 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

D302.011

DOCUMENT INFORMATION

Project	CRYSTAL
Grant Agreement No.	ARTEMIS-2012-1-332830
Deliverable Title	Milestone Report – V1 D302.011
Deliverable No.	**Error! No text of specified style in document.**
Dissemination Level	CO
Nature	R
Document Version	V1.0
Date	2014-01-29
Contact	Rüdiger Diefenbach
Organization	DAIMLER
Phone	+49 160 8689158
E-Mail	ruediger.diefenbach@daimler.com

AUTHORS TABLE

Name	Company	E-Mail
Diefenbach, Hopp, Reuter	Daimler	{ruediger.diefenbach, daniel.hopp,christian.c.reuter} @daimler.com
Lodwich, Sauter	ITK Engineering	{aleksander.lodwich, daniel.sauter} @itk-engineering.de
Bräuchle, Kampffmeyer	PTC Parametric Technology	{cbraeuchle,hkampffmeyer} @ptc.com
Hartig, Karbe	TU Berlin	{tu_berlin.hartig,tu_berlin.karbe}@daimler.com
Bogomolov, Dietsch	Uni Freiburg	{dietsch,bogom} @informatik.uni-freiburg.de

REVIEW TABLE

Version	Date	Reviewer
Internal Review	2014-01-14	Hopp (DAI)
External Review	2014-01-22	Melzi (CRF)
External Review	2014-01-22	Stieglbauer (AVL)

CHANGE HISTORY

Version	Date	Reason for Change	Pages Affected
0.1	2014-01-09	Initial creation	all
0.2	2014-01-16	Changes after internal review	all
0.3	2014-01-24	Consolidation after external reviews	all
1.0	2014-01-28	Final version after external review	all

CONTENT

1 INTRODUCTION 7
1.1 ROLE OF DELIVERABLE 7
1.2 RELATIONSHIP TO OTHER CRYSTAL DOCUMENTS 7
1.3 STRUCTURE OF THIS DOCUMENT 7
2 USE CASE PROCESS DESCRIPTION 8
2.1 USE CASE 8
2.2 PROCESS 11
2.2.1 Analysis of Existing Process Activities 11
2.2.2 Definition of Target Process Activities 11
3 DETAILED DESCRIPTION OF THE USE CASE PROCESS 17
3.1 ACTIVITIES 17
3.1.1 Kick-off 17
3.1.2 Engineering Methods 17
3.1.3 Analysis of ISO 26262 Compliance 17
3.1.4 ADSE Development Process Definition 18
3.1.5 Tool Introduction into PTC Integrity 18
3.1.6 Setup of Project Management Tool "Backlog", based on PTC Integrity 18
3.1.7 OSLC Workshop 18
3.1.8 Tool Alternatives 19
3.1.9 Contributions to Public Use Case 19
3.1.10 Deliverable Reviewing 19
3.1.11 Dissemination Activities 19
3.2 INTEROPERABILITY CHALLENGES – IOS CONTRIBUTIONS 20
3.2.1 Engineering Methods 20
3.2.2 IOS-Relevant Tool Interactions and Tool Interfaces 20
3.2.3 Possible Requirements towards IOS 23
3.2.4 Possible Requirements for OSLC 25
3.2.5 Logical Architecture of IOS-based Projects 25
3.3 ROADMAP 28
4 CONCEPTUAL WORK 30
4.1 ENGINEERING METHODS 30
4.1.1 List of Engineering Methods Sorted by Topic 30
4.1.2 Engineering Methods in the V-Model 31
4.1.3 Detail View of an Engineering Method 32
4.2 CONCEPT FOR MAPPINGS 33
4.2.1 Approach 34
4.2.2 Example of Tracing Framework 36
5 TERMS, ABBREVIATIONS AND DEFINITIONS 37
6 ANNEX 39

6.1 ANNEX I: ENGINEERING METHODS 39

Content of Figures

Figure 1 - Testing Ground – Assistance System 8
Figure 2 - Testing Ground – Misuse 9
Figure 3 - Model of Testing Ground with Example of a Trajectory 9
Figure 4 - Equipment inside the Car 10
Figure 5 - Control Station (Tower) 10
Figure 6 - Target Process Activities: Tasks and their Work Products 15
Figure 7 - Activity Workflow of Target Process represented as UML Activity Diagram 16
Figure 8 - (Rough) Development Process based on the V-Model 21
Figure 9 - Interoperability Technologies simplify Interoperability between Users 25
Figure 10 - Project Roadmap 28
Figure 11 - Engineering Methods assigned to different Steps in the V-Model 32
Figure 12 - Detailed View for the Engineering Method “Create New Requirement Entry” 33

Content of Tables

Table 1 - Process Activities of the Target Process with their In- and Outputs 14
Table 2 - Mapping Process/Tool 22
Table 3 - Identification and Classification Templates 35

1 Introduction

1.1 Role of deliverable

This document concludes activities and results within WP 3.2. It is meant for transportation of WP results back to SP 3 (automotive subproject) and the overall CRYSTAL project.

1.2 Relationship to other CRYSTAL Documents

This document is related to the corresponding deliverables D307.011 in WP 3.7 (automotive public use case) and D308.011 in WP 3.8 (automotive ontology).

1.3 Structure of this document

This deliverable is structured as follows:

- General information about this use case (chapter 2)
- Development of the process structure and its integration into the tool environment (chapter 3)
- Engineering methods and Concept for Mappings (chapter 4)
- Additional Information (Terms, Abbreviations, Annexes in chapter 5 & 6)

2 Use Case Process Description

2.1 Use Case

As part of the automotive domain (SP3) the Daimler AG provides the use case 3.2 as an automotive OEM. The role of Daimler is “user of development tool environment”. Besides Daimler other partners with a different background are involved in this use case like universities (TU Berlin, University of Freiburg), an engineering consultant (ITK Engineering AG) and a tool vendor (Parametric Technology GmbH).

The targets of this SP3 use case are to define SP6 requirements on user level derived of a “Daimler development project” and the evaluation of the SP6 results implemented in a prototype application in the project context.

This “Daimler development project” is about the development of a host computer (next generation) for automated driving. The host computer is embedded in a big system called ADSE (Autonomous Driving in Specific Environments) and controls the vehicle to support testing activities of driver assistant systems with aspects of reproducible driving of trajectories, operation together with a partner car and operation with or without a driver in the car. The host computer is not part of a series vehicle.

Figure 1 - Testing Ground – Assistance System

Figure 2 - Testing Ground – Misuse

Figure 3 - Model of Testing Ground with Example of a Trajectory

Figure 4 - Equipment inside the Car

Figure 5 - Control Station (Tower)

Figures 1-5 illustrate a typical use case for ADSE, testing driver assistance functions on a proving ground for autonomous driving.
This video gives a short impression of the latest ADSE generation.

Link: http://www.youtube.com/watch?v=PPVrqyy6nXk

The current applied procedures and their integration into the tool environment should be improved during the project. Different tools and manual activities occur in this development process. Thus, the target is to raise the interoperability for a continuous, traceable and efficient development. The solutions have to be applicable so that the mentioned development project will be able to apply the processes, methods and tools after finalisation of the CRYSTAL project, too. Furthermore, for other Daimler projects, it should be possible to easily adapt the solutions for an efficient development process.

2.2 Process

As the host computer does not have to be newly developed, but extended and refined, the existing process had to be analysed and improved in a way such that later adjustments and modifications can be made easily. According to the “Interoperability Needs Capturing Process” the examined process can be described as a use case scenario as follows:

> *Given an existing system architecture and implementation of the host computer, a new request from a customer, i.e. the ADSE operator, demands an extension or change of the existing system. This means, that new functionality needs to be provided while using the same technology. This process includes the application of several tools and the creation or modification of several development artefacts along the way.*

This use case scenario, namely *“new customer request”,* constitutes the process that represents the foundation for the subsequent detailed analysis.

2.2.1 Analysis of Existing Process Activities

The current practice of the underlying process has been analysed to gain valuable insights with the aim of achieving a sustainable enhancement. The development is based on the V-model with additional project and product layers. The rough breakdown of the development process is depicted in Figure 8 (chapter Engineering Methods). The current practice is conducted basically according to this V- Model.
During the process analysis different tasks performed in the existing approach have been written down and analysed in order to complete the overall process. These tasks are process activities that can be assigned to the particular items/steps in the afterimage. The analysis of the overall process showed potential for improvement in order to efficiently facilitate the advantages that extensive tool interoperability can offer. In this document, we decided to focus on the enhanced target process. Therefore, the obsolete development activities are not shown in this subsection.

