 THE RESEARCH LEADING TO THESE RESULTS HAS RECEIVED FUNDING FROM THE EUROPEAN UNION'S SEVENTH FRAMEWORK PROGRAM (FP7/2007-2013) FOR CRYSTAL – CRITICAL SYSTEM ENGINEERING ACCELERATION JOINT UNDERTAKING UNDER GRANT AGREEMENT N° 332830 AND FROM SPECIFIC NATIONAL PROGRAMS AND / OR FUNDING AUTHORITIES.

CRitical S**YST**em Engineering **A**cce**L**eration

Milestone Report v1 -
Public Use Case AUTOMOTIVE
D 307.011

DOCUMENT INFORMATION

Project	CRYSTAL
Grant Agreement No.	ARTEMIS-2012-1-332830
Deliverable Title	Public Use Case AUTOMOTIVE
Deliverable No.	D 307.011011
Dissemination Level	PU
Nature	R
Document Version	V3.0
Date	2014-01-29
Contact	Andrea Leitner
Organization	VIF
Phone	+43 316 873 9615
E-Mail	andrea.leitner@v2c2.at

AUTHORS TABLE

Name	Company	E-Mail
Andrea Leitner	VIF	andrea.leitner@v2c2.at
Alberto Melzi	CRF	alberto.melzi@crf.it
Oscar Ljungkrantz	VOLVO	oscar.ljungkrantz@volvo.com
Gerald Stieglbauer	AVL	Gerald.stieglbauer@avl.com
Jörg Settelmeier	AVL-R	joerg.settelmeier@avl.com
Daniel Hopp	DAI	daniel.hopp@daimler.com
Serrie Chapman	IFX-UK	Serrie.Chapman@infineon.com
Pierre du Pontavice	VALEO	pierre.du-pontavice@valeo.com
Cecilia Ekelin	VOLVO	cecilia.ekelin@volvo.com
Michael Maletz	AVL	michael.maletz@avl.com
Christoph Bräuchle	PTC	cbraeuchle@ptc.com

REVIEW TABLE

Version	Date	Reviewer
0.7	15.1.2014	Cecilia Ekelin (Volvo)
0.11	23.1.2014	Matthias Tichy (CTH)
0.11	25.1.2014	Christian Webel (FhG)

CHANGE HISTORY

Version	Date	Reason for Change	Pages Affected

CONTENT

Content of Figures

Content of Tables

1 Introduction

1.1 Role of deliverable

The aim of this deliverable is to describe the public use case AUTOMOTIVE. The corresponding workpackage WP307 can be seen as a common platform for discussions within the automotive domain. The partners share challenges and experiences from the different automotive use cases and discuss commonalities and possible collaborations. This means that the public use case AUTOMOTIVE is actually not one single use case with a clearly defined scope. Instead, it will be a collection of experiences, solutions, and best practices from the automotive domain.
Following prior experiences, it is not possible to come up with a common interoperability solution, which provides a generally valid solution for all kinds of interoperability challenges. Instead, different kinds of problems require different kinds of solutions. The public use case AUTOMOTIVE should reflect this fact and result in a set of best practices and state-of-the-art solutions, which are demonstrated using examples from the automotive domain. This will most likely not result in one coherent demonstrator, but a demonstrator which shows sample solutions for clearly defined interoperability challenges from the single automotive use cases.
This deliverable provides a first overview of the public use case AUTOMOTIVE, the process steps covered by the different partners (the different partner use cases), and their main interoperability challenges.
One first demonstrator focusing on traceability between requirements and model elements is currently in work and will be described in this first version of the deliverable.

Basically this deliverable can be seen as a living document which evolves throughout the project and results in three deliverable versions:

- v1 – Identified interoperability challenges:
  - The purpose of this first version is the introduction of the automotive domain, the scope of the use case, and a description of interoperability challenges identified so far. This deliverable will also describe a first design of the demonstrator.
- v2 – Presentation of first results:
  - The second version of the deliverable will present the harmonization of interoperability challenges, first results, experiences, and further possible challenges identified in the course of the project.

    This includes a detailed description of the interoperability challenge incl. tools, artifacts, and processes and a detailed investigation of additional aspects which have to be considered for practical use (e.g. link consistency, version management, change management, variability, functional safety, access restrictions, restrictions due to process, etc.)

    Besides the detailed problem description it should also cover first implementation concepts, including considerations what can be implemented using existing means (e.g. OSLC), what can be implemented by extending existing means (e.g. extending or specifying an OSLC domain, and which challenge requires a new solution.
- v3 – State-of-the-art solutions and best practice:
  - The final version of the deliverable will show a set of best practices and state-of-the-art solutions for the identified interoperability challenges.

The CRYSTAL public use case AUTOMOTIVE Work Package (WP307) has the following major purposes:

- Describe typical automotive challenges with respect to interoperability, safety, and variability in order to:
  - Support SP3 Use Cases refinement
  - Identification and discussion of typical automotive challenges which can be provided as an input for the IOS Working Group and SP6 Technical Bricks. This means that the challenges of the different automotive partners will be consolidated in order to come up with more general challenges.
- Perform a prototyping of IOS Concept
  - To refine and validate the feasibility and value of the CRYSTAL interoperability approach
  - Showing the main “idea” behind the CRYSTAL approach
- Facilitate the presentation of CRYSTAL results in publications without facing IPR concerns (support dissemination activities).
- Support knowledge and experience transfer between all automotive partners. The use of the public use case as a forum for discussion and transfer of experiences and know-how is an important aspect of this workpackage. We experienced so far that the discussion of problems is very useful to understand and probably solve these problems. Some partners have already participated in other projects, like CESAR[1], MBAT[2] and iFest[3], and can therefore share their experiences.

In the course of the project we have identified various similarities in the interoperability challenges. In the next iteration of the project, we will discuss these challenges in more detail in order to find out if we can consolidate the requirements. So far, we have seen that most partners struggle with the employment of traceability. Nevertheless, the specific problems seem to be quite different, depending on the development level, the applied processes, and the role of the partners.

1.2 Relationship to other CRYSTAL Documents

First of all, this document is mainly related to the use case description of the automotive domain:

- D301.010 WP3_1 Use case definition
- D301.021 WP3_1 Milestone Report - V1
- D302.011 WP3_2 Milestone Report - V1
- D303.011 WP3_3 Milestone Report - V1
- D304.011 WP3_4 Milestone Report - V1
- D305.011 WP3_5 Milestone Report - V1
- D306.011 WP3_6 Milestone Report - V1

The document also covers aspects from SP6 and is therefore also related to D601.010 (State of the art – Interoperability), D603.011 (Specification, Development and Assessment for System Analysis and Exploration - V1), D605.011 (Specification, Development and Assessment for AUTOSAR Tools & Components - V1), D610.011 (Crystal Variability Management - V1), D610.031 (Brick System Family Engineering Framework - V1), D611.011 (Specification Development and Assessment for Software Development Lifecycle Management - V1), D612.011 (Specification, Development and Assessment for

[1] http://www.cesarproject.eu/
[2] https://www.**mbat**-artemis.eu/
[3] http://www.artemis-ifest.eu/

Validation Models - V1), and D613.021 (Development of the simulation model data backbone as described in T6.13.1 - V1).

1.3 Structure of this document

This document is divided in 3 chapters, which focus on the introduction of the domain and the public use case AUTOMOTIVE, some interoperability challenges, and the introduction of the demonstrator. The descriptions in this version of the deliverable are very high-level and will be detailed throughout the project.

CHAPTER 2 - Use Case Description has two main parts:

The first part aims to give an abstract description of the public use case AUTOMOTIVE, its main purpose, structure, and scope.

The remainder of this chapter will shortly introduce the individual partner use cases – they cover various aspects of vehicle development.

CHAPTER 3 - Identification of Interoperability challenges:

This chapter introduces the interoperability challenges which have been identified in the partner use cases. The current description is just a first very high-level introduction, which will be detailed in the next version of the deliverable. The basic idea here is to collect a set of common interoperability challenges which should be covered by the Crystal project.

CHAPTER 4 - Demonstrator prototype:

This chapter shows the first ideas for our first automotive demonstrator. It mainly shows the ideas for the first iteration with an outlook on future activities. The scope of this demonstrator is the establishment of traceability links between requirements and model elements for systems as well as software development. This demonstrator will also evolve throughout the project.

CHAPTER 5 - Next steps:

This last chapter gives an outlook on the activities in the next phase of the project.

2 Use Case Description

The goal of this use case is NOT the development of an integrated tool chain for the development of a vehicle. Instead, we want to present the highlights of the six automotive use cases and as a result a set of best practices. The automotive use cases cover different stages in the development of a car – from powertrain design on vehicle level down to the development of microprocessors and software.

This means that this workpackage focuses on challenges arising throughout the entire V-model and additionally some of the main aspects defined in the CRYSTAL project including system analysis (e.g. timing analysis, requirement analysis), variability or variant management, functional safety, and traceability.

Figure 2-1 gives a high level overview on the scope of this use case. The 6 levels cover activities throughout the development process and are investigated here in more or less detail. The picture is taken from [Küpper, 2011], but has been slightly adapted. Level 0 covers requirements and design decisions on a high level of abstraction – the vehicle level. Level 1 targets the different vehicle modules (e.g. powertrain, chassis) and on Level 2 we focus here on the different powertrain elements (e.g. Transmission, Engine, Battery). On the next lower level the different elements are broken down in mechanical and E/E components. A control unit further consists of hardware and software, which are depicted in Level 4. Level 5 finally breaks the software system down to single software components.

Orthogonal to this description there are important aspects such as Traceability, Variability, System analysis, and Functional Safety. All these aspects span across the various levels and will be considered in the use case at least at one level.

The illustration additionally shows which use case covers aspects of the different levels. The concrete content of the single use cases is described in the remainder of this section. The corresponding interoperability challenges will then be shown in Section 3.

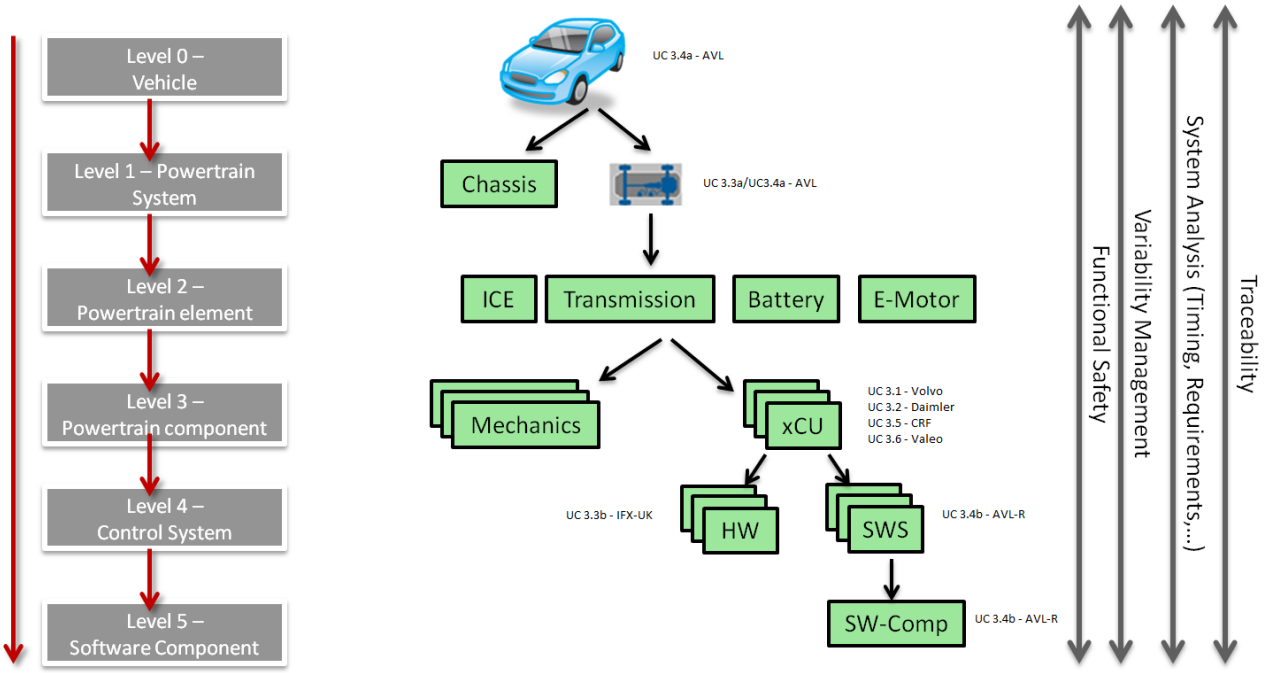

Figure 2-1: Vehicle development phases (based on [Küpper, 2011]) including the use cases which cover aspects of this level

2.1 Vehicle and Powertrain System Level

Vehicle level here covers everything from vehicle conceptualization down to powertrain engineering. Two use cases cover different aspects at vehicle level. Use Case 3.3 aims to increase quality and efficiency of powertrain systems & safety activities by applying model-based systems engineering (MBSE). Use Case 3.4 focuses on testing aspects. Details of the two use cases are given below.

2.1.1 Classical V-model aspects

Since several decades, the prominent V-model (suggested by Barry Boehm in the year 1979) is present in the automotive industry and represents the typical stages of vehicle manufacturing and the validation of these processes. A V-model has typically a central *development entity*, whose creation, developing, and manufacturing it is all about. In case of the automotive industry and according to our use case, this development entity is a *vehicle*. Figure 2-2 illustrates a typical V- model as applied in the automotive industry.

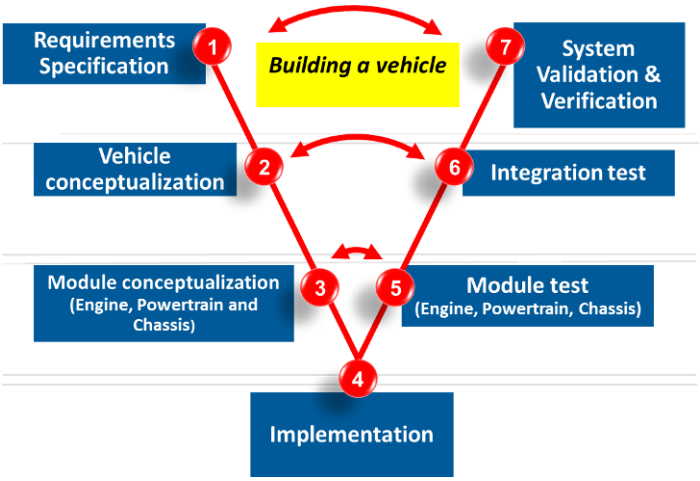

Figure 2-2: Classical V-model applied on the topic of building a vehicle

Based on a set of vehicle requirements, an engineer creates an overall vehicle concept (maybe based on previous projects with similar requirements). Then, a team of engineers continues with the conceptualization of the subparts (such as engine, powertrain, and chassis) followed by the development of concepts for even more details and so on. After the implementation of the concepts, first the modules are tested against their particular concept specification. Then they are integrated step-wise, which goes along with corresponding integration tests. Finally, the original requirements of the resulting vehicle are verified.

The additional aspect of virtualization of the vehicle and/or its components (i.e. by the use of simulation models) introduces more variants of this classical representation of a V-model for vehicle development. These simulation models enable the validation of certain design decisions already at an early stage – this is also called development frontloading. Development frontloading enables early comparison and validation of different designs. Independent of that, the goal of or the story behind these models is the same: *Building a vehicle*. A possible enhancement of the V-model illustrated above in form of a so-called W-model is illustrated in Figure 2-3:

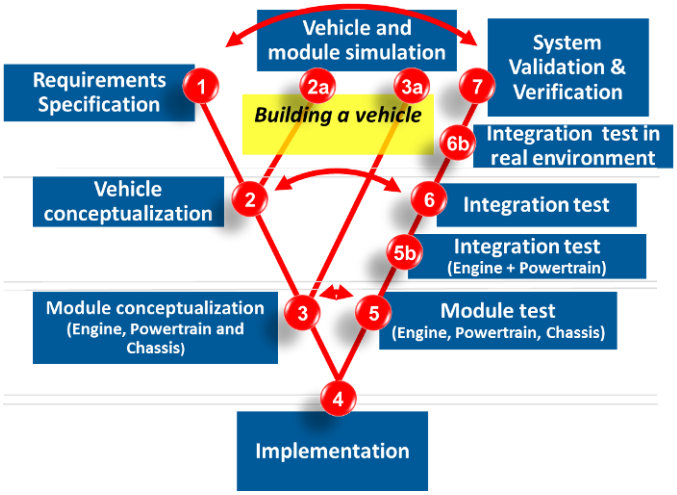

Figure 2-3: Early verification through frontloading

Figure 2-3 shows that requirement validation and verification becomes possible already at the *vehicle* or *module conceptualization* phase (2 and 3) by *vehicle and module simulation* (2a and 3a). For simplicity, we distinguish by now between three elementary vehicle modules: *engine*, *powertrain,* and *chassis,* because they are in the focus of the respective use cases 3.3 and 3.4. During module conceptualization, these modules are specified through design, parameterization, and calibration in an iterative simulation and test cycle. During these iterations, modules are specified at an ever more detailed level, e.g. by separating the engine model into two models, whereas the first one covers the physical aspects of the engine and the second one is a model of the engine control unit.

Through this stepwise refinement, the implementation phase (4) finally results in real physical components (engine, powertrain, chassis, control units, etc.) and software that runs on the control units (and may be automatically generated out of the simulation models).

These components and software units need to be tested in concrete module tests (5) according to the given requirements. In addition, intermediate integration tests may be added (5b), which combine for instance engine and powertrain testing, but still include a *rest vehicle simulation* of those parts that are not physically present in the particular testing scenario. In both scenarios additional equipment is needed to test the physical components, i.e., a so-called test bed. This test bed needs to be configured according to the vehicle requirements as well.

Development phase (6) is a full integration test. This test is accomplished with the use of a test bed as well and the environment of the vehicle (driver, street, weather conditions, etc.) still needs to be simulated. In order to overcome this simulation, so-called in-vehicle tests are performed, where a real driver is steering the vehicle on a real road and the testing equipment is built directly into a car (6b). This final test phase leads then to the system validation and verification (7) according to the given requirements.

Use case 3.3 focuses on the development of new powertrain solutions and the performance of system, safety, and requirements engineering activities based on existing solutions driven by specific vehicle goals and performance criteria. This means that this use case is mainly concerned with the left-hand side of the V-model including Step 2a and 3a. Respective safety information (ISO26262) is additionally required for the item definition (e.g. hazard and risk assessment, elicitation of safety goals etc.) early in the requirements analysis phase. In the next phase, the requirements will be modelled and analysed in more detail resulting in a model-based user requirements specification including a preliminary architecture definition in SysML as a basis for all following requirements engineering activities.

The verification of the architecture and operation / control strategy (2a and 3a) is supported by vehicle and powertrain simulation (i.e. AVL InMotion & Cruise). One important step focuses on the collaboration and

communication of requirements to discipline and development specific teams – locally distributed over several countries. PLM tools (e.g. PTC Windchill) support this activity including the investigation, further breakdown, and detailing of these specific requirements with a tight integration of a model-based system development approach (sub-system / component specifications) embedded within the PLM environment. ALM tools (e.g. PTC Integrity) support the software development process and therefore requires a tight integration to the PLM tool. Especially to support traceability and change management a mapping of functional requirements structure on a product structure is required. Since both structures are quite different, this introduces another interoperability challenge, which will be described in Chapter 3.

2.1.2 V-model aspects for Systems & Requirements Engineering

Figure 2-4 outlines the methodology of Requirements Engineering within system design as applied in UC3.3.

Traditionally, requirements engineering focuses on the definition and development of requirements based on the customer's "*wants & needs*" at the beginning of the development process, as well as at the beginning of each development generation. It describes the process of developing, cascading, and describing requirements of the system under development based on goals and targets (i.e. performance criteria, attributes, etc.) and the allocation of features/functions to elements/components and disciplines. RQ's come from different sources such as e.g. OEM/customer, legal standards/regulations, and the requirements engineering process itself (generation of requirements). Requirements management is the process of tracking and handling requirements throughout their lifecycle and focuses on the communication of requirements among all stakeholders (customer, supplier & internally) including the tracking of changes and impacts along the entire development process.

