns.

Additionally, we make use of the common asset repository for advanced requirements analysis like consistency and completeness of the whole set of requirements. When dealing with complex systems and a high number of requirements as well as many hierarchal levels, consistency and completeness checks are difficult manual tasks. Here are some examples of automated quality checks:

- Identification of inconsistent use of measurements units.
- Identification of redundant requirements by means of their respective formalization as a semantic graph. Overlapping requirements between different hierarchical levels may be an indication for an insufficient refinement.
- Completeness assessment whether any requirements of a specific category (e.g. safety, performance…) are possibly missing based on the matched pattern groups within the ontology.

Finally, analysis results are interpreted; requirements are improved and updated in DOORS NG. In addition, the domain ontology needs to be updated and validated by changing concepts and pattern formalizations to improve guidance and analysis results.

The whole set of RQS tools provides an effective support in defining well-structured requirements and performing automated analysis of requirements. With this approach, the review is more efficient since the amount of manual work is reduced; all trivial checks are performed by the tool and can even be corrected by the author itself. The reviewer can then concentrate on "difficult" points, the assessment results are reproducible and do not depend on individual reviewers' subjective opinion. Quality criteria are clear right from the beginning and hints are provided which motivate authors and drive the quality improvement process of requirements according to the PDCA (Plan-Do-Check-Act) cycle [8].

6. SYSTEM MODEL DEVELOPMENT

In order to support early validation of the system specification and improve communication within the project team and with stakeholders, we adopt a Model-based Systems Engineering approach (MBSE), which constitutes three different viewpoints:

- Functional viewpoint
- Logical viewpoint
- Physical viewpoint

The aim of the *functional viewpoint* is to describe in detail from a technical perspective the functions and behaviour the intended system shall provide, the interaction via identified interfaces with external systems, users and operators, and the interaction and dependencies between the different functions (black box approach). The functional analysis is performed with Rhapsody based on the MBSE methodology Harmony/SE [3] and results in use case diagrams, activity diagrams, sequence diagrams, block diagrams and statechart diagrams.

First, the system requirements are accessed from DOORS NG and displayed in Rhapsody. The goal is to prove the complete consideration of all relevant system requirements during the functional analysis process. Then, the system context with interfacing actors is defined with a use case diagram. Other use case diagrams may be created to group the functional scope accordingly. Subsequently, the different functional flows are captured using activity diagrams. In the next step, a set of sequence diagrams consistent with the activity diagram and describing the behaviour of the system in a particular situation, is derived. Operations are derived from actions. Events and related event receptions are defined in relation with external actors. The identified events are linked to data items that are transported across interfaces. With the identified sequence diagrams, the system ports and interfaces including all in- and outgoing events can be defined using an internal block diagram. To complete the functional analysis, a statechart is added to the system block to describe its state-based behaviour.

Thanks to the formal semantics of statecharts, code can be generated that implements the state-based behaviour of the system. Although the generation of production code is possible, we only use code generation for the execution of the functional model since we do not intend to qualify the

code generator. When executing the model, the system behaviour can visually be inspected with the help of animated statecharts. Model execution is a powerful method to check the correctness and completeness of the functional model and the related system requirements. For this purpose simulations can be setup either by manually stimulating the system with events or by implementing model-based tests. The latter is described in more detail in Section 8.

The aim of the *logical viewpoint* is to identify an architecture of logical blocks which realises the functions and behaviour identified during the functional analysis (white box approach). In this process also non-functional aspects are considered, e.g. reduction of the number of interfaces, segregation of functions with different criticalities (see Section 7), reuse of available functions (see Section 9), integration of external suppliers, and utilization of COTS elements. The system block representing the complete functional scope is decomposed into logical blocks. Activity diagrams and sequence diagrams are refined considering partitions and lifelines related to logical blocks. System internal ports and interfaces are added for connecting the logical blocks. The state-based behaviour of all logical blocks is defined which allows to validate the logical architecture using model execution.