2.2.2 Definition of Target Process Activities

As mentioned above, we defined a set of target process activities for the examined use case scenario *“new customer request”*. Process activities can be seen as tasks with optional or

mandatory input and output artefacts. A list of process activities with their in- and outputs are listed in Table 1. Each of these activities can be assigned to the consecutively numbered steps in Figure 9. The list does not provide a sequential order yet. However, even though no timed sequence is stipulated, it implies a suggested sequence due to dependencies between evolving artefacts, e.g. *Plan Release (11)* can only be performed after the *Effort and Dependencies* have been estimated in the corresponding activity *(10)*.
Note that the subsequent list does not provide a complete catalogue of all process activities but a subset with the most significant process activities, mostly related to the steps 1-5 and to the additional layers Q0-Q4 in Figure 8

	Process Activities	**Input**	**Output**
1	Collect Requirement	Delta Customer Request	Informal Requirement
2	Formalise Requirement	Informal Requirement Glossary	Formal Requirement Glossary
3	Assign Abstraction Level To Requirement	Formal Requirement	Formal Requirement
4	Derive Sub-Requirement	Formal Requirement	Requirements Tree/Net
5	Define Top-Level Concept of Architecture	Frame Conditions (Customer Needs, Technologies, Standards, Budget, …)	Top Level Concept (Architecture)
6	Check Consistency	Requirements Tree/Net	Requirements Tree/Net
7	Document Acceptance Criteria	Informal Requirement Formal Requirement	Acceptance Criteria Document
8	Assign Variation Points To Requirements	Formal Requirement System Components Feature List Feature Tree	Formal Requirement
9	Extend Feature Tree	Feature Tree Formal Requirement	Feature Tree
10	Estimate Effort	Requirements Tree/Net	Effort and Dependencies
11	Plan Release	Effort and Dependencies Resources	Release Plan
12	Controlling	Release Plan	Reports on Issue and Change Level
13	Define System Level Concept	Requirements Tree/Net System Level Concept	System Level Concept
14	Perform HARA	Top Level Concept (Architecture)	Top Level Safety Specification

15	Perform FMEA	System Level Concept	FMEA Report
16	Perform FTA	System Level Concept	FTA Report
17	Compose Delta Requirement Specification	System Level Concept Requirements Tree/Net Release Plan Test Specification	Delta Requirement Specification (on single requirements)
18	Compose Requirement Specification	Top Level Concept (Architecture) Delta Requirement Specification (on single requirements) Test Specification	Requirement Specification (Total)
19	Create Test Specification	Requirement Specification (Total) Delta Requirement Specification (on single requirements)	Test Specification
20	Implement/Model and Generate Code	System Level Concept	Implementation Artefacts (Code, Model)
21	Impact Analysis	Requirements Tree/Net Feature Tree System Level Concept Test Specification	Impact
22	Comment Implementation Artefact	Implementation Artefacts (Code, Model)	Implementation Artefacts (Code, Model)
23	Write Documentation/User Manual	Implementation Artefacts (Code, Model) System Level Concept	Documentation/User Manual
24	Create Project Configuration	Feature Tree	Configuration
25	Implement Test	Test Specification	Test Specification Test Implementation
26	Execute Basic Test	Implementation Artefacts (Code, Model) Test Specification	Test Execution Result
27	Execute Simulation Test (Software in the Loop - SiL)	Implementation Artefacts (Code, Model) Test Specification Test Implementation	Test Execution Result

28	Execute HiL (Hardware in the Loop)	Implementation Artefacts (Code, Model) Test Specification Test Implementation Test Environment	Test Execution Result

Table 1 - Process Activities of the Target Process with their In- and Outputs

Figure 6 is a graphical representation of the process activities from Table 1 with their work products (i.e. the in- and output necessary for each task).

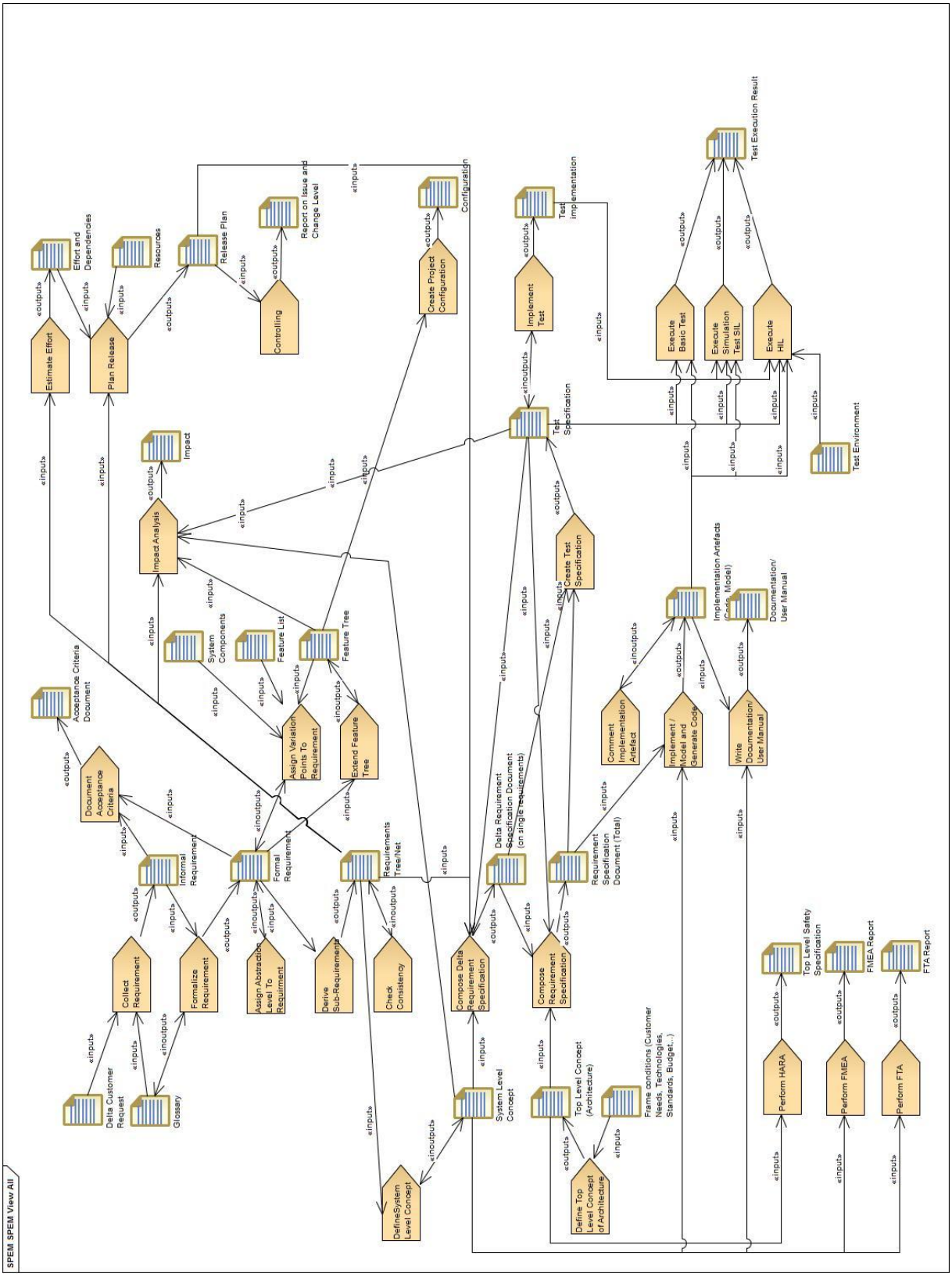

Figure 6 - Target Process Activities: Tasks and their Work Products

Figure 7 depicts an UML activity diagram representing the target process activity workflow. It illustrates the sequential order in which the different tasks have to be performed.

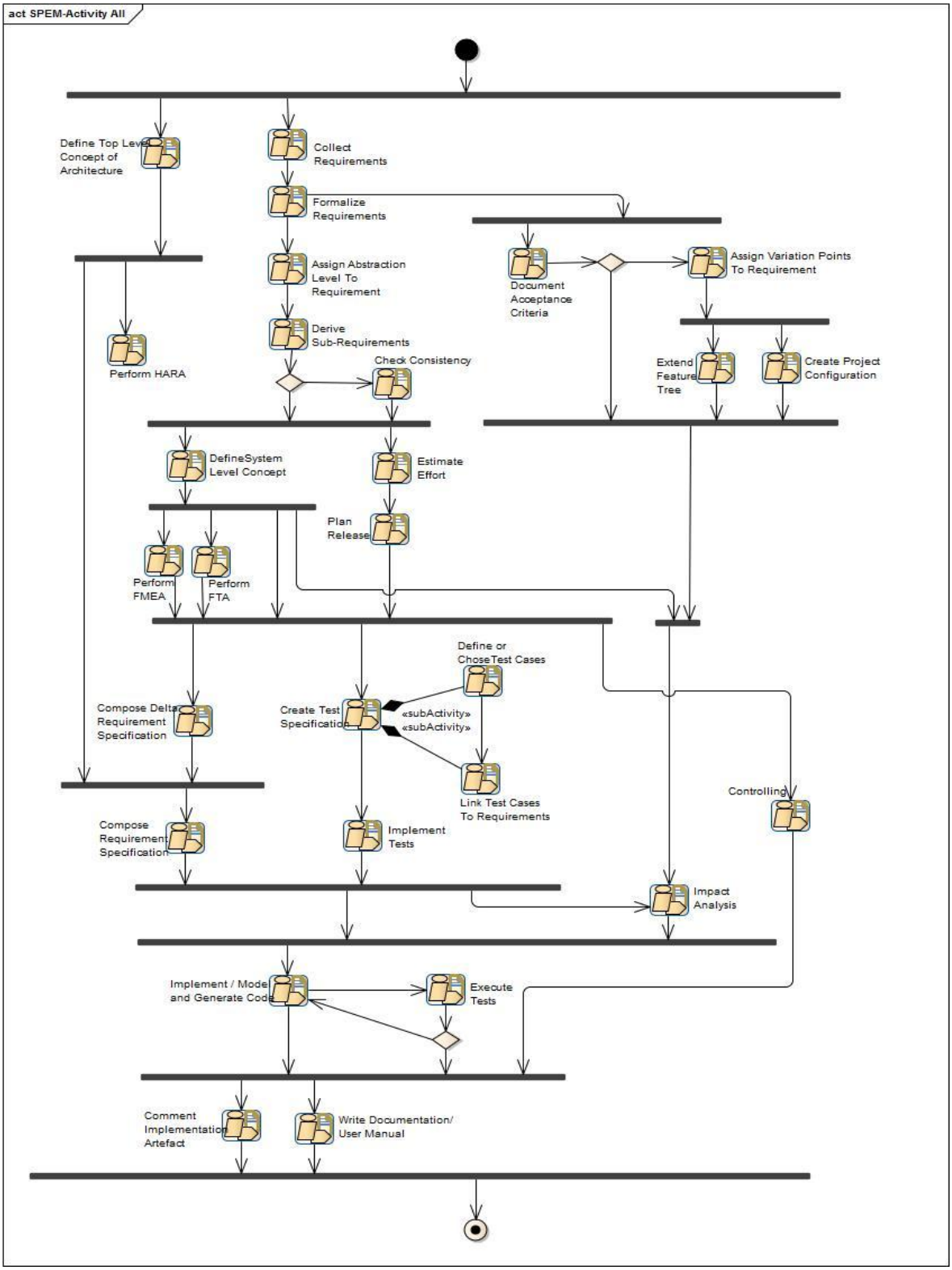

Figure 7 - Activity Workflow of Target Process represented as UML Activity Diagram

3 Detailed Description of the Use Case Process

3.1 Activities

This section briefly summarizes the activities performed in WP 3.2 during the first reporting period.