The initial point in this development process is the customer input, mainly in terms of use cases describing the system under development. Additionally vehicle goals, performance criteria, and/or technical boundary conditions serve as a base information for the derivation of technical requirements. These are further split down into functions/features, as well as the vehicle sub-systems, elements/components, legal requirements, state of the art requirements, environment requirements, and quality/process requirements. Furthermore, according requirements are specified for each level, which serve as an input for all subsequent development activities. This basic methodology follows the V-Model development approach (see Figure 2-4) that is applied within the development process for every development generation with increasing maturity of the system under development.

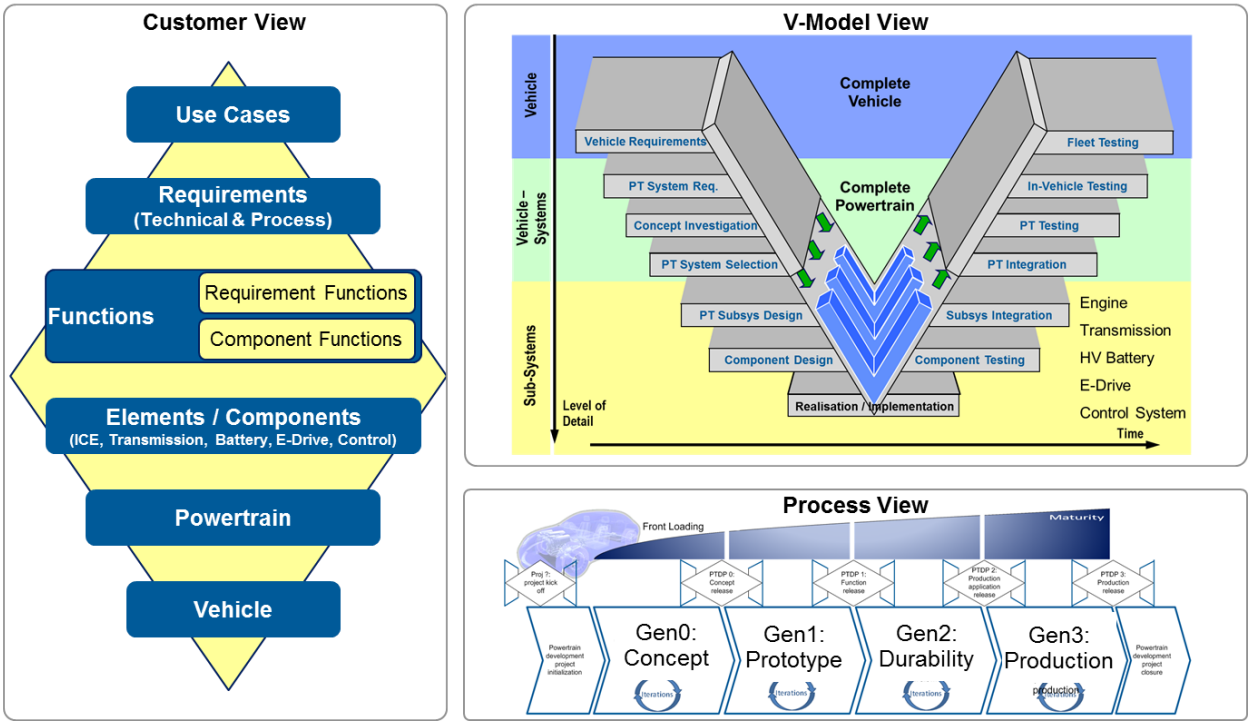

Figure 2-4: System & Requirements Engineering Method Overview

Requirement document vs. Specification

Requirements engineering is one of the main activities in system and software design. As mentioned before, requirements on one level are the basis to derive according solutions (in terms of architecture, i.e. components & functionality) for the following downstream level. This means that the developed solution satisfying the defined requirements is also the basis for the derivation of additional requirements on a more detailed level. This is an iterative step starting at vehicle / use case level down to hardware and software component level. The methodical approach is always the same, however different development teams with individual software tools and specific engineering methodologies are involved. Therefore two types of documentation are required as illustrated in Figure 2-5:

Description of requirements across the different levels in Requirements Documents:

- Requirement Document for "***P****roduct*" – e.g. Technical Specifiaction of Product (TSP) "*Vehicle*"
- Requirement Document for "***S****ystem of Product*" – e.g. Technical Specifiaction of System (TSS) "*Powertrain System*"
- Requirement Document for "***E****lement*" – e.g. Technical Specifiaction of Element (TSE) "*ICE*"
- Requirement Document for "***S****ystem of Element*" – e.g. Technical Specifiaction of System (TSS) "*ICE Oil Circuit*"
- Requirement Document for "***C****omponents in Element*" – e.g. Technical Specifiaction of Element (TSC) "*ICE Crankcase*"

The responsibility for the creation of these specifications differs depending on project type and scope. In general, system development teams are responsible for "*product*" and "*system of product*", as well as partly for the specification of the "*element*" (for elements there may be an overlap with other engineering disciplines). The creation of "*systems of elements*" and "*components of elements*" are in general within the responsibility of element teams.

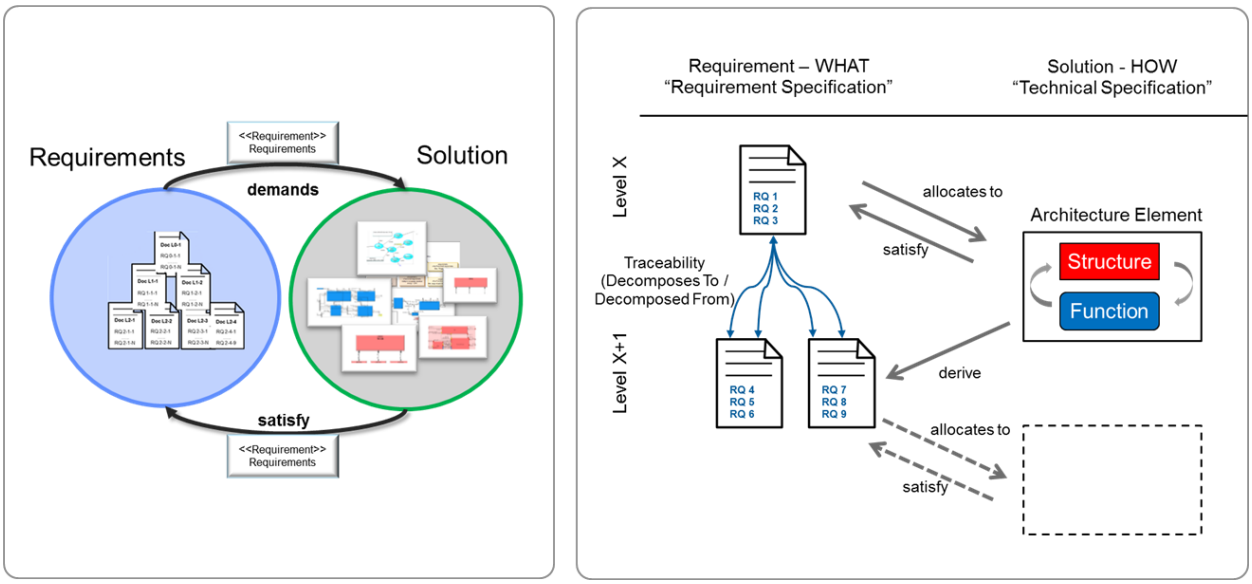

Figure 2-5: Requirements document vs. Specification

Requirement documents and specifications include information in order to understand the dependency between:

- Requirement & Solution by knowing/understanding which architectural element is allocated to the specified requirement (and respectively will satisfy the requirement). This is depicted in the requirements document.
- Requirements among each other by knowing / understanding potential effects of changing an individual requirements --> Impact/Traceability Analysis. This dependency is specified in the Requirements Management Tool by defining which downstream requirements are decomposed from an individual requirement (and vice versa which requirements decompose to from an upstream requirement). This is described in the specification.

The described requirement engineering approach also supports related activities in terms of functional safety. Each E/E-System that is able to cause failures which can harm people in the top level system is called safety relevant system. This classification is usually done by the customer. The customer provides the already identified safety-critical architectural elements, or at least the already analyzed safety goals. A safety goal is a top level functional safety requirement (e.g. avoid unintended torque). Based on the specified safety goals quantitative safety analysis shall be started by the system safety developer/functional safety manager to identify failures caused by the item under development that are able to lead to a violation of the specified safety goals. This analysis shall be done on each level of development (Level 0 – Level 5) that is within the scope of the project. The next step is to specify safety measures/safety mechanisms to avoid, mitigate, or handle the identified safety-critical failures. These safety measures/safety mechanisms shall be specified by functional safety requirements, technical safety requirements, and process. Each safety-relevant requirement inherits a safety integrity level that depends on the use case of the system.

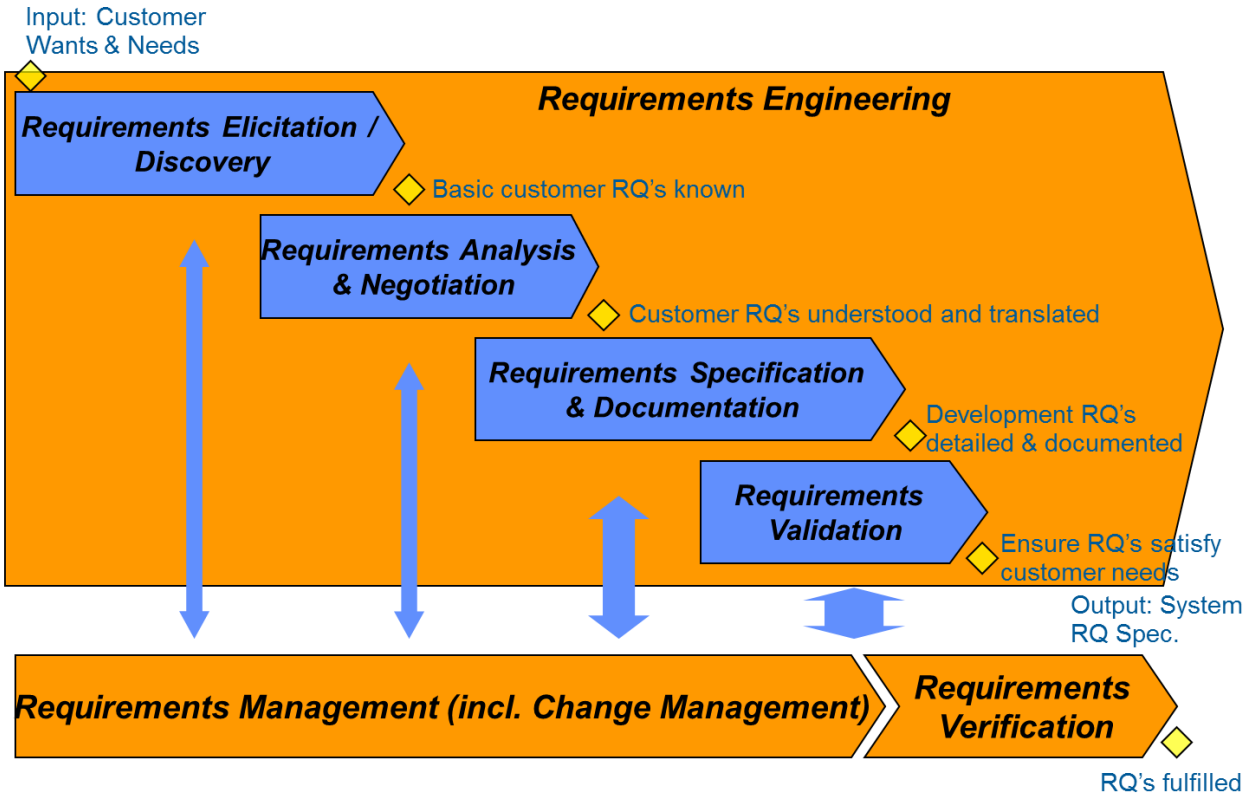

Figure 2-6: Requirements Engineering & Management Activities

Figure 2-6 outlines the main Requirements Engineering and Management activities. The following table (Table 2-1) outlines the individual activities incl. required general input and generated output information.

Activity Name	Input	Task	Output
Requirements Elicitation / Discovery	Customer "wants & needs", such as e.g. system goals, development targets, technical and organizational boundaries (e.g. project plan), contract, benchmark…	Identification and collection of relevant input that is required to start the Requirements Engineering process at project start	Collection of relevant content to serve as input to develop requirements incl. open questions to be clarified with customer – basic customer RQ's are known
Requirements Analysis & Negotiation	Collection of relevant content	Analyzing input in order to understand the meaning and purpose of the content. This includes e.g. questions, discussion, negotiations,	Structured collection of customer input incl. sources, stakeholders and answers to open questions clarified with all stakeholders

		detailing etc. in order to translate the information into a company-specific language	
Requirements Specification & Documentation	Structured and understood customer input	Documentation of customer input and deviation of further requirements (as the main technical activity of the Requirements Engineering process). Documentation of RQ's is to done in Requirements Management Tool (PTC Integrity)	All requirements are documented and described in Requirements Management Tool (PTC Integrity) according to RQ Meta model incl. rational, acceptance criteria, and traces.
Requirements Validation	Intended "Solution" (e.g. specification, simulation model, etc.)	Validation in terms of checking that the system under development will result in a solution that meets customer wants, needs & requirements	Statement that proposed solution will meet requirements verified by e.g. review, simulation, etc.
Requirements Verification	Developed "Solution"	Verification in terms that the "solution" fulfills the specified requirements verified through testing / V&V	Documented test results to proof that solution fulfills requirements

Table 2-1: Overview of Requirements Engineering Activities

Note: This is NOT a strictly sequential description of activities with respect to the development process. It is an overview of required activities that may take place in parallel, by one or more roles/persons and be repeated/detailed as required by the project scope.

2.1.3 Testing V-model aspects

The overall goal of the public use case is to build a vehicle. Testing is an important aspect for all development stages – including the vehicle level. Applying appropriate test methods, devices, and tools throughout the development processes is essential. Adequate testing procedures, techniques, and environments finally should verify the given vehicle requirements. UC 3.4 especially focuses on these testing aspects.

In addition to the given *vehicle requirements*, specific *testing requirements* enhance the list of requirements. These requirements are based for instance on a certain testing specification such as the WLTP[4]. Such specifications define concrete test-runs, the accuracy of measurement devices, formulas that have to be applied for measurement result analysis, and so on. The needed *test environment* (test bed including measurement devices, test-run input vectors, calibration variables, etc.) is then derived from these testing requirements as well.

The concrete configuration of this test environment depends then on the concrete position within the V- or W- model for *building a vehicle (see Figure 2-3)*. For instance, for early simulation phases, simulation models (e.g. for control software and virtualized physical vehicle components such as engines, etc.) are fully sufficient, whereas after the implementation phase, various test bed settings are required for a stepwise substitution of simulation models by their physical counterparts. Furthermore, the selection of development *and* testing tools may differ as well, because tools are often more specialized for a certain development phase. In other words: *Every testing phase may come up with its own tool set*. It is one *key challenge* to share data across these development and testing phases.

Setting up test environments is a complex task and needs to be easily adaptable to the requirements of the current position in the development process (i.e. the V/W-model) including additional boundaries (or testing requirements) coming from specifications such as the WLTP.

This means that besides the vehicle and model conceptualization and implementation on the one hand (left side of the V-model) and the actual test case execution, system validation and verification on the other hand (right side of the V-model), a lot of knowledge and efforts have to be spend on *test case conceptualization and implementation* as well. One may say that this *test case conceptualization and implementation* is a further sub-development step of the overall assignment of *building a vehicle*. Depending on the concrete position in the V-/W- model, however, a particular test case conceptualization and implementation has its own selection of testing processes and techniques in form of testing tools and methods as stated above. Therefore, *testing a vehicle* is a development task by its own and can therefore be represented by a separate V-model. In contrast to a classical automotive V- or W- model focusing on *building a vehicle*, an alternative V-model that focuses on *testing a vehicle* would look as illustrated in Figure 2-7.

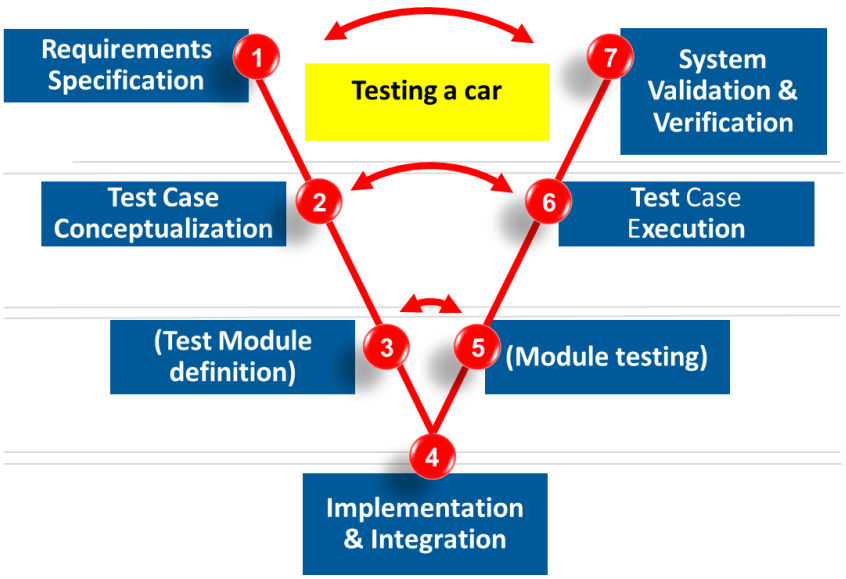

Figure 2-7: V-model for the testing process

[4] **WLTP** stands for **W**orld-wide harmonized **L**ight duty **T**est **P**rocedure and is currently available as a draft that will lead to a standard specification for emission legislation in the automotive domain in the near future.

Besides the already mentioned requirement specification and test case conceptualization phase, the latter one has to be separated in further sub-modules, similar as done for the vehicle conceptualization phase. However, instead of the vehicle itself, the test environment is now in the focus of the V model and thus these sub-modules differ significantly.

A general pattern in this test case conceptualization is the unit under test (UUT) calibration as shown in Figure 2-8.

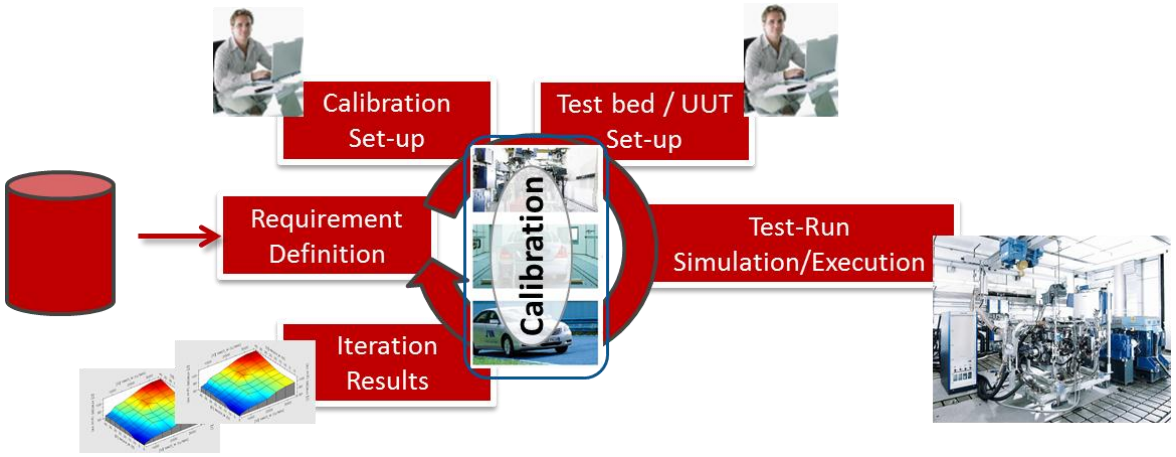

Figure 2-8: Test and calibration iteration pattern

The *test case conceptualization* is divided into two set-ups (*test modules*): the calibration and the test bed set-up. The test bed set-up configures the testing environment (including sub-modules such as required measurement devices, UUT configurations, etc.), while the calibration set-up is specialized on tuning selected parameters of the UUT in order to fulfil the given set of requirements.