The aim of the *physical viewpoint* is to find a suitable system architecture consisting of system elements (e.g. hardware or software items) that allows to implement the logical blocks identified in the logical viewpoint, while considering the business needs and non-functional constraints. In order to derive a candidate system architecture, the logical blocks of the logical architecture are mapped to physical elements of the physical architecture. In the same way, the logical interfaces are mapped to physical interfaces. Both physical elements and physical interfaces need to comply with associated performance requirements. Additionally, the properties of the selected physical elements (e.g. power consumption, memory size, …) are identified and modelled. Based on a set of prioritized criteria, different candidate solutions can be assessed during a trade-off study. Finally, a solution is jointly selected by all stakeholders.

Moreover, traceability is established between the system requirements in DOORS NG and the model elements in Rhapsody for all viewpoints of the system model to support coverage and impact analysis. For example, when requirements are changed, a suspect indicator is automatically added to the trace which helps maintaining consistency.

7. SAFETY ANALYSIS

Based on system requirements and the functional analysis of the system, hazards can be identified. Contributing system functions, as they have been identified through the functional analysis in the black box system model, are classified according to their related criticality level. The classification of the system functions as part of the Functional Hazard Assessment (FHA) considers failure modes (e.g. total loss, partial loss, inadvertent provision, and erroneous provision) of functions and the flight phase (e.g. takeoff, approach, and landing) where this failure conditions occur. Further results of the hazard assessment are derived requirements. These new safety requirements are then considered in the white box system model within the logical and physical architecture, where the system architecture has to show compliance not only to the initial functional requirements but also to all safety related requirements.

Figure 9 illustrates the black box approach that comprises the functional analysis and shows how the FHA process is integrated with the functional viewpoint.

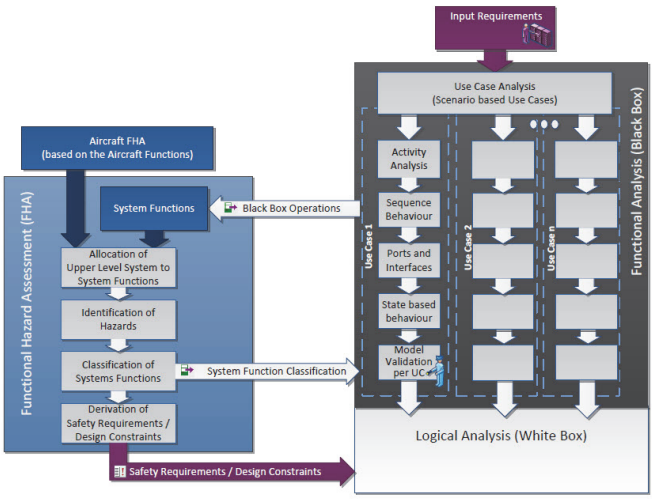

Figure 9. Black box approach.

The white box approach consists of the logical viewpoint, in which an architecture is iteratively developed preventing all failures which have been identified through the FHA. Since also the physical viewpoint, that involves the identification of real interfaces and hardware components, is emerging from the white box approach, fault trees can be elaborated with a failure condition as a starting point and refined up to physical elements, where acceptable probabilities for the occurrence of failure conditions are assigned.

Figure 10 shows the white box approach comprising the logical and the physical viewpoints.

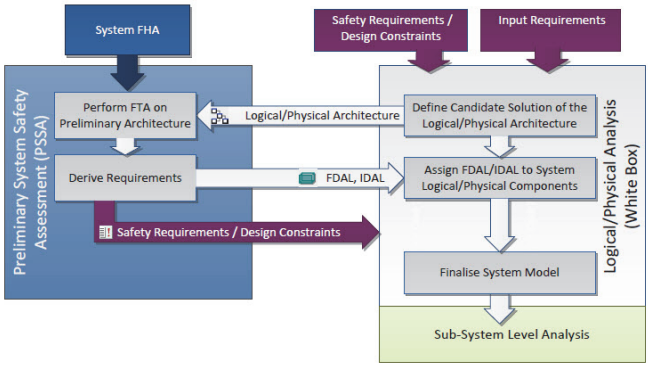

Figure 10. White box approach.