3.1.1 Kick-off

The work package has effectively started with a kick-off meeting on August 27th. The kick-off was face to face – almost all follow-up meetings were using teleconferencing. During the kick-off meeting the ADSE project was presented in great detail and the ADSE team was presented.
Each participating party has presented the type of its resources and possible contributions to the use case. From this information a scrum-like procedure has been agreed upon in order to warrant progress.
For the use case an internal milestones plan was proposed in order to monitor use case duties and progress.

3.1.2 Engineering Methods

All WP-partners have contributed engineering method descriptions for WP 6.1. During this process some problems with this approach revealed. Main point of critique was the weak grounding of those descriptions. The reasons why the engineering methods have suffered from the shortcoming were:

1. At this particular date the exact status-quo of ADSE processes was unknown
2. The engineering methods intended to describe improved CRYSTAL-like processes

WP 3.2 has communicated with WP 6.1 that more engineering method descriptions will follow in 2014. Therefore WP 3.2 has proposed to WP 6.1 to use a central requirements repository after WP 6.1 has demonstrated derived requirements from the supplied engineering method descriptions.

3.1.3 Analysis of ISO 26262 Compliance

A special meeting was held (which was in physical presence) in order to establish the commonalities and differences between the development procedures used for ADSE and the coordinated systems on the vehicle which definitively are under the government of ISO 26262. In this meeting Enterprise Architect and Simulink were identified for initial integration in order to support effortless monitoring and updating of technical safety concepts.

3.1.4 ADSE Development Process Definition

A number of meetings were held in order to establish the nature of the future ADSE development process which should be supported by the devised tool chain. After a single large meeting between all partners, DAI, ALU-FR and TUB elaborated on further features of the process.

3.1.5 Tool Introduction into PTC Integrity

An introduction into the ALM (Application Lifecycle Management) tool *PTC Integrity* was given to all partners. The goal was to educate all partners working in the use case about the functionality that the tool offers to enable future usage of the tool in the best way.

The following topics were covered in the demonstration:

- Integrity basics
- Overview of the single modules
- Requirements Management
- Change Management
- Software Configuration Management
- Test Management
- Integrity's architecture
- Reporting functionality

3.1.6 Setup of Project Management Tool "Backlog", based on PTC Integrity

It was decided that the work done in use case 3.2 will follow the agile paradigm – a process similar to SCRUM. However, a full SCRUM concept was considered inadequate. A basic concept of user stories and sprints was included. Each sprint was accompanied by a planning and review meeting. During the reviews work results were discussed and new potential tasks discovered ("user stories"). In the planning sessions each partner has committed himself to progress in a user story of his choice. During the reporting period four such sprints were executed.
In order to promote synergies between the ADSE project and use case management it was decided to use PTC Integrity to carry out SCRUM. Thereby, WP 3.2 expected to better learn about the capabilities of Integrity. Integrity was set up on a VM and made accessible via a web interface in order to enable the planning, authoring and review of user stories and the assignment of user stories to individual sprints. Before Integrity an Excel-based planning was performed.

3.1.7 OSLC Workshop

OSLC was a comparatively new technology not everybody on the project was familiar with. In order to improve the understanding of possible chances and restrictions of OSLC a small workshop has been held by one project partner. From this workshop a better common understanding of the architectural challenges was achieved. The following topics were covered during the workshop:

- What are the motives behind OSLC?
- Which technologies make up OSLC?
- What is the best classification of OSLC technology?
- What kind of interoperability is OSLC trying to achieve?
- What is the potential role of ontologies in this respect?
- What other interoperability technologies exist and how do they compare with OSLC?
- What kind of technological readiness can OSLC promote?
- What kind of problems must be expected when designing OSLC-based interoperability?
- What is the procedure for improving OSLC specifications?

3.1.8 Tool Alternatives

A list of relevant tools was compiled for the case that additional integrations could be accomplished during the project.

3.1.9 Contributions to Public Use Case

There were two meetings for the public use case from the automotive domain (UC 3.7) where example problems and possible demonstrators were compiled from private use cases.

3.1.10 Deliverable Reviewing

The use case has defined a production process for D302.011 in parallel to the sprints. Meetings on a weekly basis were held between December 2013 and January 2014 in order to assign content to partners and for discussion of the details.

3.1.11 Dissemination Activities

Dissemination activities comprise all project-related publications. On the one hand, they include general disseminations such as presentations or posters about the CRYSTAL project itself. On the other hand, they also include the dissemination of achievements such as conference or journal papers, dissertations as well as workshops, seminars or similar activities performed in order to distribute gained knowledge.

No dissemination activities were performed within this working package until the release of this deliverable. However, future plans for dissemination activities exist. They mainly include the publication of the results of different dissertation projects on related topics such as variability management, functional safety, traceability and interoperability.
The estimated outcomes are doctoral theses on those topics.
Additionally, intermediate results are planned to be published in order to share gained knowledge with the scientific world as well as receiving feedback in order to evaluate the outcome.

3.2 Interoperability Challenges – IOS Contributions

In the Daimler use case context of developing a host computer we examined several challenges connected to tool interoperability that we want to specify in the following.

1. Transfer the huge amount of „**implicit traces**“ (knowledge?) of the engineering experts into „**explicit traces**“ (methods) in a tool based development system
2. Support **efficient impact analysis** based on a given change request on requirement, system or component level to another level or to the right side of the V-Model
3. Introduction of **variant handling** to manage the upcoming use cases of the host computer and to optimise the verification and validation activities
4. Enable the **consistency of different models** on system, functional and component level
5. Almost **automatic safety case documentation** during the development phase:

3.2.1 Engineering Methods

The following engineering methods have been provided to WP6:

- EngineeringMethods-ExternalLinks
- EngineeringMethods-InsertLinkBetweenChangesetAndIssueTrackerEntry
- EngineeringMethods-VerifyModifiedSafetyFeatureInSimulation
- Create New Requirement Entry
- Manage Variability

This list has been extended during the first project phase, see chapter “Engineering Methods”
The filled-out templates can be found in Annex I.

3.2.2 IOS-Relevant Tool Interactions and Tool Interfaces

Figure 8 shows the activities of the V-Model relevant for this WP.

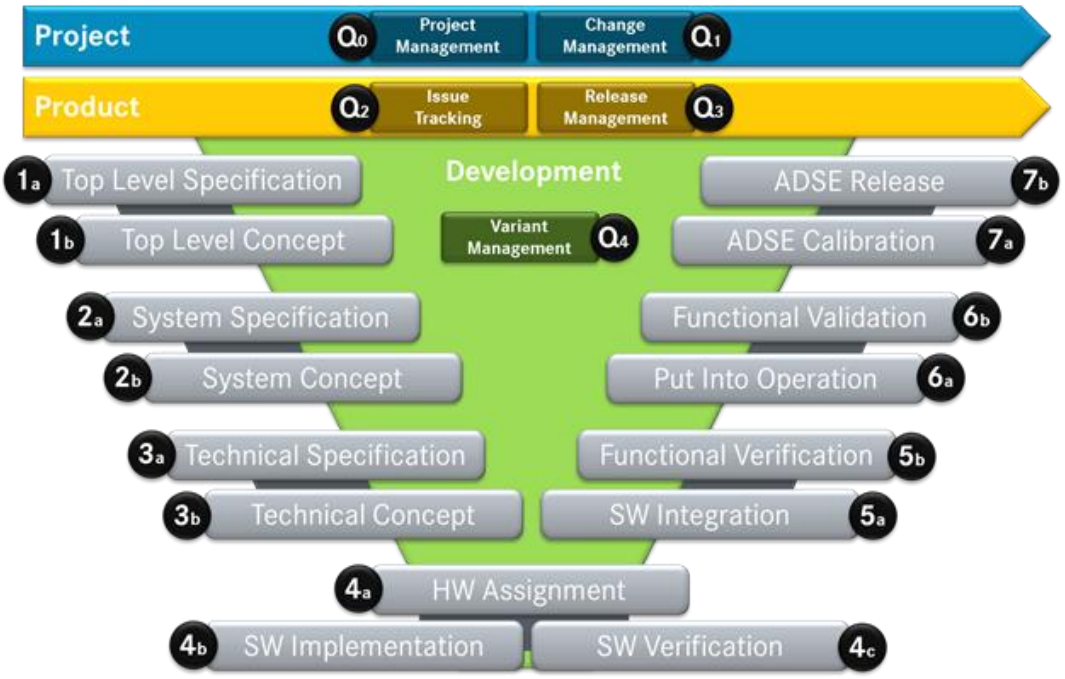

Figure 8 - (Rough) Development Process based on the V-Model

The following table lists the tools and potential interfaces relevant for IOS with references to the respective V-Model activity. The loopback interface is a feature which will be implemented,so far possible. The goal is to control activities within an integrated workflow. It helps to improve interoperability to other tools, which might take over certain aspects of the product development.