The specification of these set-ups with all their sub-modules finally leads to the implementation and integration of the overall test set-up, which can be executed with a set of given test-runs.

In many cases, the test modules *test case execution* and *calibration* are clearly separated: During a test-run execution, measurement results are linked to given test run input vectors. These pairs of date are used in corresponding calibration tools to create a calibration model that interpolates even not tested constellations and therefore supports speeding up the calibration process as a whole.

2.1.4 Mapping the *W-model "building a vehicle"* to *test phases*

Figure 2-9 illustrates a mapping of the W-model about "building a vehicle" to test phases. This mapping and the resulting test phases are then described in more detail.

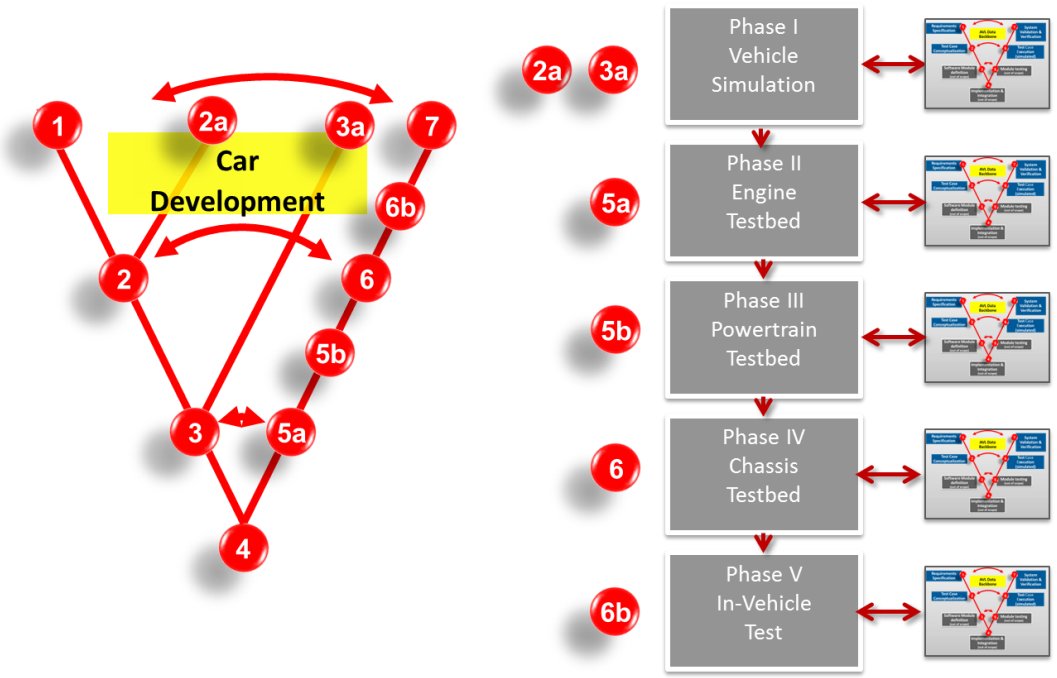

Figure 2-9: Mapping the W-model to vehicle test phases

Let's first describe the W-model "building a vehicle" once more under the aspect of testing.

1. *Vehicle conceptualization (2):* This phase can be accomplished by virtually modelling the vehicle according to its *requirement specification (1)*. These requirements can then be validated on vehicle level by *simulating the vehicle model for certain test cases (2a)*.
2. *Engine, powertrain, and chassis conceptualization (3):* Once again, simulation models are an essential aspect in the conceptualization of these three vehicle components. Associated simulation models are designed with a high level of detail for each of these parts. The execution of these simulation models according to appropriate *test cases (3a)* validates the results of the overall vehicle conceptualization.
3. After the *implementation phase (4)*, the concrete sub parts of the vehicle need to be tested against its conceptual specification. In our use case, this is done for the *engine (5a)* and the *powertrain (5b)* with appropriate *test beds*. The powertrain already includes the engine, which was tested in the previous testing phase (5a). In the *integration phase (6)* the subparts are assembled and verified using an *integration test (6)* that has to validate the overall *vehicle conceptualization (2)*. This is usually done by a chassis test bed, where the concrete environment is still simulated (usually even the vehicle driver).
4. Finally, the virtual driver and environment are replaced by real ones and tests are performed by so-called *in-vehicle tests (7)*, which finally validate the *requirements of (1)*.

Based on this description, testing phases can be assigned to specific development steps of the presented W-model, whereas each testing phase is described by its own testing V-model as illustrated in the picture above. In our use case, five different *testing phases* can be divided from the description above:

1. *Phase I:* Performing vehicle simulation (associated with development phases *2a + 3a*)
2. *Phase II:* Applying an engine test bed (associated with development phase 5a)
3. *Phase III:* Applying a powertrain test bed (that includes engine testing; associated with development phase 5b)
4. *Phase IV:* Applying a chassis test bed (associated with development phase 6)

5. *Phase V:* Performing an in-vehicle test (associated with development phases 6b)

These development phases are by no means complete (further phases such as HIL/SIL testing could be interlaced). However, additional phases do not abandon the principle concept of separate testing V-models linked to a specific vehicle development phase. In addition, however, this does not exclude a possible overlap of methods and tools across the five testing phases (for instance, the simulation of the environment takes place in any test scenario except the in-vehicle test). The *interoperability challenge* is now the transfer of the created data set from one development phase to the subsequent, in order to foster reuse and therefore improve the efficiency of test conceptualization. Some possible concepts are described in Section 3.1.

2.2 E/E System Level

EE development on system level at Daimler (UC 3.2) includes the disciplines Requirements elicitation, -engineering, and –management, as well as system level testing and the creation of system architectures as shown in Figure 2-10 at the end of this chapter. This means having two engineering directions in our development V-cycle from a starting point “system requirements” (1) to result in:

1) Verification relation from system level testing (3)
2) Realization relation from E/E system model (2)

A very important task which is done on E/E system level and influences the whole development process is the decomposition of system level requirements into component/software or sub-system level requirements to the allocation of resources to implement the systems features. Mainly this is what we call the system architecture. On system level, we already know vehicle level requirements – imaginable as “Customers view” on a black box system. The customer’s view is what we’re going to address at E/E system level and decompose this to consumable portions of hardware/software requirements as illustrated in Figure 2-10. This will result in single technical requirements (technical view) which are well addressed by specific parts (HW&SW) with specific responsibilities.

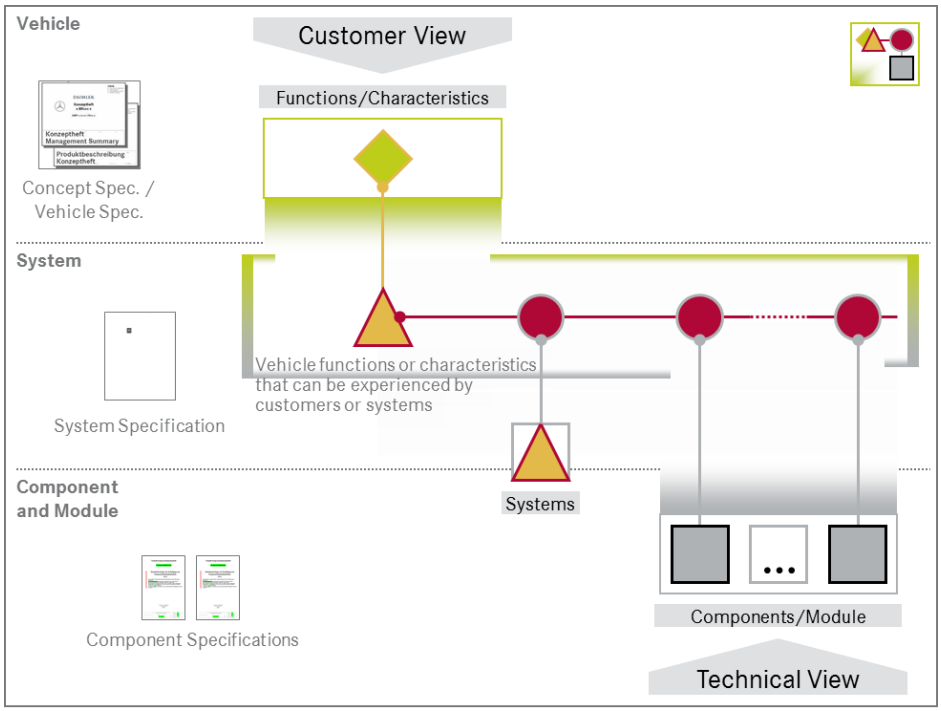

Figure 2-10: Mapping customers view to technical view

Supporting decomposition relationships (traceability links) between system level requirements and the component level, enables us to document the system architecture from a functional point of view. This will help re-using the system in different variants and enables impact analyses in case of changes to the system.

Furthermore, it's possible to have a 1:1 match between system level requirements, system level test cases, and validation criteria.

The E/E system level is going to be the point of entry for

a) Variant management processes
b) Change management processes

as depicted in Figure 2-11.

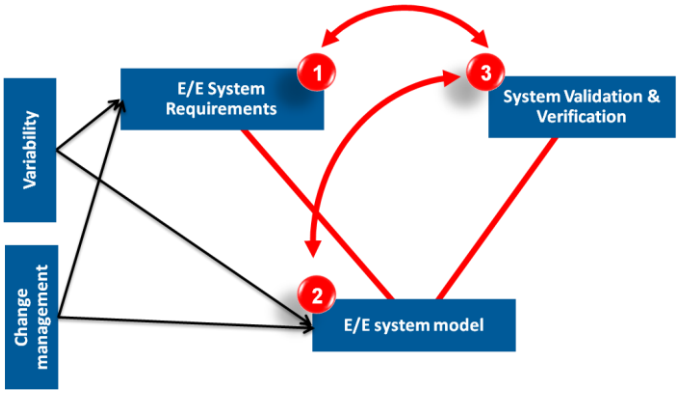

Figure 2-11: E/E system level phases and relations

For variant management, system level features, characteristics, and properties are being documented in a feature tree. This feature tree can also contain relations & constraints (i.e. feature X conflicts feature Y), so that engineering know-how becomes explicit. All variable system level requirements have to be connected to the corresponding part in the feature tree (which causes the difference).

Then selections of features will result in selection of engineering artifacts connected to these features (see Figure 2-12: Variant management at E/E system level).

In the example mentioned below, the kind of roof causes differences in requirements, implementation and test for an ECU. The opening function is related to the feature “sunroof” which is available in sedan and station wagon but not in coupé and cabriolet.

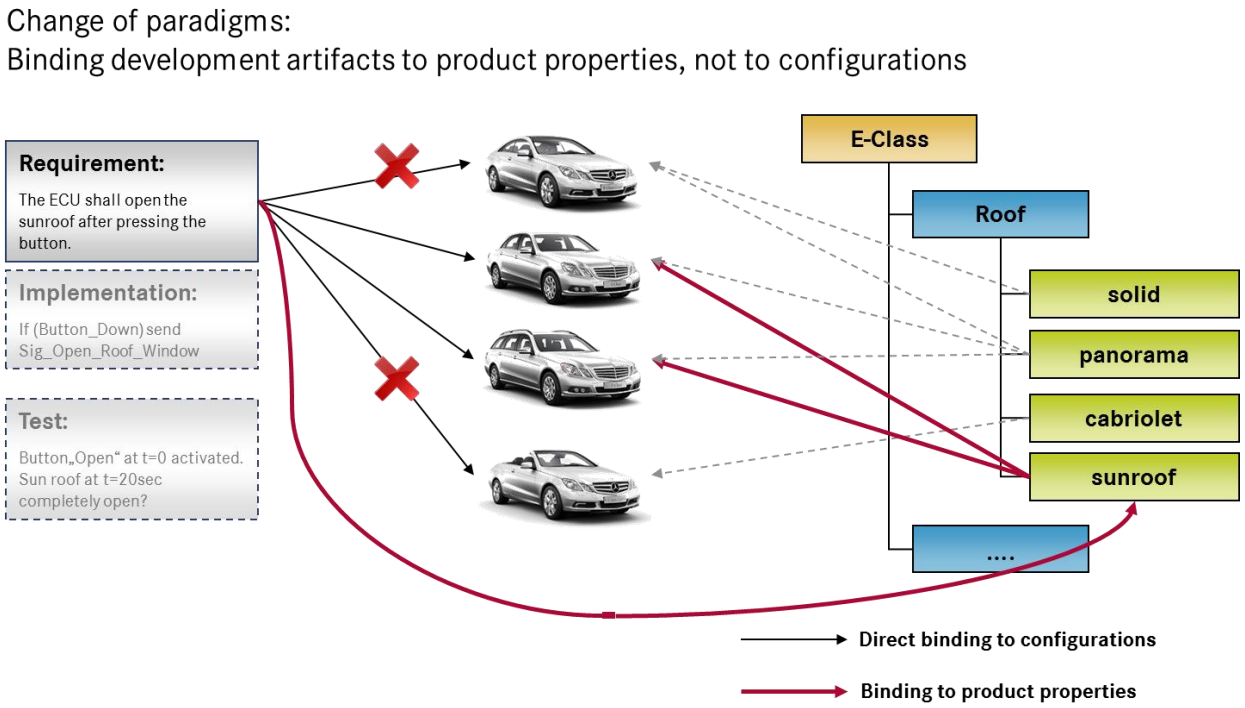

Figure 2-12: Variant management at E/E system level

For the often required process of change management, it’s also a proven way to trace changes from a system level point of entry, because impact analysis has to identify all connected requirements and HW/SW level components which are affected. Possibly several different lower scaled change processes have to be initiated after having identified what’s impacted.

In terms of ISO26262 the system level is often being addressed by the “item definition” and so becomes the root element for safety relevant systems, regarding change management, but also traceability and verification management in functional safety.

2.2.1 Functional safety according to ISO26262

In the automotive domain, different aspects have to be considered at various levels within the design workflow. One of them is functional safety, which is the main focus of UC 3.5 conducted by CRF as shown in Figure 2-13. Functional safety describes all activities in which the functional characteristics of a system (a vehicle) are analyzed and assessed in relation to their potential risks during the normal usage of the vehicle by the customer. This process is described in the ISO 26262 standard [ISO, 2011], which is divided into ten specific parts and contains all the prescriptions necessary to attempt this kind of assessment.

The standard provides a complete and rigorous workflow for attempting this aim, starting from the item (system or ensemble of systems to implement a function at vehicle level) definition and the Hazard Analysis and Risk Assessment (HARA) and down to the definition of specific levels of safety requirements including technical ones, considering both hardware and software components.

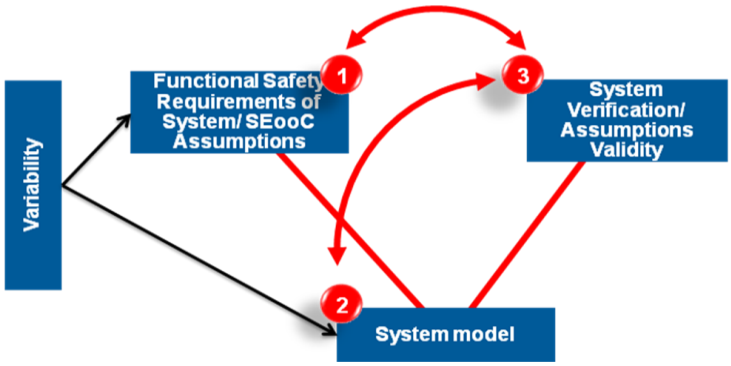

Figure 2-13: Functional safety in the public use case for SEooC at concept level

The standard itself, actually, encompasses all the subsequent processes and activities involved in the item design and production, providing prescriptions for item testing and integration into the vehicle and its final validation, covering also the aspects related to the processes and activities after production as operation, service, and decommissioning.

In this interaction some general issues arise considering the relationships between the requirements as shown in Figure 2-14 and described below:

- The domain design workflow (red arrow) starts from the initial specifications (Functions/assumptions, Scenarios/Situations), e.g. from Customers/Marketing analyses, and generates a preliminary architecture this workflow uses tools for modeling and simulation at mathematical/physical level.
- A tool/framework (e.g. Enterprise Architect) represents a special domain context (e.g. Road Vehicles Functional Safety → ISO 26262 standard workflow: green arrow), which covers special requirements (e.g. Functional Safety Requirements).

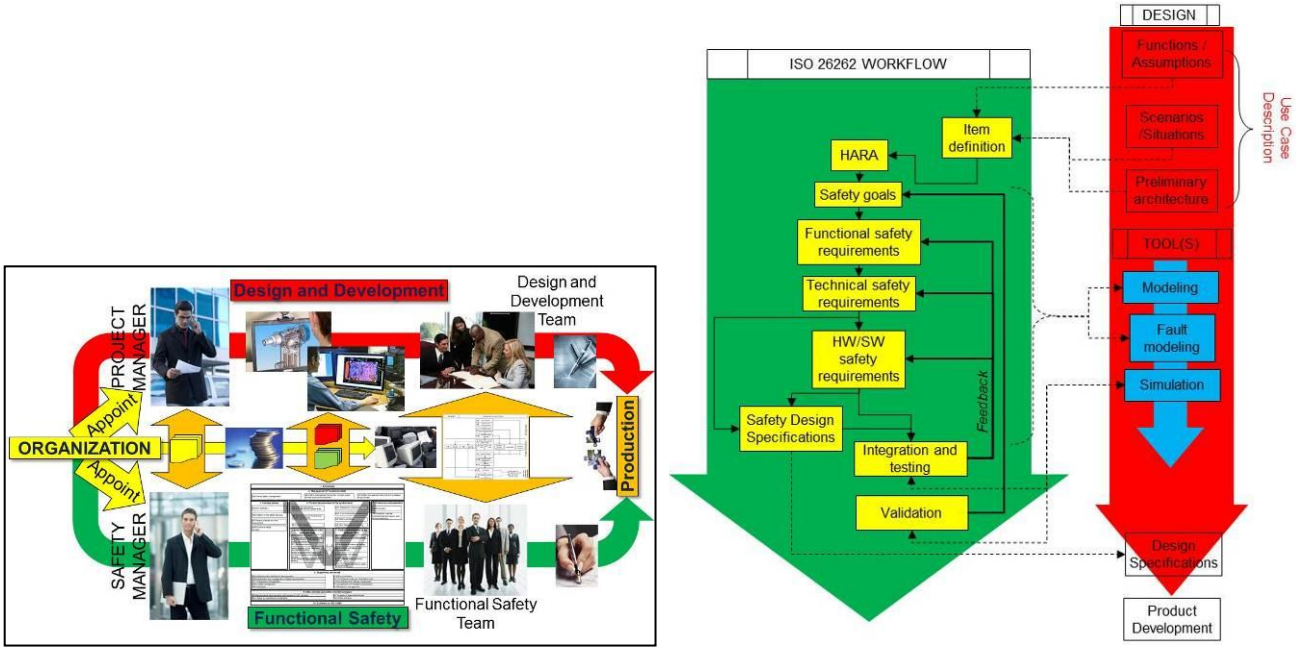

Figure 2-14: Functional safety and Design flows in parallel

The two parallel processes must exchange information at requirements level, for fulfilling the general functional targets and the special requirements (e.g. Functional Safety Requirements) of the developed product.

When a change occurs, this must be reciprocally reflected into all the related steps of the two processes: for example a change can occur during the special requirements assessment (e.g. a functional safety requirement is not validated and requires a modification) and the corresponding functional requirement from the targets must be revised, then it is necessary to trace both requirements and the propagations of modifications along the two parallel processes.

One possible solution could be the application of “system off the shelf” development in the context of functional safety. A SEooC (Safety Element out of Context) according to ISO 26262 as illustrated in Figure 2-15 can be applied with a tailored process in order to have: specifications, assumptions, hazard analysis, functional and technical safety concept with related verification according to ISO 26262, involving the inputs from the design and the traceability of the results. The SEooC is typically a system or component that is designed and developed without a specific vehicle reference (out of context). Safety considerations are based on assumptions for which the safe behavior can be guaranteed. The aim of such type of development is to reduce the cost and the time for production of the vehicles, assuring a set of available components suitable for integration, once fulfilled the assumptions related to their generation. This is also one aspect which should be covered by this use case.