System modelling including functional, logical and physical viewpoints is performed with Rhapsody, whereas safety requirements and failure condition lists are kept in DOORS NG. Fault trees as part of the Preliminary System Safety

Assessment (PSSA) are generated with isograph FaultTree+. As shown in **Figure 11**, system functions identified from black box analysis are made available for the FHA elaborated in DOORS NG. As a result from the FHA, safety requirements and design constraints are available for the white box analysis in Rhapsody. Moreover, they could have an impact on the black box analysis since new functions may have to be elaborated to mitigate the safety concerns. Using the FHA output and the white box analysis refinement into components, a PSSA is developed including fault trees. Resulting from these fault trees FDALs (Function Development Assurance Level) and IDALs (Item Development Assurance Level) are then allocated to components. This proceeding allows establishing traceability between safety requirements, function classifications and the system model.

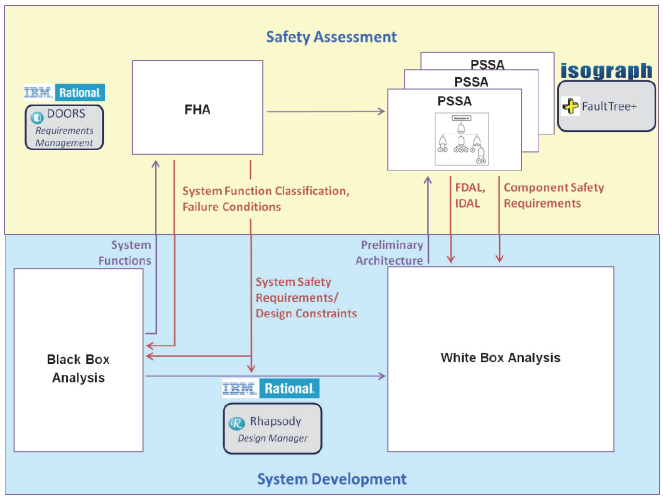

Figure 11. Tooling and exchanged artefacts.

8. MODEL-BASED TESTING

The notion of Model-based Testing (MBT) refers to the application of models for automation of testing activities as well as modelling of test artefacts. In the following MBT is used to automatically generate test artefacts from the black box system model which represents the system under test (SUT). As discussed in Section 6 this model allows the simulation of the system behaviour. The intention is to verify that this simulation is compliant with the expected behaviour as defined by the system requirements [12].

The underlying process can be divided into the following phases [13]:

1. Modelling of the SUT and/or its environment. Creation of the test architecture.
2. Generation of executable test cases from the black box system model.
3. Test execution on the SUT and assignment of verdicts.
4. Analysis of the test results.

Figure 12 provides a graphical overview of the MBT process.

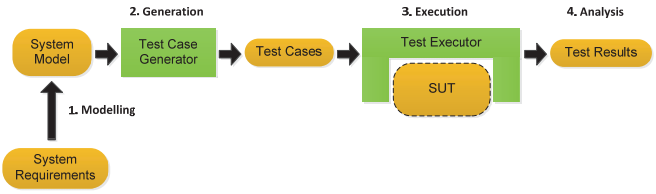

Figure 12. MBT process.

Modelling of the behaviour of the SUT is already described in detail in Section 6. TestConductor (TC), the test execution and verification engine of Rhapsody is employed for creating the test architecture of the selected SUT. Test architectures comprise all artefacts which are needed for test automation, e.g. test components are created for stubbing of external interfaces.