Reference to Process	Tool	Interface to Tool
Q1	Integrity	Loopback Interface
Q2	Integrity	Loopback Interface
Q3	Integrity	Loopback Interface
Q4	Integrity	Loopback Interface
1a	Integrity	Loopback Interface
1b	EA	Interface to Integrity
2a	Isograph Workbench, ikv++ Medini Analyze	Interface to Integrity
2b	EA, APIS IQ FMEA	Interface to Integrity
3a	EA	Interface to Integrity
3b	EA	Interface to Integrity
4a	EA	Interface to Integrity
4b	Simulink, C	Interface to Integrity
4c	TPT, Tessy	Interface to Integrity
5a	Simulink, C	Interface to Integrity
5b	TPT	Interface to Integrity
6a	Integrity	Loopback Interface
6b	ProveTech TA	Interface to Integrity
7a		
7b		

Table 2 - Mapping Process/Tool

Interoperability of Integrity and Enterprise Architect

The following interoperability scenarios are to be expected for an interface connecting Integrity and Enterprise Architect:

- It should be possible to manage versioned EA models in Integrity. This will require some introspection which shall be provided through a mining feature.
- The interface should allow a hierarchical abstract representation of an EA model in Integrity that contains just enough information to allow the creation of traces/bidirectional links between EA model elements and other artefacts of the lifecycle (requirements, test cases, etc.).

Interoperability of Integrity and Risk Management, Verification & Validation Tools

The following interoperability use cases are expected for an interface connecting Integrity and various verification & validation tools:
Test cases, test results and risk analysis data should be exchanged between Integrity and the following tools: APIS IQ FMEA, Isograph Fault Tree Plus, ikv++ MediniAnalyze. If a test case fails, the interface should allow an automatically creation of defects in Integrity. It should be possible to create traces/bidirectional links between verification & validation artefacts and other Integrity-managed artefacts of the lifecycle (requirements, test cases, etc.).

3.2.3 Possible Requirements towards IOS

This section enumerates a collection of ideas, which could have an impact on the implementation of a new interoperability technology. These ideas have been collected during the first project phase and are going to be reviewed, detailed and concretised within the second phase:

1. Developer tools are installed on workstations and operate very often on localized data. Central storage of files or the existence of file version systems does not imply centralism of operations.
2. The fact that Integrity is used does not mean that the services, which are provided by it, can be considered monolithically.
3. Despite network availability, work must be able to continue in cases of network outage. Such cases can be observed during vehicle probations or with mobile computing devices.
4. There should be a state-of-the-art solution to distribute developer tools to workstations and configure them for work with IOS. This includes possible updates of IOS.
5. It must be possible to manipulate large amount of properties and links in a transactional way.
6. It must be possible to create variants of the project without recreating every bit of it. This means that some copy-on-write strategies should be anticipated.
7. There are two kinds of variants of product origins. The first kind is intended and is the result of new product portfolio decisions. The second is the inevitable deviation resulting from speculative modification used for evaluation, exploration and analysis. The bricks provided by SP6 shall support both variation types and clearly inform developer, what kind of consequences his actions will have. Variants keep track of their heritage and support merging.
8. On a given workstation it must be possible to consistently work with multiple developer tools in parallel.
9. Different kinds of links should be supported which can be converted into each other:
    a. Explicit – class of link and all properties were chosen by developer
    b. Semi-explicit – the link has been introduced through a mechanism of pattern matching
    c. Implicit – the link is based on conventions but is explicitly depicted
10. Specialisation/Generalisation Links should be able to be automatically triggered

11. It must be assured that given modifications were authorized. Such authorizations should make use of roles and cascades of authorization.
12. It should be possible to create and manage authorized teams of developers without having to add them to the OEM's LDAP. This would be useful, if extern developers join the team which does not work at OEM's facilities. This could foster quick projects which are of smaller size. Small projects should run more efficiently.
13. Despite ALM support through Integrity, developers should be able to modify the current workflow to match their needs without having to modify workflow deposited in Integrity. Such customization may include simplification of process, but also an enrichment of process.
14. SP6 provided bricks should support acceleration of repeating activities.
15. Project/product dependencies should be navigated despite designated tools being unavailable.
16. It should be possible to decompose the project in order to create new subprojects which can be shared between variants.
17. Configurable constraints should be available to projects which help to keep work results consistent, even if the strict process has been left.
18. The system comprised from developer tools and IOS should be able to transport events (e.g. of change) to all concerned components, irrespective their true location in the distributed system.
19. It should be possible to define "final variants" in order to prevent branching of highly volatile variants where merging would be difficult.
20. All resources including variants should be able to be found in the network.
21. It should be possible to define barriers (kind of baselines) for variants which limit the extent of notification to a relevant family of variants.
22. The interoperability technology should be able to provide the developer with the correct local data given his desired activity. This data selection should be extendable but exaggerated amount of local copies should be prevented.
23. Introspection of models should be improved for process control.
24. Implementation of the bricks should be efficient as they will probably add to a permanent tag on all resources like processors, memory and network bandwidth.
25. Rich attributed linking of items shall be supported.
26. Semi-automatic linking of items shall be supported based on various properties (class, constellations, etc.)
27. Eventual conventions which indicate relationships shall be visualized.
28. Quick search of the project relationships shall be possible (e.g. via labels or meta-data searches). It should be possible to attach an arbitrary amount of meta-data to project objects irrespectively of their storage capacity to do so.
29. IOS-adapted developer tools should be able to get triggered to perform certain operations on behalf of third parties (e.g. Simulink starts simulation after a test was started in EA). Such operations may require the provision of a specific context made from data, constraints etc.
30. Centralized database-based systems must be incorporated without difficulty.
31. Bricks have to run in a Windows environment.

3.2.4 Possible Requirements for OSLC

At the moment OSLC-specific requirements can only be vaguely indicated:

1. Provide additional classes of OSLC objects.
2. Provide additional ways of object annotation.

3.2.5 Logical Architecture of IOS-based Projects

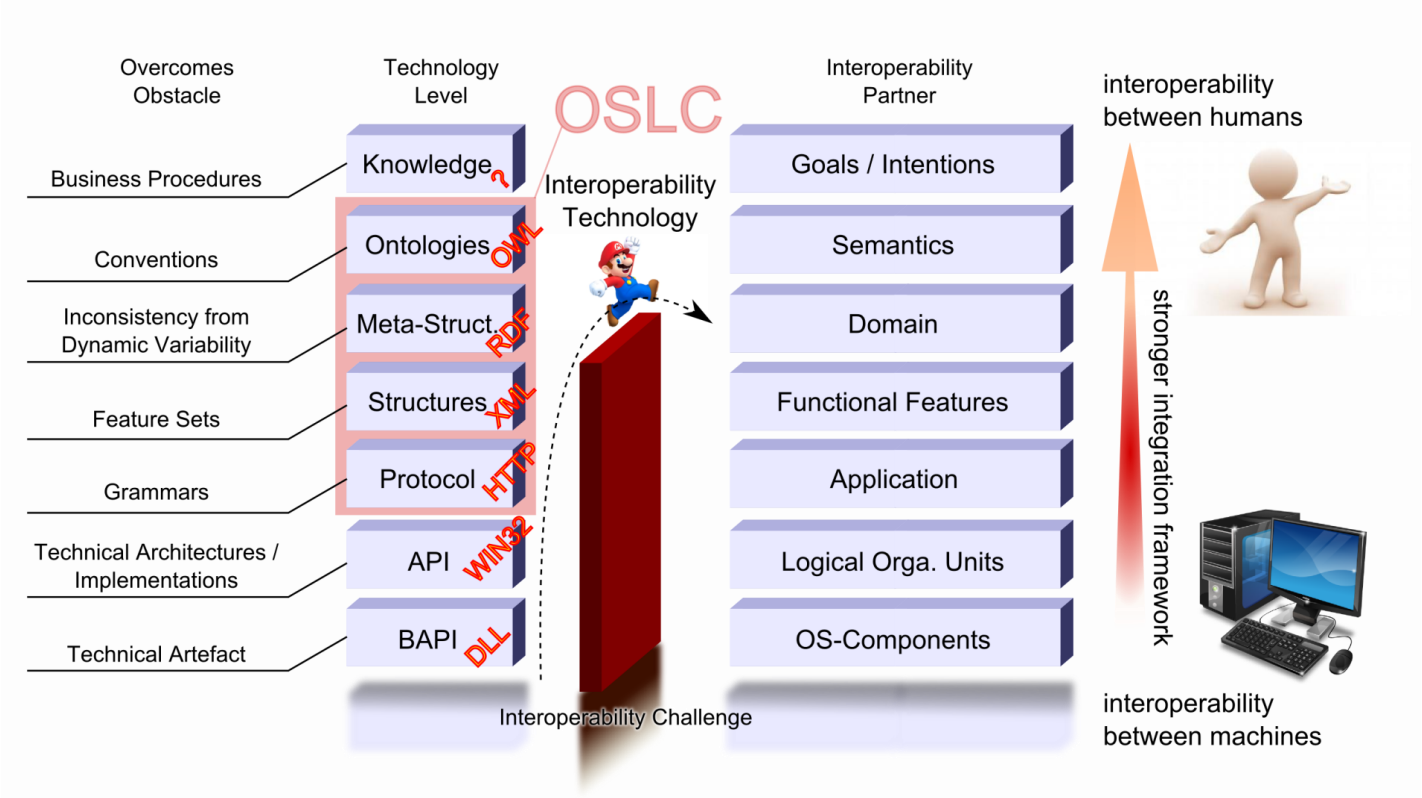

Figure 9 - Interoperability Technologies simplify Interoperability between Users

Elements of Logical Architecture

In the following section, elements which could be reflected in a logical architecture are being described:

- **Project Cloud / Project Space**. The developer typically works in project environments. The projects contain a limited and known number of objects which can be expressed as a complex graph in RDF. These objects could be of various kinds: Simulink-blocks, C-Files or a project milestone etc. This complex graph is not necessarily fully visible to developers when using developer tools but it is the actual field of work of the developer. On the developer workstation a slice of the project cloud is maintained and is the ideal storage for

meta-attributes to objects which are stored in inflexible containers. Instead of trying to interpret linking as some kind of additional interoperability the project cloud implies that developer tools are merely helpers in the process of completing the overall graph. The developer wants to switch his workstation between project clouds and wants to explore and edit them with all tools available to him at the same time. Once a project cloud gets modified by a developer it is different from others (distributed approach) and must be merged with its family members in the project space (group of workstations processing the same project) later. For this to work project clouds have to know their heritage. This approach has been proven to be very efficient in other development areas. Logical operations on the project cloud are:

  - Create
  - Merge
  - Slice
  - Refine
  - Subvariant
  - Assemble from Project Aerosols (Incomplete project clouds to be used as templates)

- **Donor Application**. A donor application is any of the developer tools which can be exposed to the project cloud via IOS. It accepts operator invocations in a donor context which is maintained by the IOS.
  - Expose and execute minimal building blocks of logic

- **Instruments**. Instruments are preconfigured operations from donor applications. They can be invoked in order to perform certain operations on the project cloud. In a trivial case an instrument is just the donor application. A purpose of an instrument can be of course e.g. native visualization of data in an artefact or the process of mining it.
  - Expose and execute customized logic for the project

- **Workstation Sandbox**. The workstation sandbox defines for the whole workstation a definitive reference to a project cloud and provides temporal storage for it. This way developer's chances to manipulate the wrong project are low. The sandbox is a place to store data and derived product variants are referencing always to a super sandbox. The sandbox which has no further parents (e.g. department sandbox) is fairly called cosmos.
  - Provide OSLC objects for file system-like artefact manipulation
  - Provide OSLC objects for driving distributed version management
  - Synchronization of data
  - Provide relative reference storage for artefacts
  - Provide caching logic for project cloud

- **Central Artefacts**. Central artefacts are typically files which can be assembled according to the needs of the situation. Such artefacts can be located in version management systems, on plain shared drives or internet pages. There is a large number of possibilities how central artefacts can be provided, mostly passive. Active OSLC proxies can decorate them in a uniform manner as version management system of file system.
  - Read
  - Write
  - Lock
  - Branch
  - Merge

- **Context**. A context is a set of collected objects in the project cloud which are intended to support developer's activities. The selection can be analysed for structure and help identify the extent to which detailed processes are applicable. The context is also useful for an efficient search of other objects in the project cloud. New objects can be introduced through the process of mining conventional artefacts.
  - Select / Drop
  - Mine
  - Infer operations or missing objects
  - Invoke instruments for selections

- **Distributed Cascade Authorization**. In any project it must be avoided that projects will be accidentally or intentionally vandalized. It is an additional concept to access rights which probably exist for central artefacts in parallel. This logical architecture element uses RSA-based encryption cascades in order to certify the legitimacy of the manipulations when trying to merge with a different variant of the project space. This way a manipulation of local authorization enforcement is unattractive. A fine grained access control can be made available as part of the project cloud which is then as project specific as desired.
  - Network of roles and the associated instruments
  - Legitimacy checks

- **Package System**. The developer in an IOS-based project will face a greater variety of artefacts during his work. The key to an excellent experience when working with IOS would be the IOS could provide native donors on local workstations. This is especially important if provided instruments require multiple donors. On the one hand, this will of course not always be possible. On the other hand even large commercial products are easy to install if floating licenses are in use. The same package system can be used to provide the project with data and model extensions for given purposes. For example, if a B2B test shall be executed then the package system could install two files in different formats but with the same logical meaning. The invoked instrument could ask the package system to provide all necessities for it to run.

- functionality a la Maven or Apt

- **Navigation and Exploration**. The developer in an IOS-based project will benefit from interoperability best, if he is allowed to understand relationships and structure of the project beyond his immediate workflow activity. The IOS' practical implementation has to support the navigation through the project and have simple means to request access to information which normally cannot be accessed.
  - Graphical Navigation
  - Tracking of requests for additional information and their processing

3.3 Roadmap

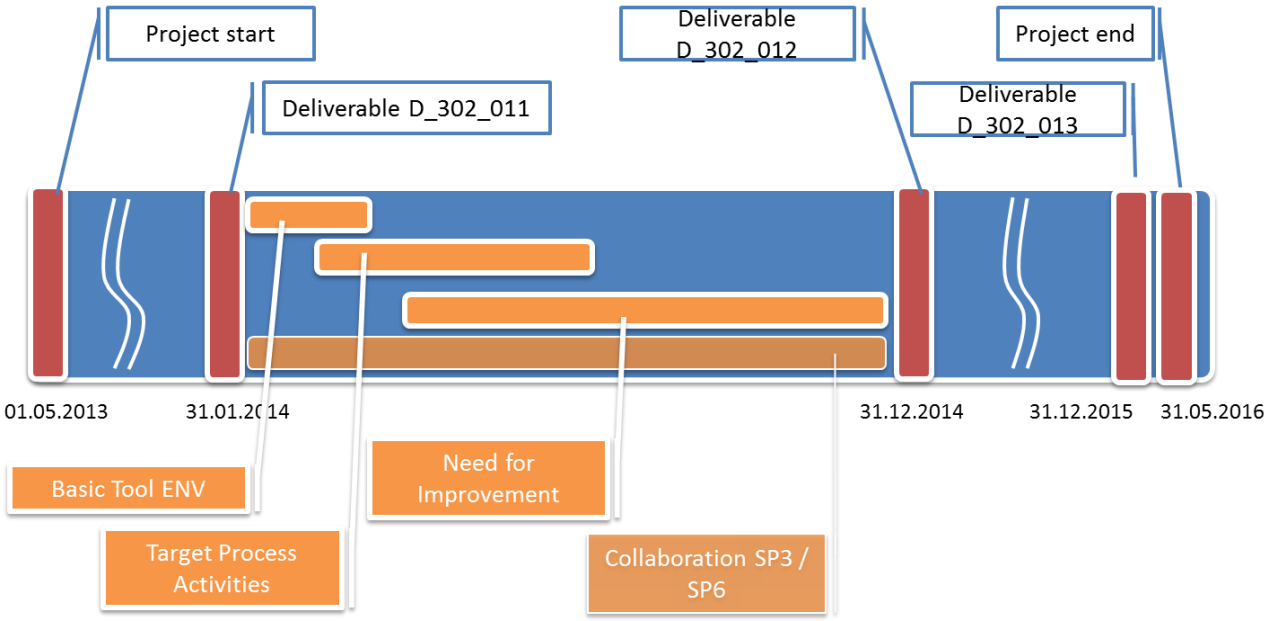

Figure 10 - Project Roadmap

Providing of basic tool environment

In parallel to analysing existing and target process activities in the ADSE project of WP3.2, a mapping to possible tool support has been made in the first phase of WP3.2. In order to integrate solutions provided by CRYSTAL it's necessary to implement a tool environment which can affiliate current and future bricks.
In the case of WP3.2 this will mainly mean an Integrity Server setup to be accessed by pilot projects users and initially store transferred data from current project status. Furthermore, workflows need to be implemented to support process activities as mentioned in chapter 4.

Application and refinement of target process activities

As soon as a basic tool environment is put in service, defined target process activities have to be applied. In the pilot project this will mean introduction of new activities as well as adaption of existing or migration between former and new tool environment.
Because of direct integration into a productive environment, only small changes at a time can be applied. Furthermore WP3.2 is confident to receive valuable feedback from this application steps from the very first moment.

Elicitation of need for improvement

Until next Milestone one main task of WP3.2 is to gather information about further improvement after application of the first set of target activities. Since it's a CRYSTAL strategy that the WP results all over the project are being iteratively improved, WP3.2 is mainly interested in identifying critical process-, interoperability- and tool-weaknesses to be concentrated on in the next step.

Collaboration with other Use Cases within SP3

Within SP3, collaboration is of high value because scenarios have similarities and solutions are partially portable. Therefore and to give a summary of SP3s results, WP3.7 (public automotive use case) has been created and WP3.2 is actively contributing to it. This participation will be continued until the end of the project because it's a key factor to projects success. Furthermore, contribution to WP3.8 (automotive ontology) is being seen as an important need within CRYSTAL and WP3.2s collaboration regarding ontologies will also be continued.