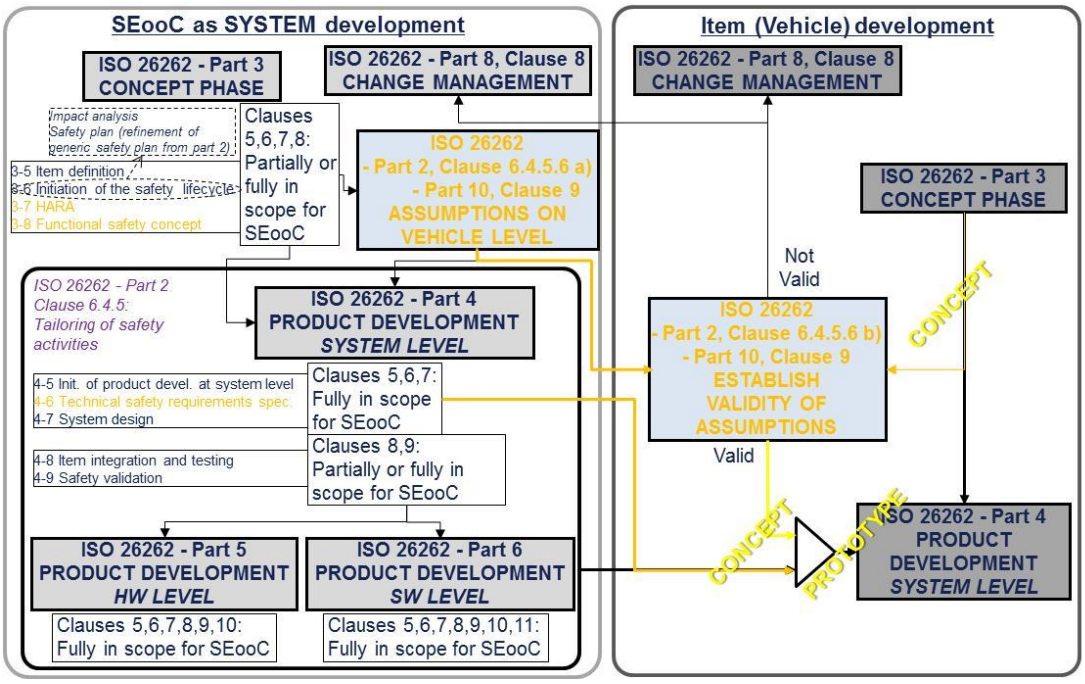

Figure 2-15: SEooC (Safety Element out of Context) example from ISO 26262

The objective of UC 3.5 is the conceptual design of an automotive climate system in terms of models. The target system is potentially safety-critical, and therefore functional safety constraints as well as functional needs have to be considered.

The starting point is a current in-vehicle system that must be upgraded. The envisaged improvement is something quite new, at least in the automotive domain. It introduces some potential risks in relation to the safety of the vehicle, due to the presence of a potentially flammable and toxic refrigerant fluid, employed with the aim of reducing the greenhouse emissions, according to the new international normative.

The application of the functional safety analysis impacts the existing design information (modeling level). This means that there has to be a new structure of requirements which takes into account the new safety relevant characteristics.

According to the previous descriptions of the workflows the interaction between the two kinds of modeling (functional and safety functional) must find a common path of integration in order to produce results reciprocally traceable.

Figure 2-16 resumes the V cycle involved in the application for the use case, but the structure of the workflow is general.

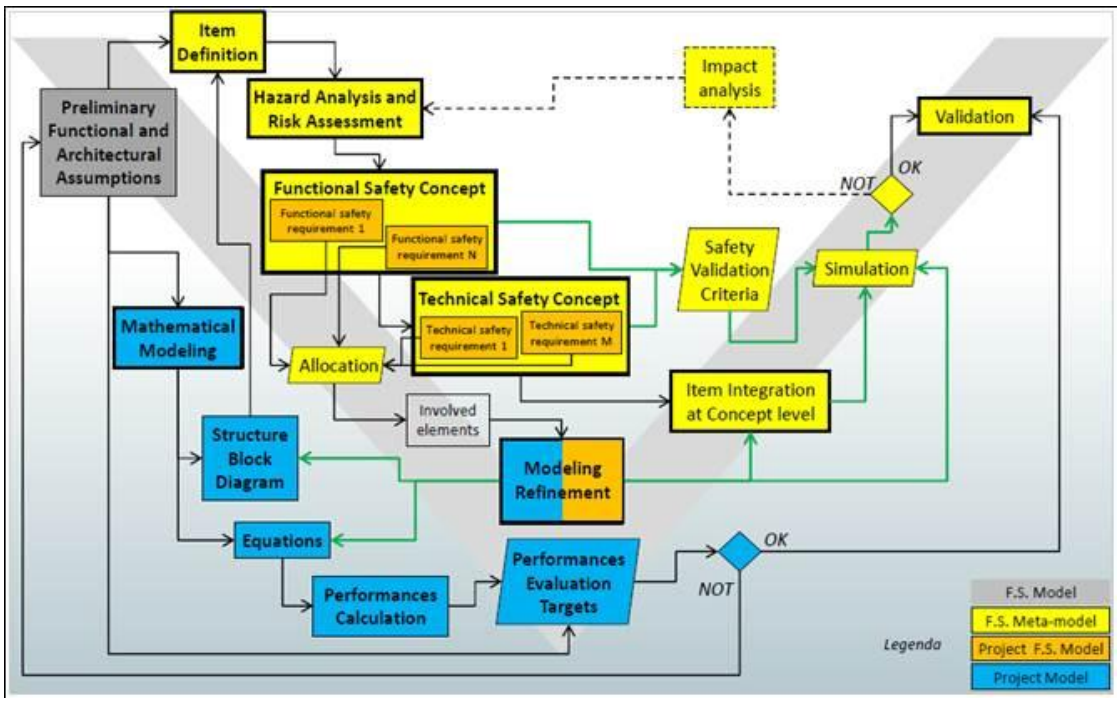

Figure 2-16: General methodological approach (F.S.: Functional Safety)

2.2.2 Timing Analysis

The Volvo use case (UC 3.1) covers the EE development process from vehicle level (end-to-end functionality) down to implementation level (software). It contains several sub-use cases dealing with specific development activities such as system behavioral modeling, architectural design, test case generation, AUTOSAR application development, AUTOSAR ECU integration & generation and timing analysis. The timing analysis sub-use case is further described in this section.

The main steps of the timing analysis are depicted in Figure 2-17. The first two steps represent the timing modeling, first on the system model (step 1) and then on functional components in the timing analysis tool (step 2). After that, the actual timing analysis can be performed (step3) and the results are made available /linked to the system model. The steps are elaborated below.

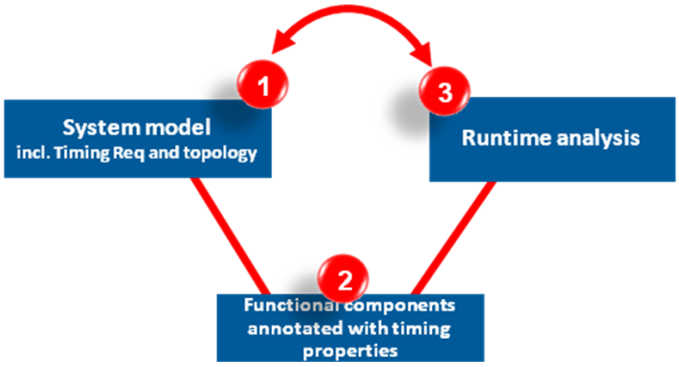

Figure 2-17: The main steps of the timing analysis in the public use case.

Step 1: Already when the system model has been created, some fundamental timing requirements can be defined. This can typically be end-to-end latency requirements involving a sequence of system/design components and signals, and the maximum allowed latency for the sequence. Required or desired execution periods for the system/design components could also be defined.

Step 2: Functional components in the timing analysis tool are annotated with timing properties. Either the model is manually created and then model elements are linked to corresponding elements in the system modeling tool, or the model is partially automatically created based on data exchange; see also Section 3.2. The timing analysis tool typically at least requires the system/design components, signals from the communication infrastructure, system topology, and the already defined timing requirements, but could also need additional network parameters and additional timing information such as priorities and Worst Case Execution Time (WCET). The network parameters may require information from the software development, which means that such timing analysis must be performed on later phases of the development than what is commonly defined at the system level. Priorities and WCETs may be assumed based on rules of thumbs.

Step 3: The system is analyzed for compliance with the end-to-end latency requirements and the results are linked with the e2e timing requirements.

2.3 Software Level

UC 3.4b is mainly concerned with the software development process as depicted in Figure 2-18.
The development process starts with the activity *analysis of software system requirements*. These *software system requirements* are delivered by the customer and have to be analyzed and put into a form to be used as basis for the software development. Based on the *software system requirements*, the *software system test cases* and the *software system architecture* are being defined. The *software system requirements* are assigned to the *software components* and handed over the function algorithm developers and software algorithm developers for further analysis.
Software component development starts with algorithm development and specification of the detailed *software component requirements*. This step already involves the set-up of a model (e.g. Simulink model) and its documentation. Based on the *software component requirements*, the *software component* is developed. After this development step, the software components implement the functionality specified in the software requirements.
On the right side of the V-Model, the implemented software components have to be tested according to the defined test cases. The test report is delivered as well as the SW component including the documentation.

If all software components have been tested, they are integrated and validated with the final *software system tests*. Finally, the software system including the test results is delivered to the customer.

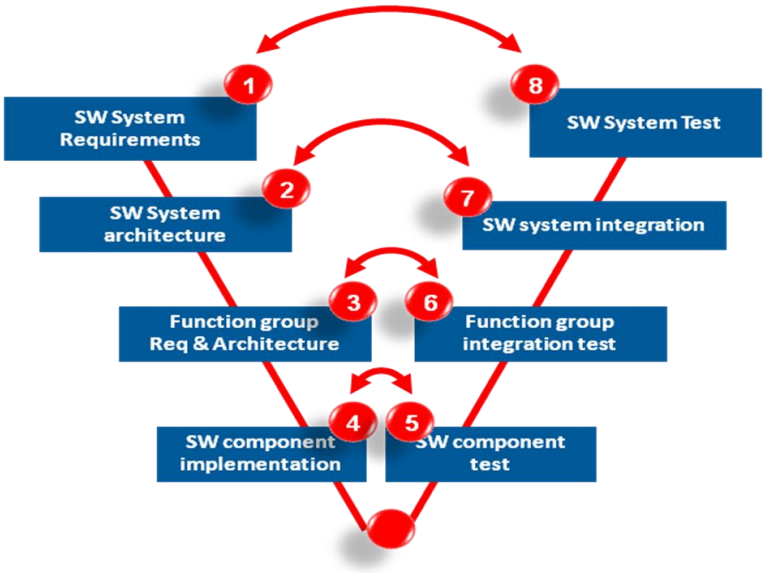

Figure 2-18: Software development process at AVL-R

2.4 Hardware Level

Infineon is a semi-conductor company and it is within the scope of implementing the system-on-chip system hardware.

- SOC system
  - Core processor + system on chip from pre-silicon to post-silicon, firmware testing and software
  - Automotive powertrain/ADAS/Body etc
  - Multi-product family across 65 and 40nm (currently 6 products in implementation 5+ incoming)
  - Multiple customers for the same products
  - Safety Accreditation with ISO26262 and other standards

Infineon will be addressing the V-model shown in Figure 2-19.

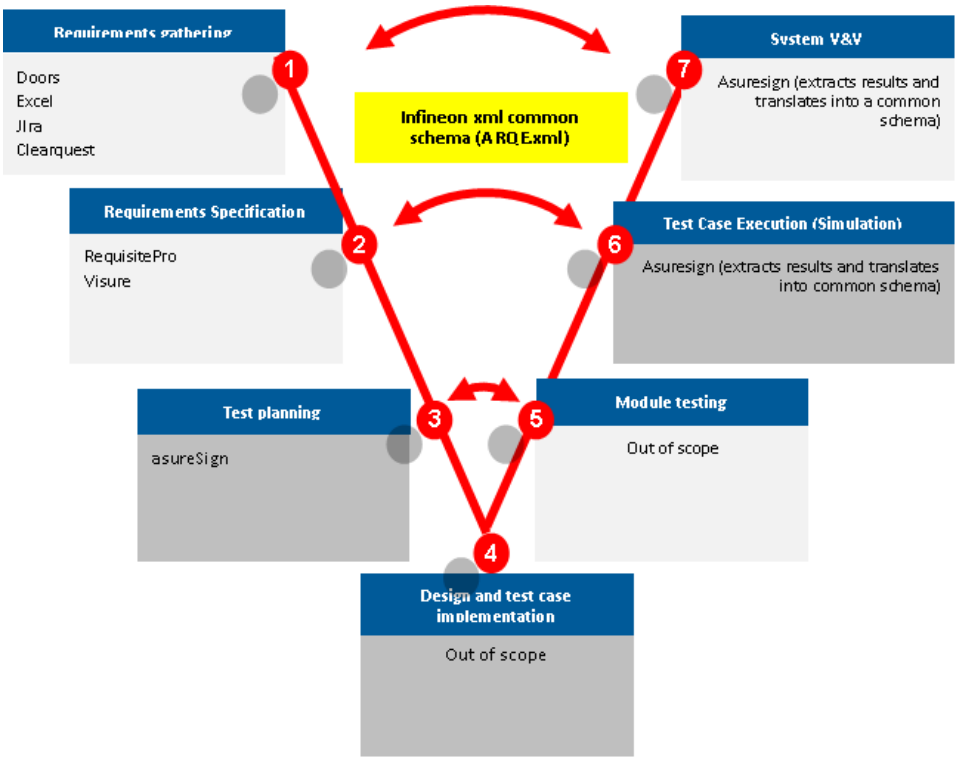

Figure 2-19: IFX-UK development process

1 Requirements Elicitation: Link requirements to sources (customers/change management systems).

- Formalize requirements for quality

2 Requirements Specification

- Link *to source*
- Link to 'intent to implement' (Target Spec)

3 Test planning

- Link to Requirements
- Link to results

6 Test Case Execution

- Link *xCU Parameters, Testbed Configurations* to Development Stage 2

7 System Validation and Verification

- Link *Requirements, Test Results and Validation Results*
- Details on how to replicate in CCM (Change and Configuration Management)

The interoperability issues we will be addressing are the following:

1) Ensure traceability between external and internal requirements – we shall be investigating the Claims language and look at a semantics checking solution which will check by the claims language rule set for adherence to ensure good quality semi-formal requirements. This may involve interoperability between the change management system, a semantics checking systems and a Requirements Management tool. We shall also investigate integration within ReQif to external customers who are primarily DOORS driven.

2) Requirements Specification, here we shall ensure that the 'intention to Implement' the requirement is linked within the flow – Infineon have since simplified the flow so that the intent to implement is an inset table of requirements from the decomposed internal requirements within the Requirement management database. The Requirements will also be translated via the ARQE.xml (internal Infineon xml schema which we have proposed as one IOS solution), into the test planning and proving tool called 'asuresign'.

3,6 & 7) The test-planning, results and translation back via the ARQE.xml format (our IOS solution) into the requirements engineering flow will be implemented by the current external tool on trial 'asuresign', this will continue development within the scope of this project to extend to all requested domains.

2.5 HW/SW Integration

Use Case 3.6 aims to port an existing application running on a single-core microcontroller to a multi-core microcontroller. Two main aspects have to be taken care of: First, the application shall run on top of an OS and Basic Software (BSW), which is compliant to AUTOSAR 4.0[5]. Elektrobit will be in charge of the BSW and the OS and will furthermore provide a tool called Tresos Studio, which is used for the development of AUTOSAR software. The second aspect is that the ECM (Engine Controller Management) SW shall be certified according to ISO26262 up to ASIL D. This requires a dedicated safety concept, which considers all aspects of multi-cores.

One main challenge is to keep the performance. Multi-core controllers use different memory mapping models (cache, shared and unshared memory) and software running on different cores needs dedicated services in order to communicate – thus, introducing a communication overhead.. Furthermore, the software has been designed for single-core architectures, which means that parallelism has not been in mind. Simply using sequential code on a multi-core processor has no performance gains, because calculations are still done in a sequential order on one core.

Figure 2-20 illustrates the applied software development workflow.

[5] http://www.autosar.org/

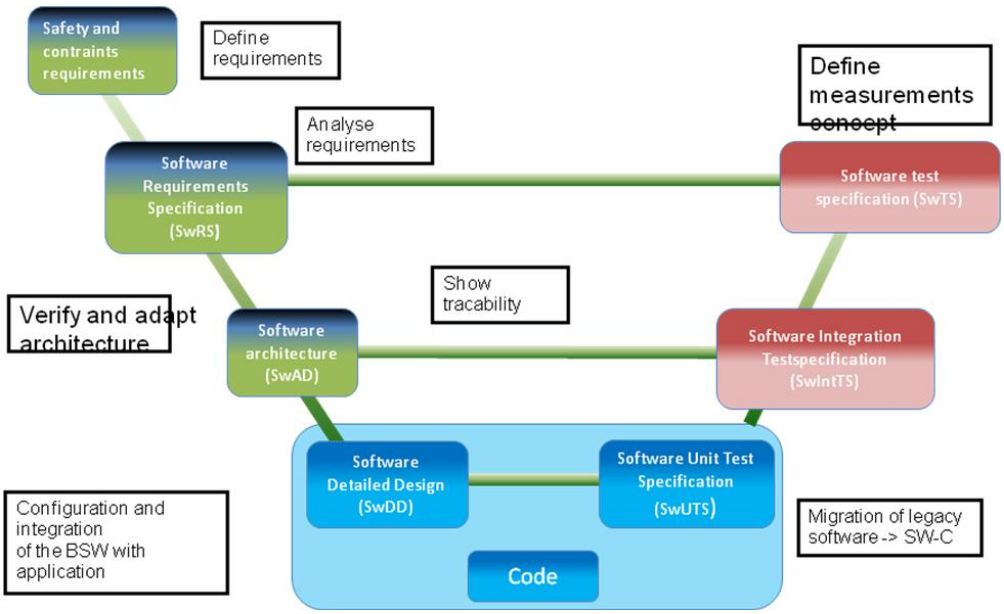

Figure 2-20: Software development workflow

Many steps have to be followed to achieve these goals:

1) Elaboration of current state of the art
2) Microcontroller selection using roadmap and selection criteria which can be shared with the community
3) Detailed specification of the use case
4) Creation of SW architecture which considers parallelism
5) Implementation
6) V&V (Verification and Validation). We intend to check that real time performances are still met and ensure ISO26262 compliance. The target is to have 100% coverage of requirement testing measured thanks to the traceability. Of course basic functionalities as well as global performances will be verified. This V&V will be performed on target using dedicated test benches.

The integration on target will be done step by step and is a joint effort between both companies, Elektrobit as well as Valeo. One basic aspect here is the support for traceability.

2.6 Summary

This Section has introduced the use cases from the automotive domain. All these use cases are part of the public use case AUTOMOTIVE, since this public use case will be used as a medium to demonstrate the achievements of the automotive domain to the public. We have identify some potential interoperability challenges, which are shown in the next Section. These challenges will be elaborated in more detail in the next versions of the deliverable.

3 Identification of Interoperability challenges

Integration and interconnection of tools is a required prerequisite in order to support the collaboration within a development process as well as with customers and suppliers. Interoperability is therefore getting more and more crucial for successful and efficient product engineering. The main technical challenge in addressing this problem is the lack of open and common interoperability technologies. Nevertheless, some interoperability solutions have been proposed in the past:

Open Services Lifecycle Collaboration (OSLC)[6] is an approach based on standardised and well-known internet technologies. It respects the needs for information sharing rather than exchange and the integrated information as well as integrated access to those information principles. It is therefore a promising approach especially for interoperability challenges which can be solved by linking certain development artifacts.