For automatic generation of executable test cases the Rhapsody Automatic Test Generation add-on (ATG) is applied. By analysis of the specified system model ATG automatically generates suites of test cases which are specified by UML sequence diagrams as depicted in **Figure 13**. Test stimuli are generated which allow observing the behaviour of the SUT. In this format the test cases are ready for execution.

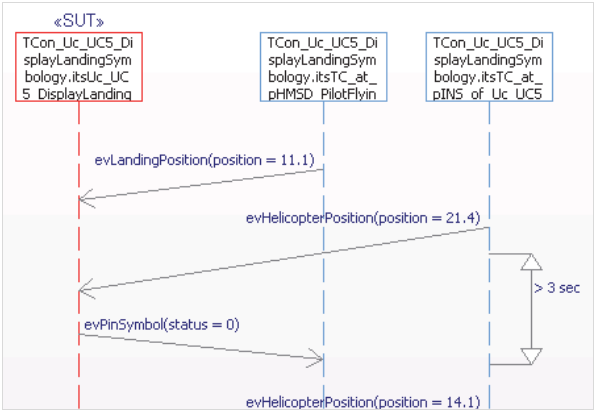

Figure 13. Test case generated by ATG.

TC is employed for executing the test cases. Test verdicts are provided which for example allow to identify all requirements which are not satisfied by the black box model or state to which extent the system model has been covered by the executed tests.

After completion of the test execution a detailed report summarizing all relevant information including test results and model coverage analysis is generated, see **Figure 14**.

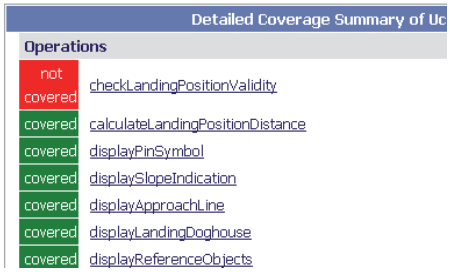

Figure 14. Model coverage report generated by TC.

9. VARIANT MANAGEMENT

The definition of product variants is supported by an auto resolver based on the feature model (see Section 3). **Figure 15** illustrates how the configuration process is performed. Mandatory features like "MarkLandingPosition" are automatically selected. Optional features without a composition rule like "TerrainDataFusion" need to be selected manually. Features affected by composition rules are selected or deselected by the auto resolver, e.g. if "Check_Obstacles" is user-selected, the feature "OWS" is automatically selected by the auto resolver.

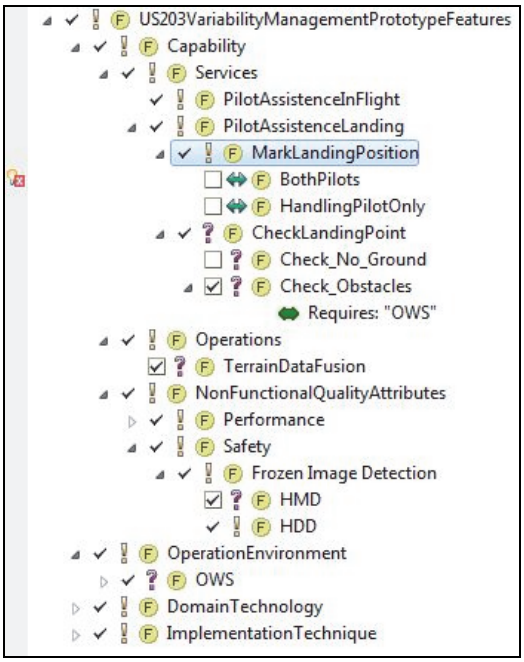

Figure 15. Configuration of a product variant.

Inconsistencies in the feature model or missing feature decisions, e.g. in case of alternative feature selections, are highlighted in order to conclude a valid variant description, see missing decision on the alternatives of feature "MarkLandingPosition" in **Figure 15**.

Product variants are compared and validated using the product variant matrix view, see **Figure 16**.