Interaction with SP6

Progress towards interoperability solutions within CRYSTAL will be achieved by applying SP6 solution bricks. Thus it's mandatory for WP3.2 to collaborate actively with relevant SP6 WPs. Most of our tool partners are directly involved in SP6 Brick development and via our use case details and IOS requirements we will continue this tight interaction during the next project phase.

4 Conceptual Work

The use case described in chapter 2 comprises the development of a host computer in the context of automated driving in specific environments. The first achievements are aligned with the steps of the CRYSTAL “Interoperability Needs Capturing Process” for the different domains. According to this approach in chapter “Engineering Methods” we provided a high-level description of the use case and its context.
The process definition including the corresponding process activities will be presented in chapter 4.1.
We derived several engineering methods from the process activities. Some of them will be described in chapter 4.2.
The analysis of the engineering process revealed specific challenges related to tool interoperability. Those challenges will be discussed in chapter 4.3.
Another achievement is a first draft of a concept for mapping the content of development artefacts to activities and tools. This concept will be introduced in chapter 4.4.

4.1 Engineering Methods

4.1.1 List of Engineering Methods Sorted by Topic

We have identified a number of possible engineering methods for the two major topics **Functional Safety** and **Verification/Validation**. The following list is a second draft which is by no means complete. We expect to come up with more topics and engineering methods as well as we expect to change or remove some of them.

Functional Safety Engineering Methods

- Perform Hazard And Risk Analysis
- Perform Fault Tree Analysis
- Perform Failure Mode and Effects Analysis
- Define Functional Safety Concept
- Define Technical Safety Concept
- Conduct Safety Case
- Perform Failure Injection Test

Verification/Validation Engineering Methods

- Create Test Plan
- Create Test Specification
- Create Test Implementation for HiL (Automatic Testing)
- Detect Run Time Errors (Static Analysis)
- Analyse Worst Case Execution Time (Static Analysis)

- Perform Static Code Analysis
- Perform SiL-Test
- Perform HiL-Test
- Perform MiL-Test

Project / Product Management

- Collect, analyse and control Issues
- Abstraction and Reporting
- Analyse the Impact of Changes
- Control Changes

Variant Management

- Define Valid Configurations

SW Engineering

- Coding or Modelling
- Check Coding Guidelines
- Create new Requirement Entry

4.1.2 Engineering Methods in the V-Model

In Figure 11 we have assigned the different engineering methods to the different steps in the V-Model.
All methods from section 4.1.1 as well as some additional methods from variant management, project management, change management and issue tracking have been assigned.

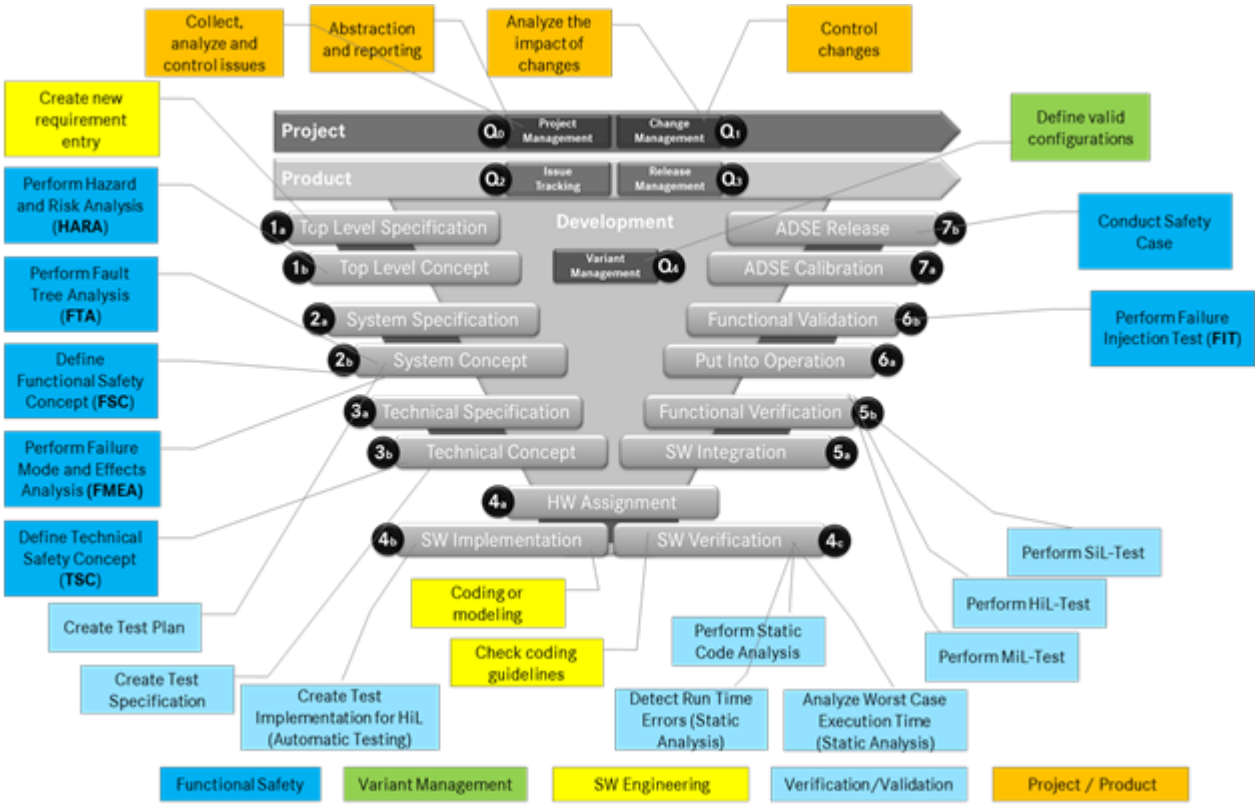

Figure 11 - Engineering Methods assigned to different Steps in the V-Model

4.1.3 Detail View of an Engineering Method

We analysed one engineering method in detail in order to come up with useful information for the engineering method template that we had to fill out. Therefore, we chose the engineering method “Create New Requirement Entry” that describes how a requirements engineer enters a new requirement into his requirements management (RM) tool of choice. We deliberately chose to omit specific tool names to make the example more general and independent from specific tools. Figure 11 depicts the detailed view for this engineering method.
While the engineer is entering the requirement details, the RM tool sends all existing requirements and the newly entered content to an analyser tool. The analyser tool then compares the existing requirements with the new requirement and detects similarities. Those similarities are then sent back to the RM tool, which can display hints to the RM engineer. Thus, the tool could provide information about possible duplicates of a requirement, or e.g., propose tags or different wordings that have been used for similar requirement entries.
The filled-out templates provided to SP 6 can be found in Annex I: Engineering Methods.

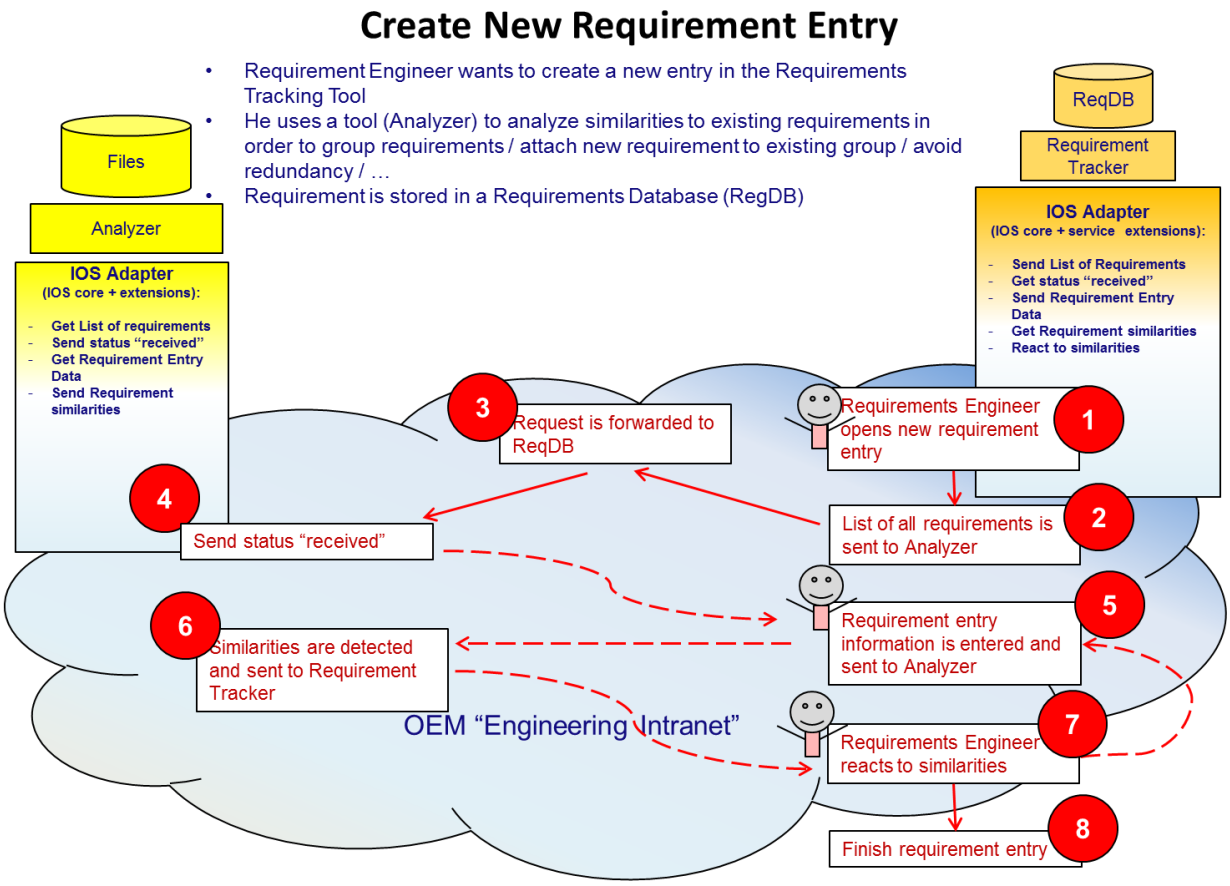

Figure 12 - Detailed View for the Engineering Method "Create New Requirement Entry"

4.2 Concept for Mappings

The initial cost associated with the introduction of traceability links between different software development artefacts in already existing projects grows with the age of the project. As the projects considered in the Daimler use case are already very mature, we need to search for methods and techniques to reduce this initial cost to allow for a more efficient introduction of traceability links (or **mappings**) between different artefacts or artefact elements. We distinguish between two different kinds of methods:

1. Methods that allow the initial creation of mappings between artefacts or elements of artefacts.
2. Methods that support the validation and verification of existing mappings.