Another approach is the ReqIF[7] data exchange format. This standard specifies an open, non-proprietary exchange format for requirements. This format supports the import of requirements from various authoring tools. This means that customers can have other requirements authoring tools.

XMI is another widely used interchange format for sharing models using XML. One important example is the Autosar XMI standard. AUTOSAR has intentionally been defined to improve the interoperability and exchange of software components in the automotive domain. There is also a strict definition of interoperability in AUTOSAR[8] that describes interoperability aspects for AUTOSAR. Nevertheless, although models are usually based on the same specification, there is often the issue of a multitude of dialects [Broy, 2010]. This means that the implementation of the same specification can vary between tools – thus hindering interoperability.

In a model-based development environment the use of Model-to-model transformations (e.g. ATL[9] or Query View Transformation (QVT)[10]) can be valuable in order to transfer knowledge between different modeling tools. Last but not least interoperability can also be realized by using Interfaces APIs. Usually this approach is used for point-to-point solutions, which contradict the open Crystal interoperability approach.

So far, these solutions have mainly focused on static data exchange. Especially for simulations in heterogeneous environments, data has to be exchanged dynamically during runtime. Therefore, the Functional mock-up interface[11] standardizes how different simulation programs can communicate with each other during run-time.

Especially OSLC is a very important aspect in CRYSTAL and therefore also for the public use case AUTOMOTIVE. This chapter summarizes some of the main interoperability challenges in the automotive domain. Some already show challenges which arise through the use of OSLC to solve previous challenges. The public use case will therefore also identify potential improvements for existing interoperability standards. Furthermore, it should be investigated whether or not OSLC is applicable for all challenges and what are potential alternatives. The selection of one of these techniques is dependent on the concrete application scenario. Various issues have to be considered, e.g. how much data has to be moved, how do ensure the integrity of data, how can the approach be integrated with the current process. This means that the challenges have to be identified and analyzed in-depth in order to come up with a useful solution. Often tools drive the process, but with a flexible and open interoperability specification the process drives the tool environment.

In the following sections, some important challenges of the automotive domain are described.

[6] http://open-services.net/specifications/
[7] www.omg.org/spec/ReqIF/
[8] http://www.autosar.org/download/R4.0/AUTOSAR_TR_InteroperabilityOfAutosarTools.pdf
[9] http://projects.eclipse.org/projects/modeling.mmt.atl
[10] http://www.omg.org/spec/QVT/1.0/
[11] https://www.fmi-standard.org/

3.1 Integration of OSLC to other interoperability concepts and standards

In this section, some approaches about the integration of interoperability standards and concepts are presented. In particular, a high level linked-data concept such as the upcoming OSLC standard is set in context to central data repositories (data backbone) and well established automotive standards of the ASAM consortium.

3.1.1 Integration of OSLC and the Data Backbone concept

Section 2.1 already introduced the vehicle development process with a special focus on vehicle testing processes. Various tools are used to create different kinds of artefacts throughout the testing process. Most of them are currently stored locally or in a non-transparent manner (and thus are hardly to access), which makes data-reuse and traceability complicated or even impossible.

A so-called *data backbone* for various testing phases is a generic concept to overcome these limitations. Figure 3-1 below illustrates the basic idea of this concept.

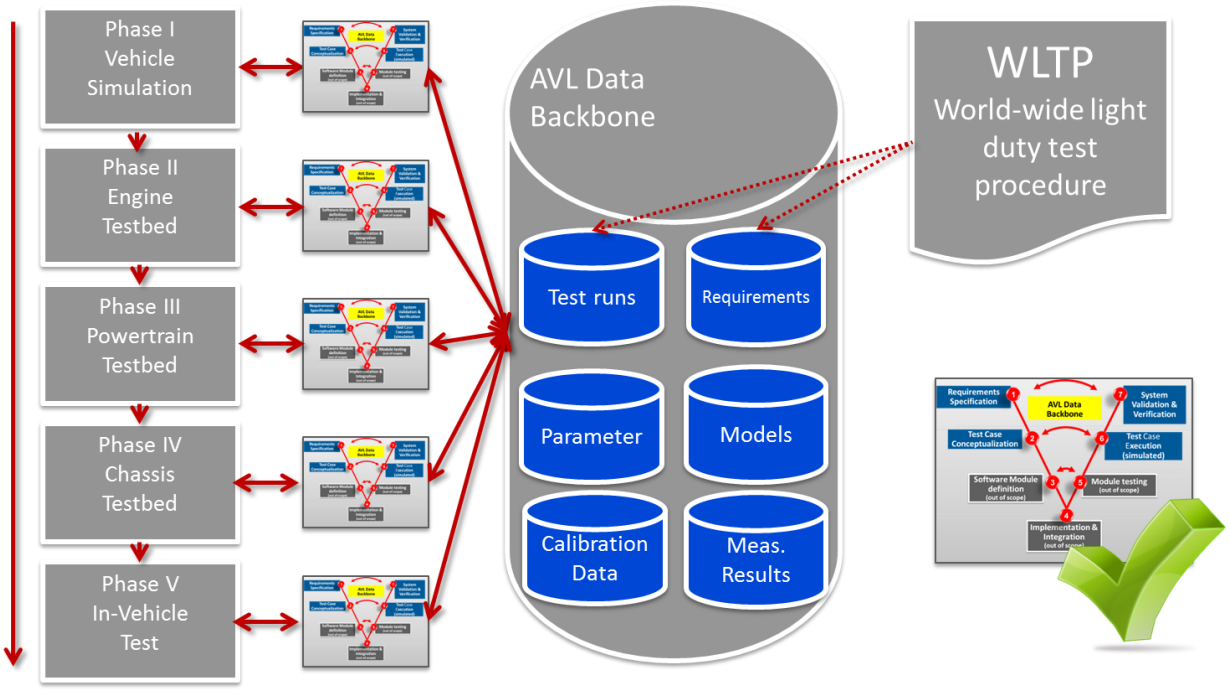

Figure 3-1: The Data Backbone concept for testing phases

The data backbone acts as a single-source-of-truth for all tools and related data categories applied in all testing phases represented by different testing V-models (and thus different tools applied in these testing V-models). With this concept, consistency among the development processes should be achieved and effective frontloading of development tasks becomes possible.

It is important to understand what *consistency* means in this context. For some situations it might be sufficient that a central data repository ensures the single-source-of-truth concept, i.e. a unique and transparent way of data access. In practise however, a single-source-of-truth concept does not necessarily ensure consistency of data content (e.g. re-using the same set of calibration data throughout two testing phases). It may happen that for various reasons (e.g. if different naming conventions are common in different

development phases) two variants of calibration data sets are created at two testing phases. A stronger meaning of consistency may force the *data content* of the several data categories to be aligned with the different testing phases. For instance, with a fully consistent data set, the parameters, calibration data, and measurement results of two test-runs in different test phases would become directly re-useable and comparable. It is part of the project to evaluate, which interpretation of consistency is sufficient to overcome the most important limitations of today's systems.

Figure 3-2 illustrates a possible data backbone concept based on the OSLC concepts. The various data categories are stored in one or even more (3rd party) data providers. The data consists of all the details created by the related authoring tool and only these authoring tools are able to fully interpret and modify this kind of data. OSLC adapters, however, abstract from these details and provide only a reduced data model per data category (also called OSLC domains). These (potentially standardized) OSLC domains are designed in a minimal manner in order to just fulfil the needs for defining data interrelations and navigation. On top of this minimal OSLC data structure a uniform workbench could navigate over this data structure, without the need of understanding all the details the authoring tools have to deal with. If a deeper data analysis or modification is needed, the workbench just delegates this task by invoking the corresponding tool with the appropriate OSLC link or requests an appropriate data artefact representation.

In addition, a customer has the freedom of choice which kind of data backbone he wants to use. For instance, a classical ALM tool such as PTC Integrity may provide important features such as variant and version management. However, this feature may not be needed by every customer. Consequently another data provider (e.g. an in-house database) is sufficient. With the use of OSLC both the authoring tools as well as the uniform workbench does not depend on which data provider is in use.

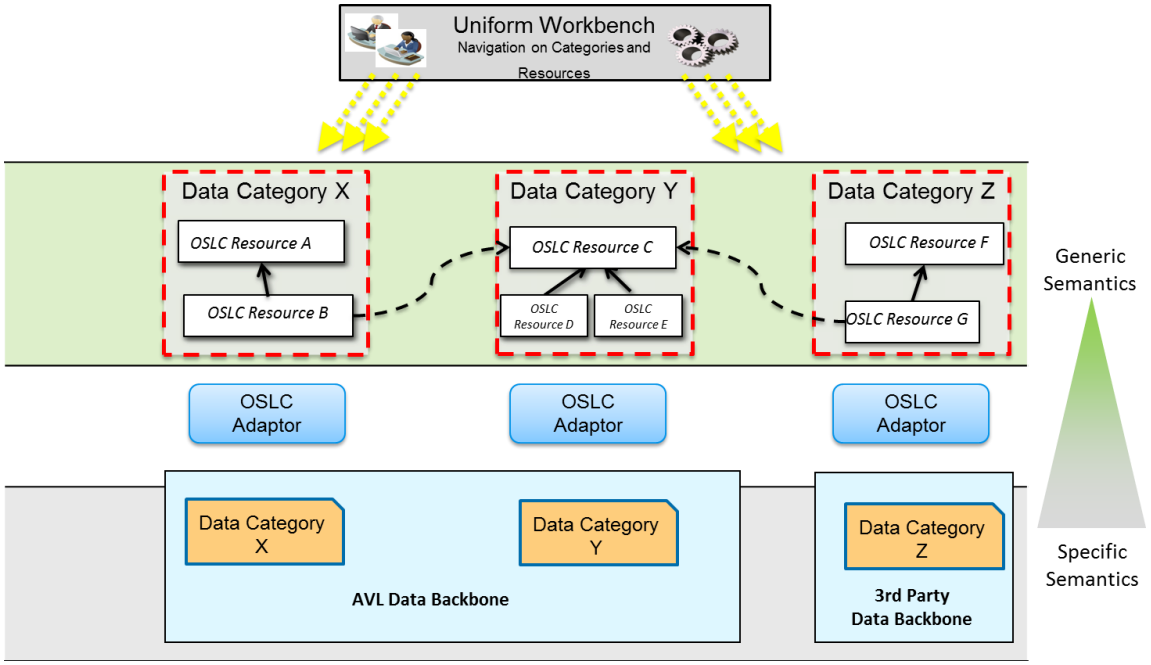

Figure 3-2: Integration of OSLC and the AVL Data Backbone concept

APPLYING THE *USE CASE SCENARIO*

In terms of OSLC, it is mandatory to develop a corresponding OSLC model that interlinks the data categories appropriately. One purpose of the CRYSTAL project is to develop and standardise such appropriate OSLC resource models by analysing the interoperability challenges of the provided use cases.

3.1.2 Towards a concept for the Integration of OSLC and Other Standards

OSLC does not cover everything what can be subsumed under the term interoperability. In addition, it is not the purpose of the CRYSTAL project to invent a new standard that is capable of such universality. Instead, existing standards will most likely have to combine with the upcoming concept of OSLC.

In Figure 3-3, such a combined approach is sketched for calibration data management: A calibration tool (e.g. AVL Cameo[12]) is storing its data in a proprietary format, but is also capable of exporting some aspects of this data to different standardized formats (such as ASAM MCD-2 MC[13] for calibration data and ASAM ODS[14] for measurement results). The exported data may be stored in a central data base such as the data backbone concept (e.g. using AVL Santorin[15], which is an ASAM ODS compliant data base). OSLC adaptors on top of all involved tools and data providers, however, allow direct access to top level elements of the data artefacts applied in the engineering method about test and calibration iteration. These top level elements are elements of a high level OSLC resource model that complies with artefacts presented by this engineering method. A uniform workbench can then be used to navigate on a concrete instance of this OSLC model. Consequently, if the related activities of a particular use case scenario are embedded in this workbench properly, the usage of this tool would ensure that links are set properly and enables corresponding data navigation and reuse in related projects.

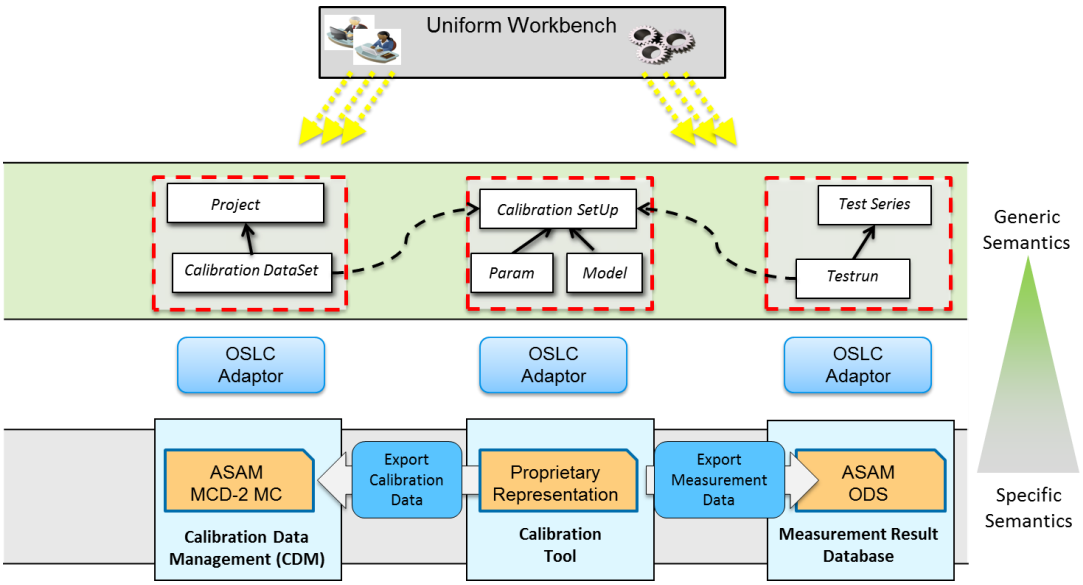

Figure 3-3: Combining the OSLC linked-data approach with calibration data management

Another example of combining OSLC with other standards is illustrated in Figure 3-4. Co-simulation is the ability to couple two or more simulation models executed in different tools at run-time. The FMI standard currently evolves itself to be *the* standard for co-simulation (so far there has been none). This kind of interoperability has by nature nothing to do with the linked-data concept of OSLC. Nevertheless, a useful combination of these interoperability types is possible: Independent of the run-time aspects and details such as how data ports of the models have to exchange data, the given fact that two models are related to each other can be defined by OSLC links as well. Corresponding OSLC adapters abstract from the particular

[12] https://www.avl.com/cameo
[13] http://www.asam.net/nc/home/standards/standard-detail.html?tx_rbwbmasamstandards_pi1[showUid]=531
[14] http://www.asam.net/nc/home/standards/standard-detail.html?tx_rbwbmasamstandards_pi1[showUid]=2027&start=
[15] https://www.avl.com/avl-santorin-asam-ods-server-standardized-data-storage

modelling tools and map the corresponding model to a OSLC resource model, which consists just of basic standard modelling elements representing a hierarchically structure and which model elements of a model A are related to what model elements of model B. Further details about the interfaces of the models such as the applied data types of concrete connected ports, parameters of the models and the programming interface to access the model at run-time via a simulation engine remain at the responsibility of the FMI co-simulation standard.

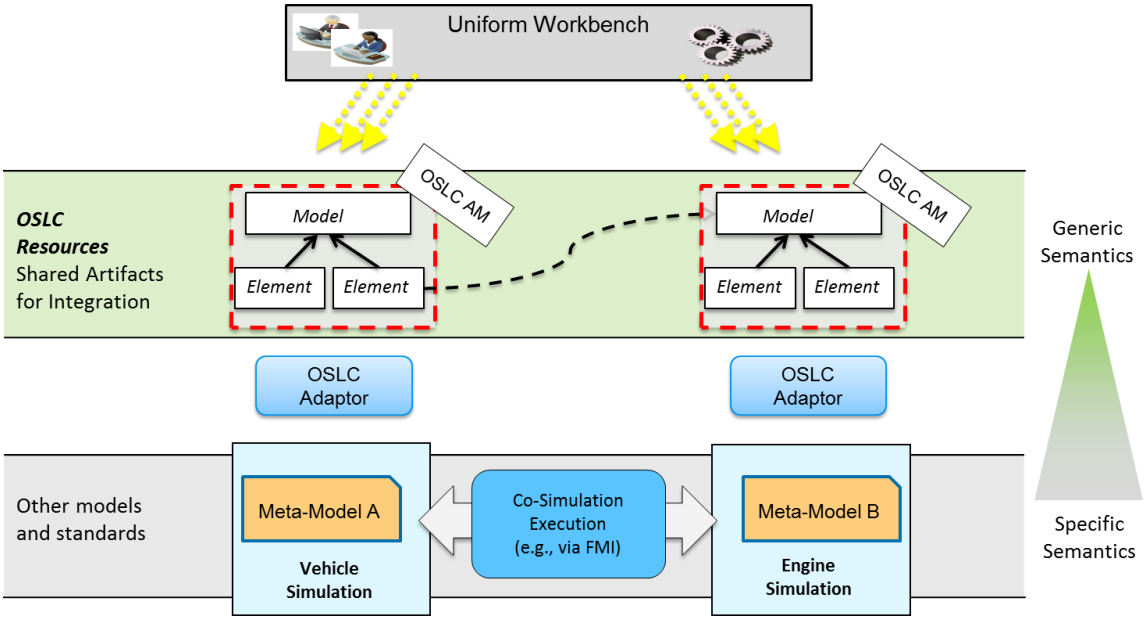

Figure 3-4: Combining the OSLC linked-data approach with co-simulation aspects

3.2 Model transformation

Interoperability between different tools can in many circumstances be accomplished via linking, where an element in one tool is linked to an element in another tool, for instance an architectural or design element linked to the requirement it satisfies. This is the basic idea of the OSLC approach to handle interoperability.

Nonetheless, in some tool interactions as for example in UC 3.1 (Volvo), a more comprehensive data exchange is desired. For instance in the timing analysis described in Section 2.2.2, the timing analysis tool requires an internal model with the functional components, topology, signals etc. This model can of course be created manually and then linked to the previously existing models, but a more efficient approach is to automatically create this model, at least partly, from relevant information in the previously existing models. For this reason, data exchange is needed.

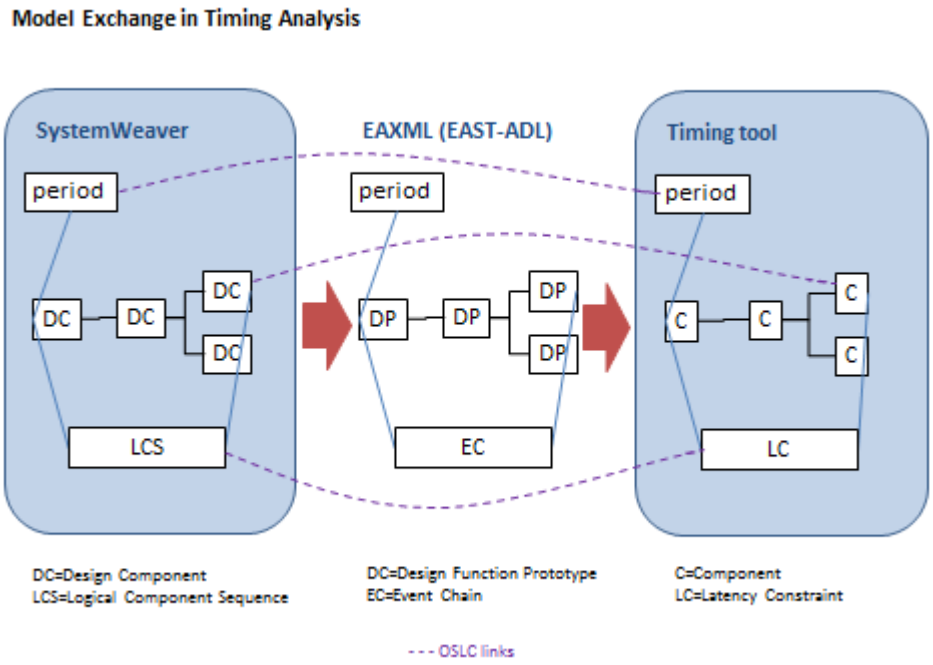

Figure 3-5: Conceivable use of EAST-ADL in model exchange

One way to accomplish the desired data exchange is to use a common meta-model and to transform the tool-internal models according to this meta-model. For the automotive domain, AUTOSAR [1] represent a de-facto standard at the software level, and EAST-ADL [2] is a corresponding standard for the upper abstraction levels. Both AUTOSAR and EAST-ADL have well-defined meta-models that can be used for this purpose.