Model Elements	Level	Class2	Class3	Class4
US203VariabilityManagementPrototypeFeatures				
US203VariabilityManagementPrototype...		✓	✓	✓
Capability	1	✓	✓	✓
Services	1.1	✓	✓	✓
PilotAssistenceInFlight	1.1.1	✓	✓	✓
PilotAssistenceLanding	1.1.2	✓	✓	✓
MarkLandingPosition	1.1.2.1	✓	✓	✓
BothPilots	1.1.2.1.1	☑	☑	☒
HandlingPilotOnly	1.1.2.1.2	☒	☒	☑
CheckLandingPoint	1.1.2.2	✓	✓	☐
Check_No_Ground	1.1.2.2.1	☑	☑	☐
Check_Obstacles	1.1.2.2.2	☑	☐	☐
Operations	1.2	✓	✓	✓
TerrainDataFusion	1.2.1	☑	☐	☐
NonFunctionalQualityAttributes	1.3	✓	✓	✓

Figure 16. Product matrix view.

Each variant configuration is input to the transformation process that creates system requirements, system models, and tests for product variants. Finally, system specifications can be generated automatically capturing the information gathered during the requirements and system model development in a structured way. The generated documents can be used to perform formal reviews, fulfil contractual obligations or show regulatory compliance.

10. CONCLUSION

We reported on our industrial case study in the aerospace domain which was conducted in the frame of the ARTEMIS Joint Undertaking project CRYSTAL. The case study and the demonstrator cover the identification of variabilities, the construction of a feature model using pure::variants, the development and quality analysis of system requirements using DOORS NG and RQS, the elaboration of a system model using Rhapsody and safety analysis tightly integrated within the MBSE process. Moreover, requirements-based test cases are developed with TestConductor and ATG that allow verifying the system model by simulation runs.

Applying our approach to the industrial case study has produced promising results and addresses the following lean enablers (see Section 1): the *basic PDCA method is applied* during the requirements quality analysis to resolve problems related with badly written requirements. The *use of architectural methods and models* in the model-based systems engineering approach allows to clarify requirements early. Through the combination of linguistic analysis, model-based systems engineering, and model-based testing the quality of system specifications can significantly be improved. This contributes to the "*first time right*" objective. The tightly integrated safety analysis supports *cross-functional* team work between systems and safety engineers, *promotes the flow of information* and *minimizes handoffs* and rework. Additionally, the product family engineering approach allows to derive system specifications for different variants and *promotes reuse* of shared artefacts. The integrated SEE supports *efficient access to systems engineering data*. It enables to *make the work process visible* using metrics and facilitates end-to-end traceability analysis across different tools.

The experience gained from the realisation of the case study can be summarized as follows:

- Setting up useful ontologies that effectively support authoring and quality analysis of requirements still requires a lot of experience and specific skills. Therefore, the depth and thoroughness of quality analysis has to be carefully planned depending on the objectives and boundary conditions of each project.
- In the case study a monolithic feature model was chosen. In more complex systems a modular feature model structure might be considered. A modular feature model approach is easier to understand and provides a better separation of concerns but has the burden of maintaining the

traceability and dependencies between the feature models.

- The tool integration approach of the SEE allows loosely coupled tools to share and link data based on standardized and open web technologies. It should be noted that an increasing number of tool vendors support the OSLC standard. However, standardisation needs to be extended in order to support domains not covered yet, e.g. variability management and safety analysis.

We are currently working to further develop the integration of safety analysis into the MBSE approach. In addition, we are using formal specifications that can be analysed with model checking techniques. Moreover, we are adopting a more holistic approach of configuration management which provides means to define global system configurations containing all relevant types of artefacts (e.g. requirements, system model and test cases) including the traceability links between them.