We call the combination of a mapping together with a description of the mapped artefacts, a method for the initial creation of the mapping, and a method for validation and verification of an existing mapping a **tracing framework**.

4.2.1 Approach

Our approach consists of the following phases.

1. Analysis of existing artefacts and their interconnections.
2. Identification and classification of possible mappings using templates.
3. Design of methods for the initial import of artefacts in the tracing framework.
4. Design of methods for validation and verification of existing mappings during the software development process.
5. Evaluation of the methods from phase 3 and 4.
6. Implementation of the most suitable method for the initial import of artefacts.
7. Implementation of the most suitable method for the validation and verification of existing mappings.
8. Evaluation of the implemented method on the basis of a real-world example.

We are currently executing phase 1 and 2 in parallel with phases 3 and 4 per artefact using the templates from Table 3.

	Template
Mappings	1. Name/ID 2. Description 3. Type of mapping (Semantics) a. Signature -- M x N relation between artefacts b. Affected artefacts c. Meaning of relation in natural language d. Relation to other mappings 4. Version 5. Applicable initial creation methods 6. Applicable validation & verification methods

Methods	1. Name/ID 2. Description 3. Preconditions 4. Type (Initial creation or validation & verification) 5. Version 6. Evaluation with respect to: a. Automation potential b. Fault tolerance c. Completeness d. Initial costs e. Running costs
Artefacts	1. Name/ID 2. Description 3. Version 4. Level of formalization a. Formal, i.e. formal syntax and semantics b. Semi-formal, i.e. partly formal syntax or semantics c. Unstructured, i.e. without formal syntax or semantics 5. Potential target elements

Table 3 - Identification and Classification Templates

4.2.2 Example of Tracing Framework

Version control systems (VCS) and issue tracking systems (ITS) are ubiquitous tools in professional software development processes. Those tools create – among others – the artefact types *change set* and *issue tracking item* (item).
A change set contains the changes between two revisions of a VCS. Changes are typically grouped by files and can therefore be analysed on a file-by-file basis. If files contain source code implemented in the same programming language, change sets can also be grouped by **symbols**, i.e. functions and variables of a represented program. We call the set of all symbols of a change set the **changed symbol set**. A changed symbol set can be extracted automatically from a change set by employing well-known techniques, e.g. from the area of compiler construction.
Most ITS can already keep track between items and revisions of a VCS. This is normally done manually by developers, which insert special instructions in their commit log messages to inform the ITS of the affected items. Together with the automatic extraction of changed symbol sets one can map symbol sets to single items in the ITS (e.g. by simply creating the union over all changed symbol sets connected to a change set). The resulting mapping between symbols and items can now be used for further analysis.
Assumed that the complete version history is available, the method is applicable for both, initial creation and verification and validation of mappings between source code and ITS items.

5 Terms, Abbreviations and Definitions

Please add additional terms, abbreviations and definitions for your deliverable.

ADSE	**A**utonomous **D**riving in **S**pecific **E**nvironments
ALM	**A**pplication **L**ifecycle **M**anagement
ALU-FR	**A**lbert-**L**udwigs-**U**niversität **FR**eiburg
API	**A**pplication **P**rogramming **I**nterface
AUTOSAR	**AUT**omotive **O**pen **S**ystem **AR**chitecture
BAPI	**B**usiness **A**pplication **P**rogramming **I**nterface
cf.	**c**on**f**er
CO	**CO**nfidential, only for members of the consortium (including the JU).
COM	**C**omponent **O**bject **M**odel
CORBA	**C**ommon **O**bject **R**equest **B**roker **A**rchtecture
CRYSTAL	**CR**itical S**YST**em Engineering **A**cce**L**eration
D	**D**emonstrator/**D**eliverable
DAI	**DAI**MLER
EA	**E**nterprise **A**rchitect (SparxSystems)
e.g.	**e**xempli **g**ratia
ENV	**ENV**ironment
etc.	**et c**etera
FIT	**F**ailure **I**njection **T**est
FMEA	**F**ailure **M**ode and **E**ffect **A**nalysis
FSC	**F**unctional **S**afety **C**oncept
FTA	**F**ault **T**ree **A**nalysis
HARA	**H**azard **A**nd **R**isk **A**nalysis
HiL	**H**ardware **i**n the **L**oop
ID	**ID**entification
i.e.	**i**d **e**st
ISO	**International Organization for Standardization**
IOS	**I**nter**O**perability **S**pecification
IT	**I**nformation **T**echnology
ITS	**I**ssue **T**racking **S**ystem
LDAP	**L**ightweight **D**irectory **A**ccess **P**rotocol
MiL	**M**odel **i**n the **L**oop
No	**N**umer**o**

O	**O**ther
OEM	**O**riginal **E**quipment **M**anufacturer
OOP	**O**bject **O**riented **P**rograming
OSLC	**O**pen **S**ervices for **L**ifecycle **C**ollaboration
P	**P**rototype
PLM	**P**roduct **L**ifecycle **M**anagement
PP	Restricted to other **P**rogram **P**articipants (including the JU).
PTC	**P**arametric **T**echnology **C**orporation
PU	**PU**blic
R	**R**eport
RE	**RE**stricted to a group specified by the consortium (including the JU).
ReqDB	**Req**uirements **D**ata **B**ase
RDBMS	**R**elational **D**ata **B**ase **M**anagement **S**ystem
RDF	**R**esource **D**escription **F**ramework
REST	**RE**presentational **S**tate **T**ransfer
RM	**R**equirements **M**anagement
RSA	**R**ivest**,** **S**hamir and **A**dleman
RTP	**R**eference **T**echnology **P**latform
R&D	**R**esearch **&** **D**evelopment
SE	**S**oftware **E**ngineering/**S**ystems **E**ngineering
SiL	**S**oftware **i**n the **L**oop
SP	**S**ub**P**roject
SQL	**S**tructured **Q**uery **L**anguage
SW	**S**oft**W**are
TSC	**T**echnical **S**afety **C**oncept
TUB	**T**echnische **U**niversität **B**erlin
UC	**U**se **C**ase
URI	**U**niform **R**esource **I**dentifier
WP	**W**ork **P**ackage
UML	**U**nified **M**odeling **L**anguage
V	**V**ersion
VCS	**V**ersion **C**ontrol **S**ystem

6 Annex

6.1 Annex I: Engineering Methods

Engineering Method: UC2302_LinkFromRequirementToModelElement					
Purpose:					
Comments:					
Pre-Condition		**Engineering Activities**		**Post-Condition**	
Requirements are complete, analyzed, without contradictions. Initial system model has been created in Modeling Tool EngineeringMethod: "Determine Modeling Context" has been executed.		1. In RequirementManagementTool, open service "ModelElementSelector" to navigate to context (modeling element) to which an external link should be created. 2. ModelingTool responds with List of ModelElements (could be a subtree, a top level element or a list of elements). 3. User selects one or more ModelElements to which an external link should be created. 3. User can optionally fill in meta-information for links (Type, name, bi-directional=true|false) 4. SelectorService presents to the user a preview, which links will be created based on the user selection. 5. User accepts. 6. In both tools (RequirementsMangementTool and ModelingTool) , external links are added to respective requirements and model elements. Links are navigable by the user such that when the user clicks on an external link, the respective tool is opened. Alternatively at least a preview is shown when hovering over the link.		Links are created between the selected Requirements and ModelElements	
				Notes:	
Artefacts Required as inputs of the Activities		**Artefacts used internally within the Activities**		**Artefacts Provided as outputs of the Activities**	
Name		Name		Name	
Generic Type: (Tool or language independend type)	Requirements	Type:		Generic Type: (Tool or language independend type)	External Link
Required Properties: (Information required in interactions between steps)	ID, Title, External Link Reference	Properties:		Provided Properties: (Information provided in interactions between steps)	URL, Label, Type, Name, isBedirectional
Description & Interoperability Additional Constraints:		Description:		Description & Interoperability Additional Constraints:	
Name		Name		Name	
Generic Type: (Tool or language independend type)	ModelElement	Type:		Generic Type: (Tool or language independend type)	
Required Properties: (Information required in interactions between steps)	ID, Name, External Link Reference	Properties:		Provided Properties: (Information provided in interactions between steps)	
Description & Interoperability Additional Constraints:		Description:		Description & Interoperability Additional Constraints:	
Name		Name		Name	
Generic Type: (Tool or language independend type)		Type:		Generic Type: (Tool or language independend type)	
Required Properties: (Information required in interactions between steps)		Properties:		Provided Properties: (Information provided in interactions between steps)	
Description & Interoperability Additional Constraints:		Description:		Description & Interoperability Additional Constraints:	