An illustration of model transformation is shown in Figure 3-5 in the context of timing analysis. The idea is that a system model with some timing information is available in a system modeling tool (SystemWeaver). In order to make a timing analysis this data needs to be transferred to the timing analysis tool. The timing tool and the system tool may however have different internal representation of the data. Both tools may however be able to import and export data in EAXML which is an XML representation of EAST-ADL. This is for example true in UC3.1 where SystemWeaver is capable of exporting EAXML and the timing analysis tool Rubus is capable of importing it. Thus, EAST-ADL can be used as a pivot model between tools. Once the model has been transferred, additional more detailed timing information needed for the analysis can be added in the timing tool. In order not to waste these annotation efforts it is likely that the information is maintained also in the timing tool. This means that the import/export functionality also needs to create the necessary OSLC links such that the information in the two tools can be held consistent and that the annotated timing information can be visible also in the system modeling tool.

3.3 Linking between hierarchies and artifacts

The Requirements Engineering process implemented within the Microcontroller Business unit at Infineon UK identified various problem areas:

1. The quality of the requirements and how to transform from the change management system into the Requirements flow without losing data integrity.
2. The linking of all of the various artifacts (bricks), how the data is transformed/linked/moved between them.

3. The “proof of implementation”: how we prove that the requirement has been implemented and argue that it is working correctly.

Within the public use case Infineon plans to look at the second part of this, the linking of the various artifacts. Within their separate use case Infineon will look at the tooling issues related to this and their common interface protocol options as well as the other two problem areas (quality of requirements and proof of implementation).

Within the public use case Infineon will look at how to control the hierarchical issues when tracing down the data tree. The problem arises with multiple domains, tools and people so a strict control of the different hierarchies needs to be adhered to. If the hierarchy changes relating to the data, then the context or meaning of the data may also be affected.

For Example:

- An IP[16] implementation table may be necessary within the documentation to explain the numbers of reset/errors/IP’s etc required within the microprocessor.
- However when tagging within documentation the contextual meaning of the text within the table is lost, for example:
  - The Requirement : “there shall be two reset options within the X module”
  - Within the table naming the modules on the rows and the reset names on the column names.

	Module X	Module Y	Module Z
Reset y	1	1	1
Reset z	1	0	0

  - The requirement for the modules having a “reset y” or “reset z” are then identified with 0’s and 1’s as in the table above, which quite clearly does satisfy the requirement of Module X having both resets
  - The tagging[17] however cannot extract the context of the 0 or 1 so the tooling just states that “1” needs to be satisfied with no context. Within the text that is extracted by the tooling we are unable to replicate the column and row headers here so what is needed to be tested has no context – rather than asking the test developers to implement “Module X shall have 1x ResetX” we end up with the text “1”, at this point the test developer would need to go back to the intial document or to the initial requirements to clarify what is meant by the number “1” to ensure they test the correct functionality (see Figure 3-6).

[16] *An IP within the microprocessor domain is an Intellectual Property and is essentially a module. Each microprocessor is made up of a set of modules which all have a particular function, these functional pieces of IP are then linked together into subsystems and finally into a System on chip which is the complete microprocessor.*

[17] *Tagging is the term used to identify words or pictures that implement a requirement when using the Reqtify (http://www.3ds.com/products-services/catia/capabilities/catia-systems-engineering/requirements-engineering/reqtify/) tool*

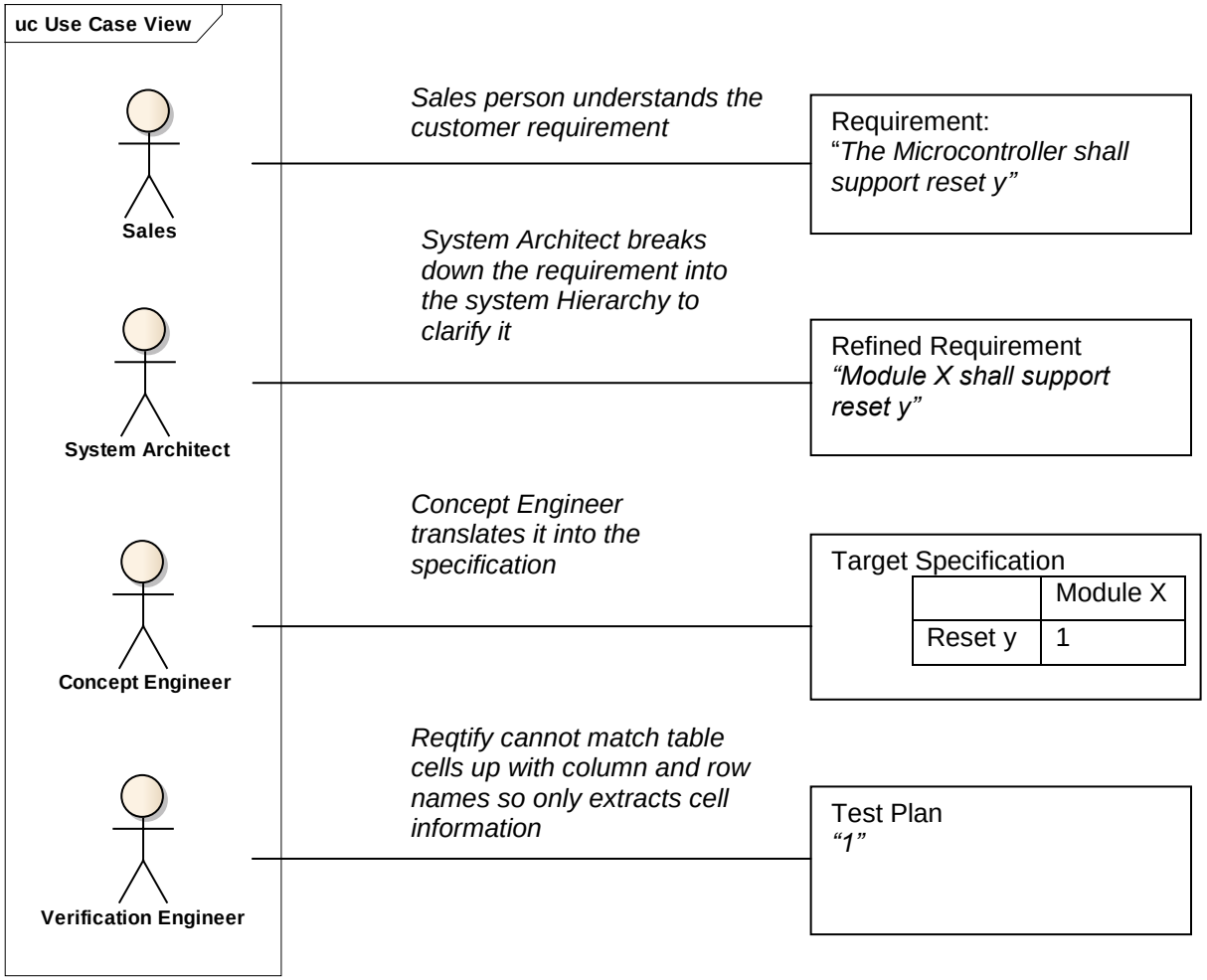

Figure 3-6: Diagramatic view of contectual loss of data through table usage

The meaning of a requirement is, as with all language, affected by its surrounding contextual setting. If the surrounding words or section change then the meaning of the requirement itself may be affected. When translating requirements through a variety of documents any manual transformation, for example from natural language into models or into a test plan or a target specification, has the ability to 'corrupt' the meaning of the requirement from its initial intention.

To clarify the meaning it may be necessary to go back to the original requirement in order to assure that the original intention is still being adhered to when it comes to being tested.

If you can ensure that either the intention is not corrupted or indeed ensure that the original intention and requirement is viewed directly at all levels then the testing and implementation of the requirement can safely be assured.

Infineon's Automotive Microcontrollers are all required to achieve the ISO26262[18] safety standard; the hierarchal structure for the safety requirements is a mandatory part of this standard (see extract below):

"6.4.3 Management of safety requirements

6.4.3.1 The set of safety requirements shall have the following properties:

a) Hierarchical structure,

NOTE Hierarchical structure means that safety requirements are structured in several successive levels as presented in Figure 2. These levels are always aligned to comply with the corresponding design phases.

[18] (http://www.iso.org/iso/catalogue_detail?csnumber=43464)

Table 2 — Methods for the verification of safety requirements

Methods		ASIL			
		A	B	C	D
1a	Verification by walk-through	++	+	o	o
1b	Verification by inspection	+	++	++	++
1c	Semi-formal verification[a]	+	+	++	++
1d	Formal verification	o	+	+	+

[a] Method 1c can be supported by executable models.

”

Infineon therefore are setting up a new relational database within the Crystal project and will have this as a single data source where all of the bricks may access to ensure they have the correct data info to produce their expected results. This database is the Knowledge and Interface Database – shown within Figure 3.3 as KID. This will become heavily involved within either an ARQE.xml (Infineon IOS) or an OSLC IOS solution to share information for variant and tool management within the flow, however it is still under investigation to analyze the what and how of the information required to be stored and protected. The amount of data currently requiring manual updates due to changes etc mean that holding the information within an excel or other format renders the Excel obsolete within a couple of days, a database holding the information, with restricted access to all via a change management system will allow us to manage this centralized data. Currently we are supporting over 6000 semi-variable link paths and a variety of other semi-variable data information for aligning our tools.

Infineon shall analyze what information they found of use, the relations required and feedback on the positives and negative of implementing this as a solution. Infineon will also be building two new tools – one within the project and one at another site so outside the scope of the project called DaD and MoM (so you could say we have a family of tools – DaD [data analyser dashboard] MoM [Measure of Metrics] and the database is called KiD [Knowledge and information Database])

Figure 3-7 below shows the area circled in red where these tools will fit within the Data flow for the Infineon product family. The areas in pink all have an independent hierarchical structure; the database works essentially as a look up table so that we can align these hierarchies.

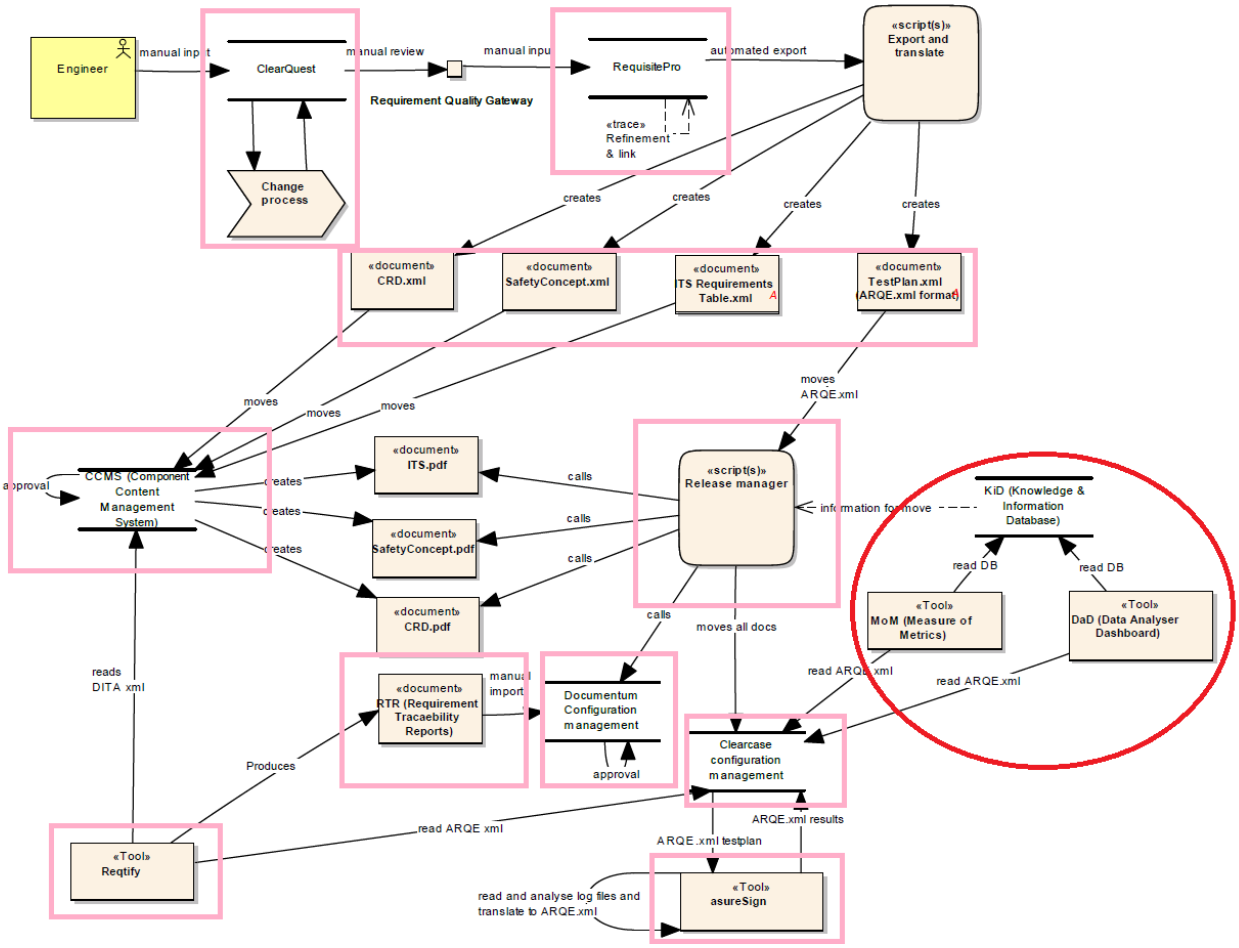

Figure 3-7: IFX Data flow diagram

The purpose and plan for the tools within the flow is as follows:

asuresign is a new external tool from TVS

This tool contains the blackbox testplan as a requirements list. Within the tool the requirements are internally linked to a set of goals required to prove the requirement and then to simulation results etc. These results are translated into a format to be fed into the requirements traceability tree. The tool has been driven by Infineon to extend into the requirements tracing to ease the cross domain issue that Infineon faces (pre-silicon/post-silcon/software domains). Currently it is being rolled out across the pre-silicon teams. The next step will be the roll out into validation and then firmware/software teams, etc.

MoM is the Measurement Over Metrics

This will be used to debug across multiple domains, pre/post silicon, software etc. This is not part of the deliverable but a user of the KiD database, asuresign, simulator logs, fault analysis and other software tools.

DaD is the Data Analysis Dashboard

This links the asuresign xmls (translated results of the log files) together to allow for visibility across the project domains. It will therefore allow a high level overview that all of the requirements were implemented and passed and in which domain this occurred, it also assists for managing person resources.

KiD is the Knowledge and Information database

This will contain info on paths for moving documents into configuration management (via the release manager), paths for the Reqtify tool to ensure the correct linkage, roles and responsibilities for IP's across the domains and the variants', the hierarchical pairing within the current mismatched flow, in future it will define the hierarchy and not just reflect it. The database will be protected via the change management tool which will be Jira in the future.

3.4 Mapping functional structure and product structure (BoM)

Systems engineering has become multi-disciplinary, with a growing importance of software and electronics.

In the past, mechanics have been the primary aspect.

Product lifecycle management (PLM) is the process of managing the entire lifecycle of a product from its conception, through design and manufacture, to service and disposal[19].This terminology has emerged in mechanical engineering. The development in software engineering has been slightly different. Here we talk of Application Lifecycle Management (ALM), as a continuous process of managing the life of a software application from inception through development and maintenance to end of life.

For a long time, software has been treated as one item in a BoM (Bill of material), but Software development can no longer be seen as a black box process. ALM must become a natural extension of PLM with seamless, process based integration.

Depending on the position in the product lifecycle, engineers might have different understandings and views of the product leading to several parallel structures. The functional structure heavily depends on the requirements, whereas the product structure often is a hierarchical relationship of parent and child nodes. Therefore, a mapping between requirements and product structure offers a special view on demands for products functionality and allows the engineer to verify the correctness of the functionality of a product.

As mentioned above, UC 3.3 is concerned with systems engineering on powertrain level. This means that mechanical aspects are at least as important as E/E aspects and respective ALM and PLM environments have to be connected accordingly.

Figure 3-5 depicts the main interoperability challenge. It shows the application of two different tools: one (PTC Integrity) for requirements and another one (PTC Windchill) which reflects the product structure in terms of the Bill of materials (BOM). In [Eigner 2005] the BoM was defined as representation of the product structure, or a specific domain dependent view of the product (e.g. design or manufacturing BoM).

This requires not only a simple interaction of tools, but a mapping of the structural aspects as well. One important aspect which requires a tight integration is change management. If hardware and product changes are managed in a PLM tool and software and system changes are managed in an ALM tool, there needs to be some kind of synchronization. The same applies for configuration management.

[19] http://en.wikipedia.org/wiki/Product_lifecycle_management

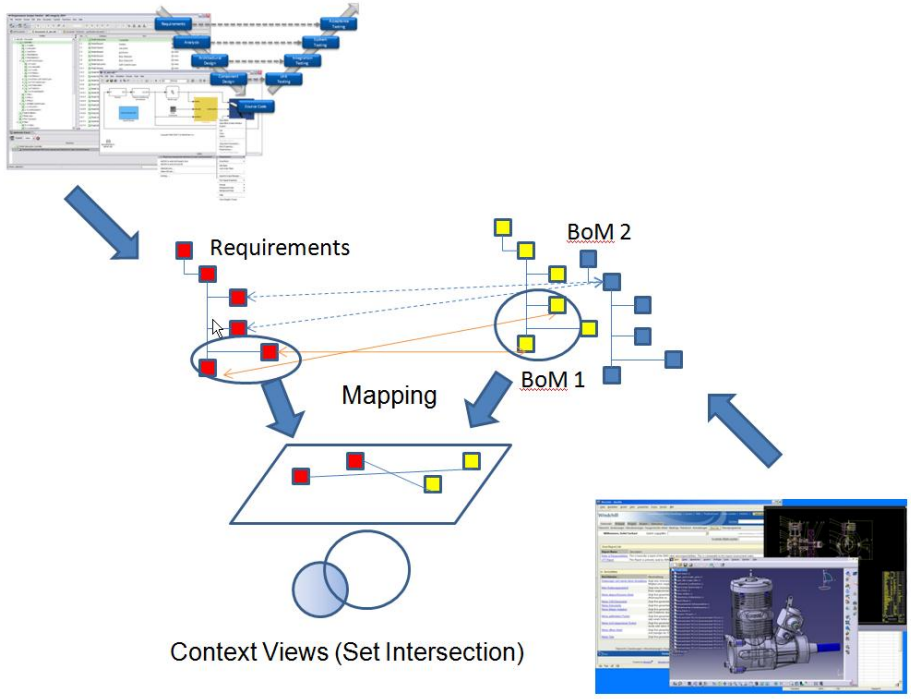

Figure 3-8: Conceptual mapping of requirements to the BoM

3.5 Interoperation between role specific authoring tools

A large company such as Daimler with many departments and persons involved in the development of high-quality E/E systems requires a number of tools to fulfill the various tasks throughout the entire development process.

This means to use tools as optimal partners of each development engineer to fulfill his or her very specific task in the most efficient and beneficial way. Considering that most E/E systems are complex and require the collaboration of many different organizational units within a company, it becomes clear that local processes arise to reach the goal of optimizing efficiency in each of these organizational units. Even though all these processes are derived from a common master process, tools vary between those units, even though they are based upon the same commercial product.

This raises the necessity of interoperability between the tools.