Our future work will advance and improve component-based and model-based systems engineering practices. Components will comprise executable specifications and ease virtual integration and prototyping. They will provide variation points in order to support product families by derivation of variants. Contractual specification of components enable more efficient system verification and continuous checks of functional safety properties, e.g. by formal checks performed during virtual integration. Components already being qualified provide certification evidence to support incremental qualification and certification approaches. New systems can then be constructed with components according to domain-specific reference architectures. Domain-specific languages will further ease the specification, architectural design and verification of complex systems.

ACKNOWLEDGMENTS

The research leading to these results has received funding from the ARTEMIS Joint Undertaking under grant agreement no. 332830 (ARTEMIS project CRYSTAL) and German BMBF.

REFERENCES

[1] Oppenheim, B.W., *Lean for Systems Engineering with Lean Enablers for Systems Engineering*. John Wiley & Sons, 2011.

[2] Pohl, K., G. Böckle, and F. J. v. d. Linden, *Software Product Line Engineering: Foundations, Principles and Techniques.* Springer-Verlag, 2005.

[3] Hoffmann, H.-P., Deskbook Release 4.1, Model-based Systems Engineering with Rational Rhapsody and Rational Harmony for Systems Engineering. IBM Corporation, 2014.

[4] Requirements Working Group, International Council on Systems Engineering (INCOSE), "Guide for Writing Requirements", INCOSE, 2012.

[5] International Standard ISO/IEC 29148: Software and systems engineering — Life cycle processes — Requirements engineering, 2011.

[6] Boehm, B.W. and Ph. N. Papaccio, "Understanding and Controlling Software Costs", *IEEE Transactions on Software Engineering* 14 (10), October 1988, pp. 1462-1477

[7] Lee, K., K. Kang, J. Lee: Concepts and Guidelines of Feature Modeling for Product Line Software Engineering, ICSR-7, 2002.

[8] Bogusch, R., "Towards Automatic Quality Evaluation of Natural-Language Requirements", In: M. Maurer, S.-O. Schulze (eds.), *Tag des Systems Engineering, Bremen 12.-14. November 2014*, Hanser, München, 2015, pp. 401-410

[9] Fuentes, J.M., A. Fraga, J. Llorens, L. Alonso, and G. Génova, Requirements Authoring: Towards the Concept of a Standard Requirement. International Council on Systems Engineering (INCOSE) Symposium 2014, Las Vegas, 2014.

[10] Fraga, A., J. Llorens, J.M. Fuentes, and L. Alonso, Knowledge-based Requirements Engineering Process: A Guided Example. To appear in *Proc. FDL Forum on Specification & Design Languages*, München, 2014.

[11] Requirements Quality Suite, The REUSE Company, http://www.reusecompany.com/requirements-quality-suite (last visited October 13, 2015).

[12] Dias, N.A., R. Subramanyan, M. Vieira, and G. Travassos, "A survey on model-based testing approaches: A systematic review", *Proceedings of the 1st ACM international workshop on Empirical assessment of software engineering languages and technologies*, Atlanta, 2007, pp. 31-36

[13] Utting, M. and B. Legeard,. *Practical Model-Based Testing: A Tools Approach*, Elsevier Science & Technology Books, 2006.

[14] Binder, I., "Towards seamless integration of functional safety and model-based systems engineering", In: M. Maurer, S.-O. Schulze (eds.), *Tag des Systems Engineering, Bremen 12.-14. November 2014*, Hanser, München, 2015, pp. 53-62

[15] Bizer, Ch., T. Heath, and T. Berners-Lee. "Linked Data – The Story so Far". *International Journal on Semantic Web and Information Systems* 5 (3), 2009, pp. 1-22

[16] OSLC. Open Services for Lifecycle Collaboration. OASIS. http://open-services.net (last visited October 13, 2015).

[17] Lean Systems Engineering Working Group, International Council of Systems Engineering (INCOSE), Lean Systems Engineering - an Introduction, INCOSE UK, 2015.

[18] Womack, J.P. and D.T. Jones. *Lean Thinking*. Simon & Schuster, New York, 1996.