Since different persons are working in different environments, data has to be synchronized, exchanged, checked for consistency, and linked to each other in order to ensure an overall controllable system-scope development. In the Daimler case, variation of tools is related to the following dimensions as shown in Figure 3-9:

- Development process step
- Abstraction level
- Engineering domain
- Actors role

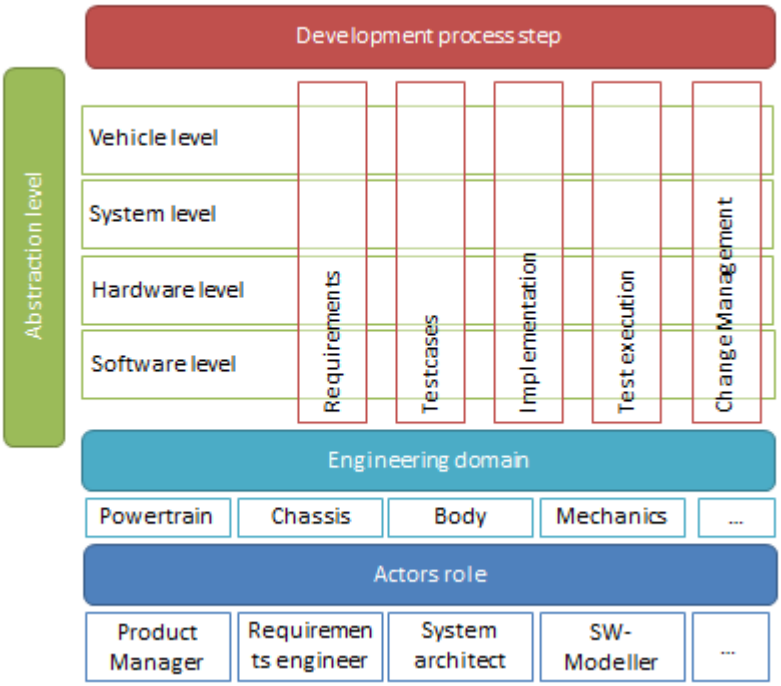

Figure 3-9: Overview of development dimensions

Each of these dimensions is widening the range of best-fit tools for local purposes. To enable collaboration among locally optimized conditions (regarding ergonomics, efficiency, interfaces), there are two major possibilities to align with:

A) Include all development data in one big tool which reflects each local optimum (centralized data management)

B) Connect all existing tools (properly identified as best pick) with each other (decentralized data handling with traceability)

In our environment, option B) becomes much more attractive than option A) because the impact is way less than changing every tool in the company.

Authoring tools being specific for each role, process step, domain, and abstraction level are facing the following challenges to offer a feasible interoperation in larger scale automotive E/E system development projects:

- Atomic objects, identifiable and traceable between different tools
- Document any kind of change on objects and containers of objects, including changes in trace information
- Exchange data with other tools, able to consider being master or slave for specific parts of this data
- Ability to navigate from and to objects in neighboring tools, keeping the level of user attention for using different tools low

Former approaches were aiming to point-to-point integrations which were hard to buy in, maintain and operate. Modern approaches favor “loose coupling” by standardized and open protocols which can be implemented by any tool vendor. The more these solutions are spreading amongst automotive companies the better is the connection effect. This is the challenge that Crystal IOS can target within and related to this project.

3.6 Requirement traceability in the software development flow

Traceability is one of the main interoperability challenges in Valeo's software development flow. In terms of IOS the main need is a tool that would help trace requirements from their definition to their measurement or verification. This traceability is also required by the ISO26262.

Valeo defines the traceability graph below (see Figure 3-10) that we intend to use for the UC.

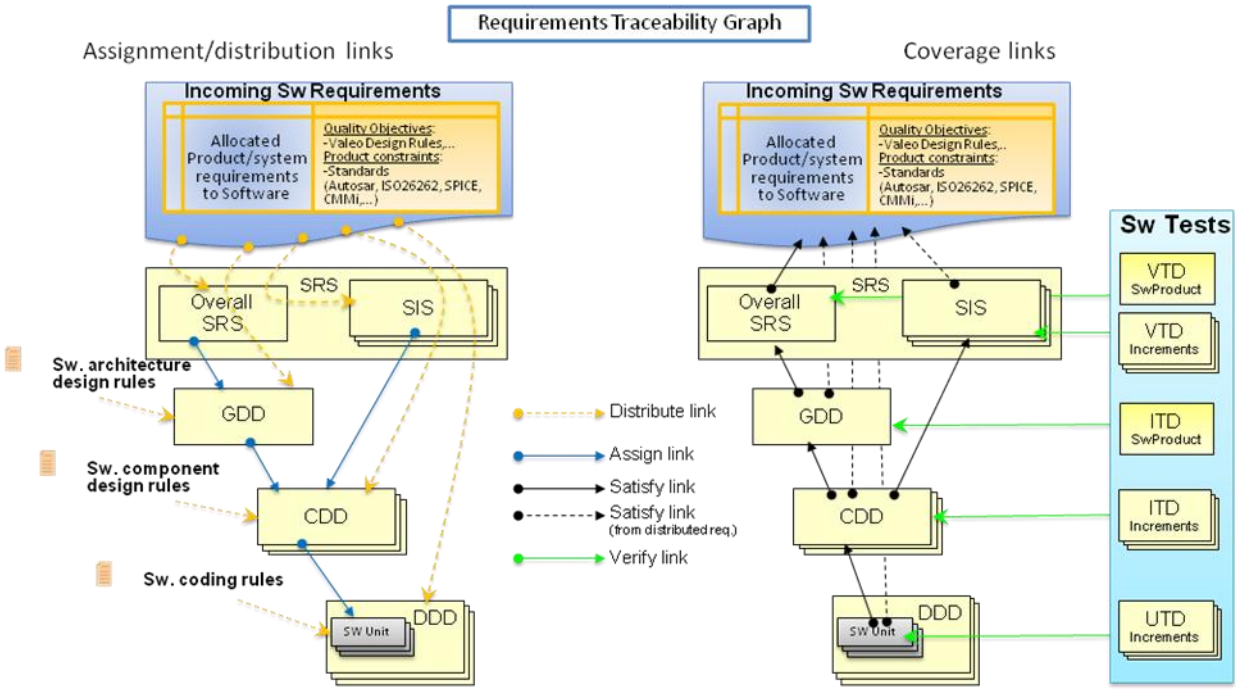

Figure 3-10: SW traceability workflow

Currently the SW traceability stops at application level and neither the BSW nor the OS are included. The challenge there will be to include them for a complete coverage.

3.7 Traceability between different workflows

The interaction between design process (red arrow) and functional safety process (green arrow) in UC 3.5 (CRF) can be again shown putting in evidence the IOS (possible/necessary) relationships, with reference to Figure 3-11: the black and gray lines distinguish the relation with respect to the current Use Case: black means covered, while gray stays for not covered but envisaged/desired for a future (out of scope for the moment) implementation; dotted pattern simply indicates the absence of a direct, but envisaged/suitable/necessary link, manually operated for the moment, but also still manually operated (probably) in the future (gray color). The presence of a line, anyway (gray, dotted or not), represents the need for a link.

The Mirroring of work items for traceability and the Transformation/Application between models for interactions are general IOS issues for the use case, where the main challenges are to identify which links are necessary and which artifacts need to be linked and at which level of detail; this last problem is a matter

of “granularity”, depending at which level we need to descend for having a complete description of all the necessary elements of the process analyzed.

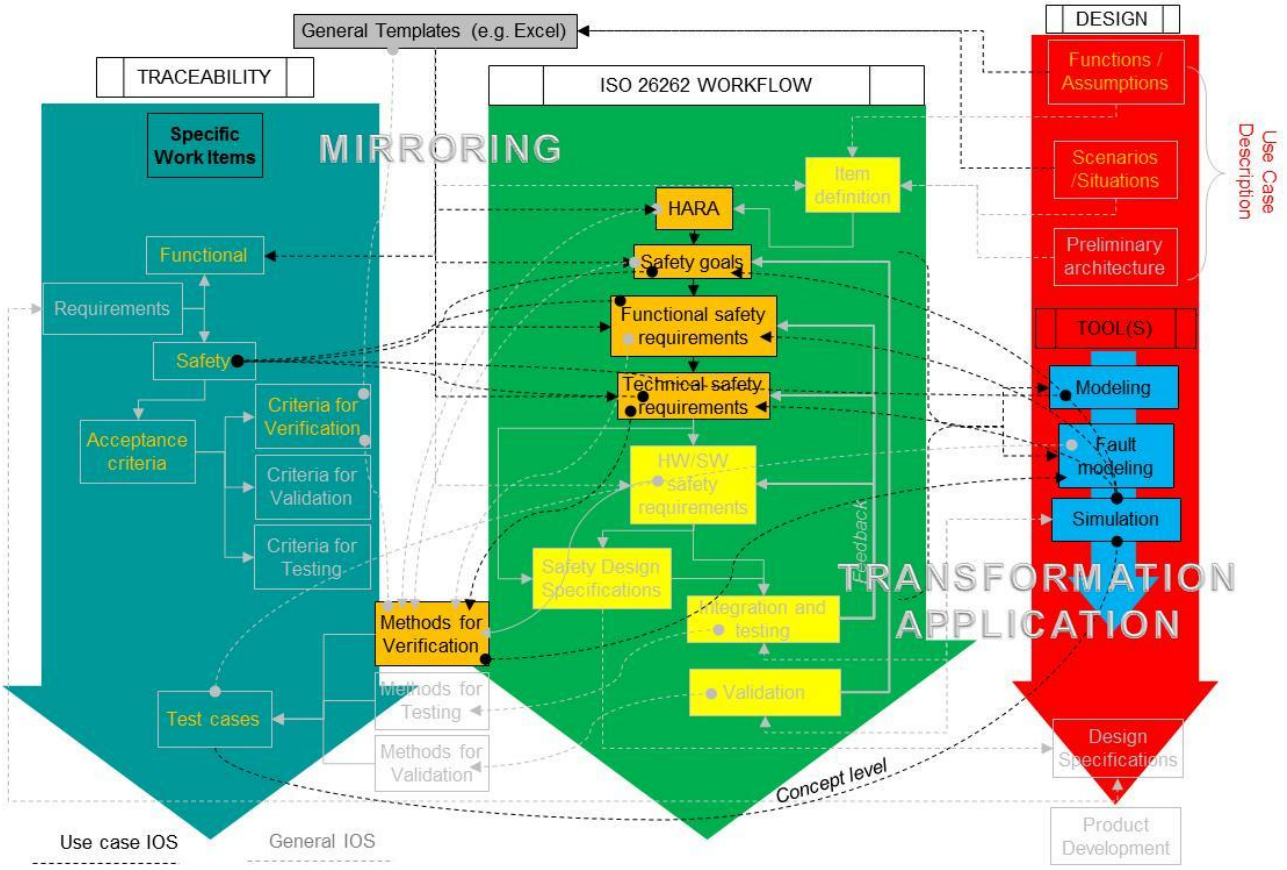

Figure 3-11: Interactions between workflows at data level from and to Functional safety-Design

In orange (block and text) use case real data are indicated in orange (block and text): functional safety project model, design inputs and traceable work items, these last “mirrored” from the project model and design inputs.

“Mirroring” is something more complex than duplication: more exactly it should be the "reflection" for traceability purposes of the "work items developed by the other frameworks (design and functional safety, this last described as in the ISO 26262 workflow). There is the possibility to provide traceability means also within the frameworks themselves, without an additional tool, according to the development stage of the related work items. This is still work in progress. The way of OSLC could help, but It is necessary to analyze/verify the possible impact versus the actual available frameworks. Some of the problems could be: to “rework” all the current work items and/or to redefine the design of the processes (but in the case of the ISO 26262 the process is firmly stated and cannot be "redesigned": it should be represented by the tool/tools). “Rework” could be to introduce links if it is possible, but if it not possible it means to restart quite from zero.

“Redesign” is something that can happen if the framework/tool does not match with the process: in this case the winner should be the process (e.g. ISO 26262: redesign not possible) and the framework unable to do this should be rejected. In general a framework/tool that supplies a sufficient flexibility in the work items definition could be suitable, but for several types of requirements a tool only could be not sufficient: e.g. one need could be the traceability, while other needs could be related to the modeling of the process itself (e.g. ISO 26262 framework); from this last case, for instance, we could have the need of a structure suitable for representing the requirements (but according to the standard anyway) and their relationships in a “safety

case" and the traceability capability alone is not sufficient for this aim. A superior stage could be the way of integrating together modeling and traceability and to introduce a certain degree of automation where possible.

In this context of work, the specific use case of a SEooC development is concentrated in the analysis and development of the functional safety concept and technical safety concept (high level) for a system not immediately targeted to a single vehicle reference, but generally conceived for a class of vehicles encompassing a large range of versions.

Within this range of application the assumptions are related to the existence of an electronic and mechanical interfacing environment on vehicle, suitable for the installation of a climate system electronically controlled with safety relevant characteristics.

SEooC is also described by ISO 26262 and the core issue about this are to define the "assumptions" and to manage them for establishing their "validity" during the integration into the major items (e.g. system or vehicle).

3.8 Integrated Tool Environment for embedded controls development

UC3.4b has defined three different user scenarios in order to improve the development process within AVL-R.

1) *Common AVL Simulation Model Data Backbone* to enable simulation and plant models exchange with integration and configuration of versatile System-of-systems (SoS) platform
2) *Integrated Tool Environment* for embedded controls development
3) Improvement of the Development via „*Efficient Variant Handling within V-Cycle*".

Due to these different user scenarios, also different interoperability challenges exist.

@ 1) In order to improve the quality of the Control system models, AVL-R is using BOOST RT[20] from another department, AVL AST (Advanced Simulation Technologies).

BOOST RT is a real-time capable, system-level, engine simulation tool dedicated for the investigation of transient operating conditions offline in desktop applications and online in HiL environments. It is the consequent next step to further integrate the well-established BOOST 1D Cycle Simulation tool and CRUISE, the vehicle dynamics, fuel consumption, and emission simulation package. BOOST RT focuses on three main application areas used in the different engine development phases:

a) **Concept Development Phase**: BOOST RT supports a fast setup and simulation of engine design variants without considering all details of pipe gas dynamics to get guidelines on engine performance.
b) **Powertrain Design Phase**: BOOST RT engine runs drive cycles fully coupled with a detailed vehicle model (built in CRUISE) resolving both engine and vehicle dynamics with adequate level of modeling depth within reasonable computational times.
c) **Component, Powertrain, Vehicle Test Phase**: BOOST RT engine runs as plant model to support the development, calibration and testing of engine control functions.

The exchange of the configured engine model is done via personal request to the responsible persons.

One *interoperability challenge* is the exchange of data from the *Data Backbone* (see Section 3.1.1) into PTC's INTEGRITY[21] used from AVL-R via providing / selecting the needed information (e.g. INTEGRITY

[20] Based on AVL PRODUCT DESCRIPTION BOOST RT
[21] http://www.mks.com/platform/our-product

needs variant and version information, etc.) and describe these for the exchange mechanisms (see Figure 3-2 and chapter before).

Additionally to data exchange, these data should be used for the design and development of a flexible and configurable System-of-Systems (SoS) platform including the necessary tools for configuration of hardware, software, application, and communication aspects. The motivation for this step is that many upcoming applications in the automotive domain require versatile and adaptable systems (or system platforms) that can be tailored to specific application area requirements and constraints. To reach this ambitious goal (*interoperability challenge),* research and development not only on a platform level but also in the area of engineering processes and tooling needs to be undertaken. In particular, ways to integrate different subsystems and applications to a common platform and to provide tailored configuration tool-chains that support this process are the main work items of this use case scenario, illustrated in Figure 3-12.

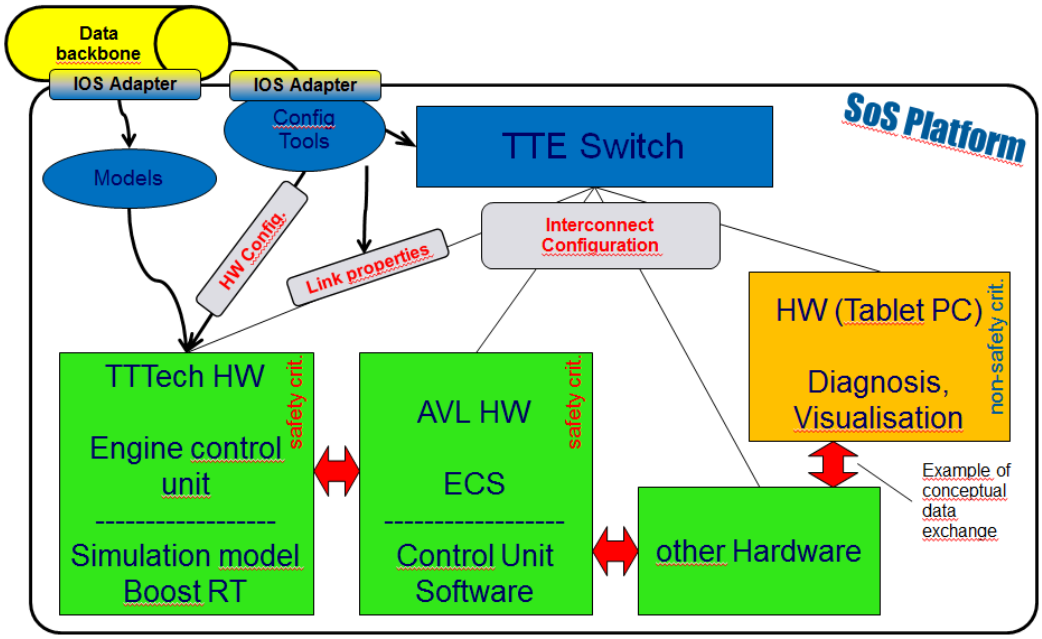

Figure 3-12: Integration of a configurable SoS Platform with Use Case requirements

@ 2) Various tools are used for the different development steps throughout the software development V-Cycle as illustrated in Figure 3-13.

AVL-R is currently using the *Integrated Tool Environment* (AVLab) as a common user interface which is a single point of contact for all related tools, i.e. AVLab bundles several tools supporting each development activity steps from modeling over testing to code generation => **seamless tool chain**

- ADD (Visu-IT!) ⇔ Simulink: Data synchronization via SyncTool
- Integrity (PTC) ⇔ Matlab: Support and Integration of PTC Integrity Source in AVLab
- Simulink (The Mathworks)⇔ AVL Concerto: allowed Concerto Plots for data visualization via AVLab
- MXAM (MES) with AVL modeling rules is started from AVLab
- MIL / SIL / back2back tests supported by AVLab
- Code Generation

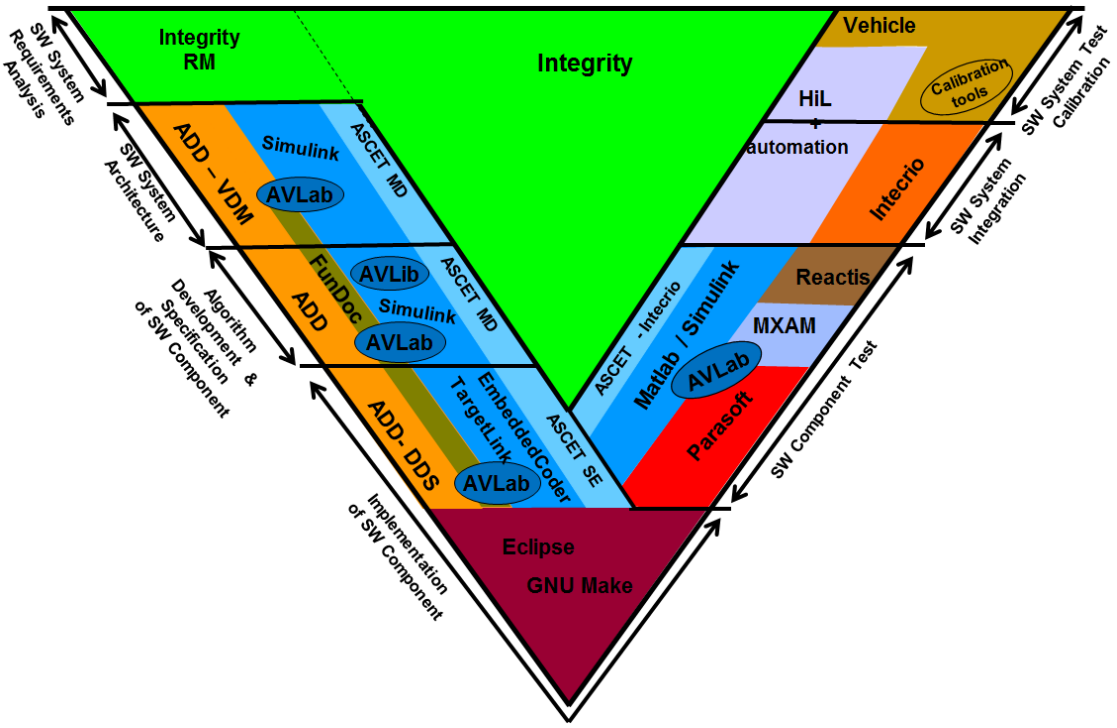

Figure 3-13: Development process including tool landscape

It also provides some kind of guidance for the developer through the development process (requirement management, architecture, model development, tests, and code generation).
Further background is the harmonization of the process & tool environment for PTE Controls.
Key points for this harmonization are:

- **Component-based development** approach is enforced.
- **Scheduling** of components in function groups (and domain) is enforced by model template in AVLab.
- **SW Architecture** is enforced by ADD as architecture tooling.
- **Code generation** for SW components is supported by tool environment in Embedded Coder and Target Link. Code generator configuration is unified.
- **Build environment** (= generation of flashable hex file) is based on generated and archived C-Code.
- Integrated **test framework** enforces common way of testing.

AVLab also shall support function development from model development over model testing to **code generation** in a Matlab/Simulink environment, with Embedded Coder or TargetLink as code generator. In addition, AVLab shall support the **methods** in engineering area:

- **A 3 Level Architecture** is the basis of component-based development
- **Naming** is ensured by the usage of the Name Checker inside ADD
- **Modeling Guidelines** are checked by the usage of MXAM (model style checker)
- **Product Documentation** is ensured by the usage of FunDoc (Visu-IT)
- **Verification/Validation** is supported via MiL, SiL, and Back-to-back testing in AVLab
- **Coding Guidelines** are supported via Code Generation Helpers (Embedded Coder Toolbar or TL Code Generator)
- **Build** is directly supporting component-based approach

The current implementation of AVLab increases the efficiency / quality of function development.

- Provide a template for modeling, with operating system to allow a simulation closer to the reality than a pure Simulink simulation.
- Simulink toolbar, with shortcuts for faster action in Simulink
- Traceability between Model, ADD Container, Integrity

The interconnection between several tools and AVLab is currently realized in a non standardized way and has to be improved. Furthermore, the “Integrated Tool Environment” provides a single point of control during all development steps including MiL/SiL Tests and should be extended to support HiL / Engine Bench tests which are currently done manually.

AVLab needs to improve its interoperability capabilities with the tool chain.

The aim (*interoperability challenge)* is to provide a standardized interface for the “Integrated Tool Environment” in order to harmonize interfaces, facilitate the substitution of tools, and to be more independent from concrete tool versions. Furthermore, seamless traceability between all artifacts should be supported. One concrete *interoperability challenge description from this use case is tackled in* Chapter 4 and there especially in Figure 4-1.

@ 3) AVL-R provides software solutions for various domains (e.g. automotive, shipping, trucks). Solutions for single domains can be very different, but within one domain there is often a huge potential for reuse.

Currently, there is no explicit and systematic variant handling at AVL-R. SW variants are stored in PTC Integrity, without mechanisms to search for or select a specific variant.
Furthermore, there is no detailed definition of variant handling and usage throughout the development process.
Variant handling cannot be considered as an *interoperability challenge* on its own, but has to be considered for the IOS specification.

4 Demonstrator prototype

As mentioned before, there will not be one coherent demonstrator for the entire automotive domain. By the end of the project we will provide several demonstrators showing best practices and solutions from the domain.

So far, there exists a first demonstrator prototype, which will evolve incrementally following the Crystal project structure. The selected application scenario is a very common interoperability challenge, which will most likely be implemented in various use cases using different tools and artifacts in different processes. This interoperability scenario describes the linkage of requirements, models, and test cases. Sometimes it is sufficient to link requirements and complete models. This means that there is no need to know or understand the inside of the model – the model can be treated as a black-box. Often this coarse-grained view is not sufficient. In this case, a corresponding authoring tool has to provide the internal structure as well or the structure has to be understood by the requirements management tool

This first prototype implements an interface between PTC Integrity, Simulink, and Artisan Studio[22], respectively. Currently, the interface is based on point-to-point solutions, but they should be using an standardized and open specification (e.g. by using OSLC) in the near future.

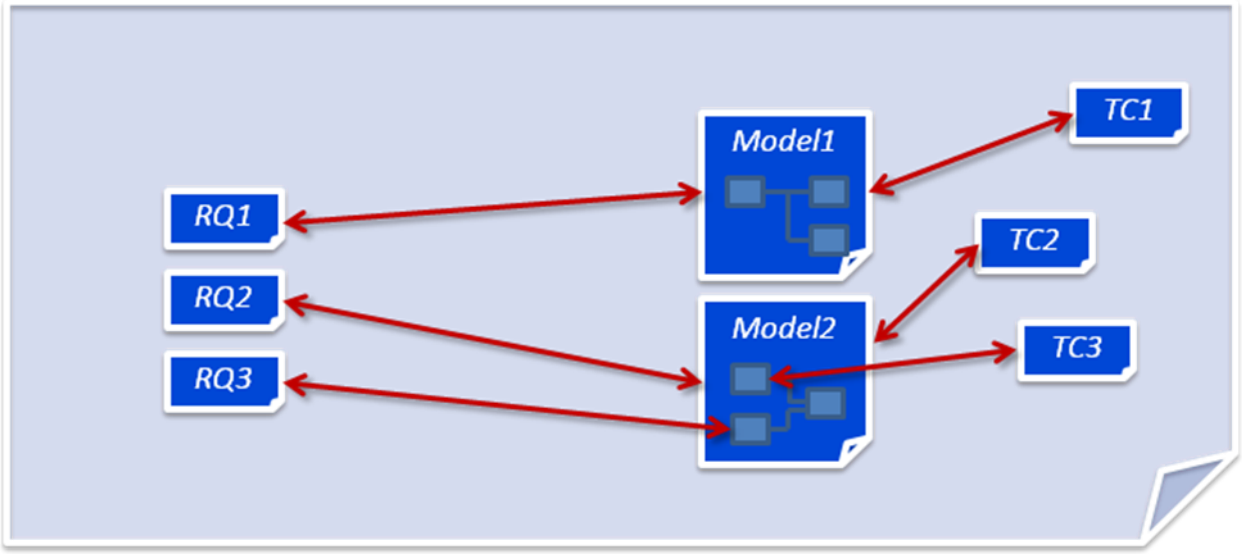

Figure 4-1: Traceability between requirements, simulation models, and test cases

OSLC is intended to be one of these open, standardized specifications for the representation, access, and linking of resources. It consists of various domains, one of which is “Architecture management”. The goal of the Architecture management working group is “to define a common set of resources, formats and RESTful services for use in modelling and ALM tools. This effort is not an attempt to define new modelling languages or storage formats, but rather enable the easy integration of existing ones through the use of simple services to link to and access them” [23].

This basic linking has been described in the OSLC – AM specification[8] as one of the essential use case scenarios as follows:

Create link from AM resource to requirement

[22] http://www.atego.com/products/artisan-studio/
[23] http://open-services.net/wiki/architecture-management/

Goal: **A designer wants to indicate that an AM resource implements a specific requirement**

1. The designer navigates to an AM resource (UML diagram).
2. The designer invokes the action to create a new link from this resource to a requirement type resource.
3. If the system is configured with more than one requirements service provider (i.e. project) associated with the context that the AM resource is in, then the user is prompted to select which requirements provider to find the requirement in.
4. The system delegates resource selection to the requirements service. The requirements service provides a resource picker to the designer.
5. The designer uses the resource picker to select a requirement to link to.
6. The designer selects the link type 'implements' from the list of available types, and confirms the link creation. The list contains the labels for each link type (not the raw URI). Systems may optionally display hover help or a short description for each.
7. The system may allow the user to add additional information about the link itself (i.e. description, priority).
8. The system creates a new link resource that has the AM resource as the subject, the requirement resource URI as the object, and the link type as the predicate. Any additional information allowed by the system is stored with the link. This link will appear in the links view for AM resources, and can be queried for.

4.1 Engineering methods in more detail

The application scenario for the demonstrator will be a model-based systems engineering approach.

Systems engineering is characterised by the tight connection of a system and a software view. Both views have a slightly different focus: Systems engineering focuses on the description of the "what", whereas software engineering focuses on the "how" [Le Sergent, 2012]. Nevertheless, any change, either in the system or software view has to be reflected in the respective counterpart. Traceability links on system and software level (requirement – model – test) as well as between these levels is therefore a required prerequisite.

Figure 4-2 illustrates this scenario and gives an overview on the engineering methods which have been identified for this basic systems engineering interoperability challenge.

Engineering method 1 "Requirement allocation" corresponds pretty much to what has been described above.

This means that this scenario has already been included in the OSLC specification.

Engineering method 2 "Test specification" links test cases to the corresponding model elements.

Engineering method 3 "Requirement decomposition" describes the refinement of system requirements to software level requirements.

Engineering method 4 "Requirments traceability" links software requirements to the respective simulation models and "Test traceability" links test (simulation) results to the concrete simulation runs.

Engineering method 5 "Test Coverage and Progress" states that information should be available to evaluate whether or not all requirements have been implemented and tested.

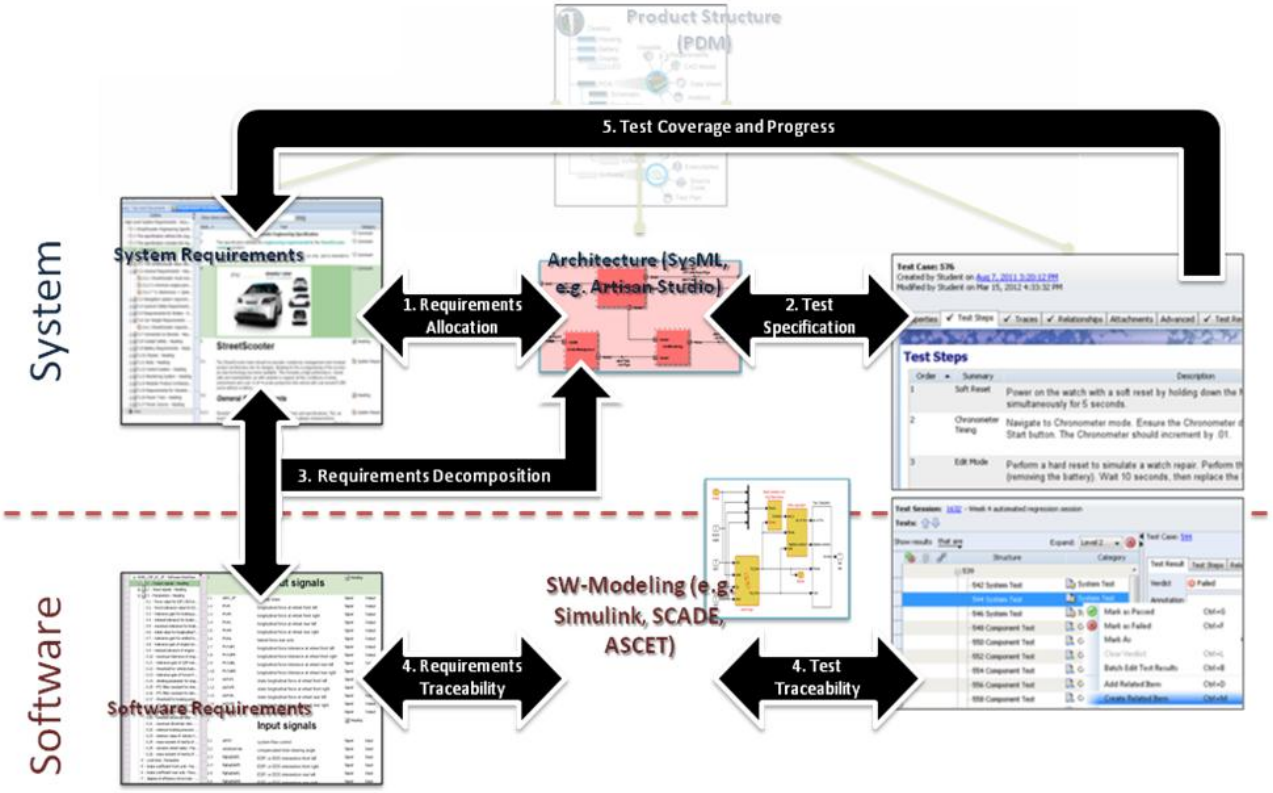

Figure 4-2: Engineering methods for the demonstrator

4.1.1 Simulation Scenario for Software

The public use case intends to use a tool interface between PTC Integrity and ArtisanStudio on system-level and an interface between PTC Integrity and Matlab Simulink on software-level. Both interfaces are currently based on proprietary APIs and should be re-based on an open interoperability specification during the CRYSTAL project.

Figure 4-3 shows a sample use case scenario for the integration of PTC Integrity and Simulink. It shows on the one hand the linkage of requirements, models, simulations, and simulation results and on the other hand the influence of a change request. This is basically a first implementation of engineering method 4 and can be understood as follows:

First, the engineer will start with the requirements document and then build the model in Simulink or update an existing Simulink model based on the specified requirements. These models (or model elements) can then be linked with the corresponding requirements. In the same way, test cases can be linked with the requirements that are validated by the test case and the corresponding model elements. This information can also be used for coverage analysis to ensure that all requirements are covered in the model and that each model element is linked to a requirement, and also to ensure full test coverage of the requirements.

Simulations and their results can be tracked in a separate item (Simulation Run), which has a reference to the underlying test cases. In case of a failed test case, a change request can be created based from the context of the simulation run.

In case of changes, the links can also be used to investigate the impact of this change by showing all related model elements and test cases. If for example a requirement will be changed, a “suspect flag” is set on the relationship and the engineers responsible for the model will be notified that the requirement and model element are not synchronized any more. The impacted model elements can be automatically highlighted in Simulink and the engineer can reflect the requirements changes in the model or – if no changes are needed

– clear the “suspect flag”. Furthermore, the change order, which tracks the updates, will also be linked to the respective artifacts.

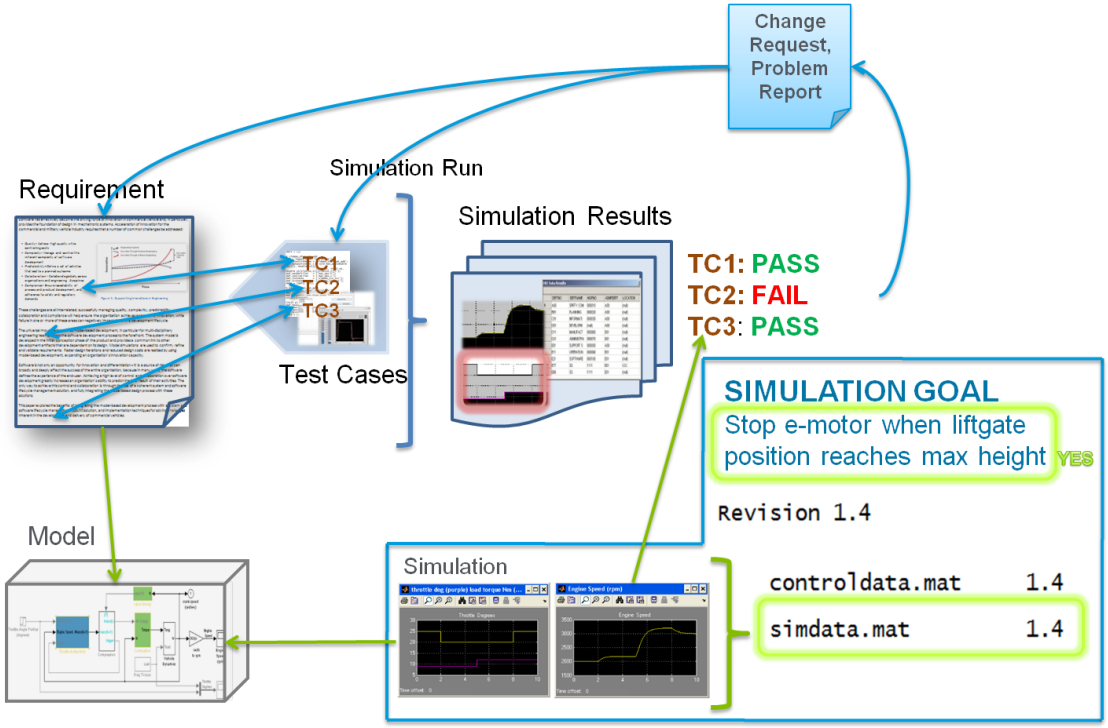

Figure 4-3: Simulation scenario for software

5 Next steps

This chapter gives a short outlook on the activities in the next phase of the project. The results of these activities will be described in the next version of this deliverable.

One of the main activities in this next phase will be the detailing of the interoperability challenges. This means that the next version of the document should include a very detailed problem description.

These descriptions will be formulated in the same way as the use case scenarios in OSLC. This is a valuable way to specify concrete application scenarios and can be used to compare the problem description with currently existing specifications.

This comparision shows if the specific interoperability challenge can simply be implemented using existing means or if it requires an extension of the current specification. For some cases it could also be possible to identify the need for a completely different interoperability specification.

The challenges should not be restricted to the linking and exchange of data. They should also include aspects such as version management, consistency, authentication, and so on.

Another result of the next project phase will also be an overview what is realizable in the course of the CRYSTAL project and what might serves as an input for follow-up projects.

6 Terms, Abbreviations and Definitions

The following terms and abbreviations are used throughout this document.

ISO	**International Organization for Standardization**
IEC	**International Electrotechnical Commission**
SEooC	**Safety Element out of Context**
OSLC	**Open Services for Lifecycle Collaboration**
BOM	**Bill of materials**
ALM	**Application lifecycle management**
PLM	**Product lifecycle management**
MBSE	**Model-based systems engineering**
ASAM	**Association for Standardization of Automation and Measuring Systems**
ASIL	**Automotive Safety Integrity Level**

Table 6-1: Terms, Abbreviations, and Definitions

7 References

[Küpper, 2011]	Klaus Küpper; Patrick Teufelberger, Raimund Ellinger; Evgeny Korsunsky; *From Vehicle Requirements to Modular Hybrid Software;* ATZ worldwide eMagazines Edition: 2011-10; http://www.atzonline.com/Article/13803/From-Vehicle-Requirements-to-Modular-Hybrid-Software.html
[ISO, 2011]	Technical Committee ISO/TC 22, Road vehicles, Subcommittee SC 3, Electrical and electronic equipment; *ISO 26262 standard: Road Vehicles – Functional Safety [Part1-10]*
[Eigner, 2005]	M. Eigner, R. Weidlich, M. Zagel. (2005) The Conceptual Product Structure as Backbone of the Early Product Development Process, Proceedings of the ProStep iViP Science
[Klassen, 2013]	Matt Klassen, PLM and ALM –strange but necessary bedfellows, EclipseCon 2013, http://www.eclipsecon.org/2013/sites/eclipsecon.org.2013/files/PLM_and_ALM_-_strange but necessary bedfellows FINAL for Upload.pdf
[Le Sergent, 2012]	Thierry Le Sergent, Alain Le Guennec, François Terrier; SCADE System, a comprehensive toolset for smooth transition from Model-Based System Engineering to certified embedded control and display software; ERTS 2012
[Broy, 2010]	Manfred Broy, Martin Feilkas, Markus Herrmannsdoerfer, Stefano Merenda, Daniel Ratiu **Seamless Model-Based Development: From Isolated Tools to Integrated Model Engineering Environments** In *Proceedings of the IEEE,* pp. 526 - 545, volume 98, number 4, IEEE, 2010