Engineering Method: 3.2_InsertLinkBetweenChangesetAndIssueTrackerEntry_1					
Purpose: Given an entry of the issue tracking system (ITS), the project team member has done some work related to the entry and wants to commit an update to the version control system (VCS). The					
Comments:					
Pre-Condition		**Engineering Activities**		**Post-Condition**	
An entry in the ITS exists (and has an id). An update to the VCS is ready.		The project team member commits the update to the VCS. In detail: 1. In the VCS, prepare the entering of a commit log message. 2. Enter commit log message with a keyword and the id of the given ITS entry (e.g., "[entry #27]"). 3. VCS receives commit. 4. VCS parses commit log message (e.g. with a post commit hook). 5. VCS identifies the reference to the ITS entry. 6. VCS invokes a request to ITS to add a comment regarding the entry with the given id. The comment contains the revision number of the VCS changeset, possibly a url to the changeset (depending on the VCS system), and the commit log message. 7. The ITS receives the request and adds the comment to the log message		The ITS entry has a link to the changeset of the update in the ITS.	
Notes:		Notes: This entry describes a 1-to-1 relation between commits		Notes:	
Artefacts Required as inputs of the Activities		**Artefacts used internally within the Activities**		**Artefacts Provided as outputs of the Activities**	
Name	ITS entry	Name		Name	ITS entry
Generic Type: (Tool or language independend type)	ITS entry	Type:		Generic Type: (Tool or language independend type)	ITS entry
Required Properties: (Information required in interactions between steps)	ITS entry has a unique id and it has a field to store link to VCS changeset	Properties:		Provided Properties: (Information provided in interactions between steps)	ITS entry has a unique id and it has a field to store link to VCS changeset
Description & Interoperability Additional Constraints:		Description:		Description & Interoperability Additional Constraints:	
Name	VCS changeset	Name		Name	VCS changeset
Generic Type: (Tool or language independend type)	VCS changeset	Type:		Generic Type: (Tool or language independend type)	VCS changeset
Required Properties: (Information required in interactions between steps)	needs unique id, needs way of storing link to id of issue tracker entry	Properties:		Provided Properties: (Information provided in interactions between steps)	needs unique id, needs way of storing link to id of issue tracker entry
Description & Interoperability Additional Constraints:		Description:		Description & Interoperability Additional Constraints:	
Name		Name		Name	
Generic Type: (Tool or language independend type)		Type:		Generic Type: (Tool or language independend type)	
Required Properties: (Information required in interactions between steps)		Properties:		Provided Properties: (Information provided in interactions between steps)	
Description & Interoperability Additional Constraints:		Description:		Description & Interoperability Additional Constraints:	

Engineering Method: UC3.2_VerifyModifiedSafetyFeatureInSimulation_0					
Purpose: Verify efficacy of safety requirement implementation after changes during an early stage of system development					
Comments: This method uses advantages of MiL testing for safety requirements. It appears sensible to maintain confidence into first class requirements (which are safety reqs.) right from the start.					
Pre-Condition		**Engineering Activities (made of steps)**		**Post-Condition**	
- a requirement with some SIL-like attribute (e.g. ASIL=A..D) has been marked as modified - relevant source-code/-model was updated to reflect the changes in the requirements - consistency between safety requirement and safety goal was reassured - a test-set exists that covers the safety requirement - model is enriched with failure rate information - model is executable in relevant simulated environement - derived test-set was updated to reflect the changes of safety requirements context descriptor exists for current workflow		1. engineer figures out in the safety explorer which safety requirements have not yet been validated 2. engineer selects all test-cases necessary for safety requirement verification, links help to do so 3. engineer triggers static & dynamic safety-tests in simulated environments. Static tests estimate failure probabilities under different conditions. Dynamic test test efficacy of solution. This action causes the creation of a safety-run object, which is something like a snapshot of relevant objects 4. the results of tests are attached to the safety-run object which - are logged in project's automatic safety-log 5. status of all safety requirements up to safety goals is updated 6. if test fail then the engineer will provide an explanation, an estimation of difficulty for a fix and a proposition how to fix 7. new situation is reflected in safety explorer		- testing activity logged as entity to project's safety log - results of all test cases logged (static and dynamic) - test appears in worflow history and at least in safety explorer - status of requirements and goals updated - proposition added to knowledge base
Notes: context descriptor is a container with a collection of highlighted requirements, models and simulation objects, user data (safety engineer), and guidance information		Notes: requirements are classifed as "fulfilled", "compromised", "incomplete", "manually verified", "inconclusive" and safety goals are marked as "attained", "endangered" or "violated"		Notes:	
Artefacts Required as inputs of the Activities		**Artefacts used internally within the Activities (optional)**		**Artefacts Provided as outputs of the Activities**	
Name	safety requirement	Name	safety requirement	Name	safety-run
Generic Type: (Tool or language independend type)	abstract object in a graph dependency with other nodes	Type:	-||-	Generic Type: (Tool or language independend type)	node linked to snapshotted objects
Required Properties: (Information required in interactions between steps)	linkable, invalidation flag, navigable to parent and sibling requirements, snapshotable	Properties:	-||-	Provided Properties: (Information provided in interactions between steps)	date, user-id, ability to contribute to overall project's safety case
Description & Interoperability Additional Constraints:		Description:		Description & Interoperability Additional Constraints:	
Name	test case	Name	test case	Name	workflow object
Generic Type: (Tool or language independend type)	program to run a model simulation	Type:	-||-	Generic Type: (Tool or language independend type)	instance of current workflow process
Required Properties: (Information required in interactions between steps)	executable, returns results, snapshotable	Properties:	-||-	Provided Properties: (Information provided in interactions between steps)	petri-net like automaton instance with objects linked to states
Description & Interoperability Additional Constraints:		Description:		Description & Interoperability Additional Constraints:	
Name	model	Name	model	Name	knowledge base
Generic Type: (Tool or language independend type)	mechanism formulation	Type:	-||-	Generic Type: (Tool or language independend type)	tracker like software with strong mining capabilities
Required Properties: (Information required in interactions between steps)	snapshotable, specific language implementation, distributed among different tools	Properties:	-||-	Provided Properties: (Information provided in interactions between steps)	human typed information (text,pictures,audio,movies) for humans, rating in various ways
Description & Interoperability Additional Constraints:		Description:		Description & Interoperability Additional Constraints:	
Name	safety explorer	Name	safety explorer	Name	
Generic Type: (Tool or language independend type)	browser for safety related issues, objects and information	Type:	-||-	Generic Type: (Tool or language independend type)	
Required Properties: (Information required in interactions between steps)	per active engineer, current project representation, graphical user interface	Properties:	-||-	Provided Properties: (Information provided in interactions between steps)	
Description & Interoperability Additional Constraints:		Description:		Description & Interoperability Additional Constraints:	
Name	safety log	Name	safety log	Name	
Generic Type: (Tool or language independend type)	historical database of safety related events (like test runs of safety test cases)	Type:	-||-	Generic Type: (Tool or language independend type)	
Required Properties: (Information required in interactions between steps)	collection of safety-run objects	Properties:	-||-	Provided Properties: (Information provided in interactions between steps)	
Description & Interoperability Additional Constraints:		Description:		Description & Interoperability Additional Constraints:	

Engineering Method: 3.2_CreateNewRequirement_1					
Purpose: Requirement Engineer wants to create a new entry in the Requirements Tracking Tool. He uses a tool (Analyzer) to analyze similarities to existing requirements in order to group requirements / attach new requirement to existing group / avoid redundancy. Requirement is stored in a Requirements Database (RegDB).					
Comments:					
Pre-Condition		**Engineering Activities**		**Post-Condition**	
List of informally collected requirements is available		1. Requirements Engineer opens new requirement entry 2. List of all requirements is sent to Analyzer 3. Request is forwarded to ReqDB 4. Send status "received" 5. Requirement entry information is entered and sent to Analyzer 6. Similarities are detected and sent to Requirement Tracker 7. Requirements Engineer reacts to similarities 8. Finish requirement entry		Requirement Database is updated depending on new requirement	
Notes:		Notes: control structures, e.g., loops, conditional behavior can not be expressed in this template		Notes:	
Artefacts Required as inputs of the Activities		**Artefacts used internally within the Activities**		**Artefacts Provided as outputs of the Activities**	
Name	Requirement	Name	Requirement Tracker internal requirement	Name	Requirement
Generic Type: (Tool or language independend type)	Natural Language Requirement	Type:	Requirement as stored in Requirement Tracker	Generic Type: (Tool or language independend type)	Natural Language Requirement
Required Properties: (Information required in interactions between steps)	Req.-ID, text description in natural language, component ID	Properties:	Req.-ID, text description, component ID,author name, assignee, responsible, priority, severity, release, version, milestone, ..., plus links to test cases, version management, project management, code, "Lastenheft"	Provided Properties: (Information provided in interactions between steps)	Req.-ID, text description in natural language, component ID
Description & Interoperability Additional Constraints:		Description: detailed requirement information		Description & Interoperability Additional Constraints:	
Name	Similarity Detection Result	Name	Analyzer internal requirement	Name	Similarity Detection Result
Generic Type: (Tool or language independend type)	List of Integers	Type:	Requirement as stored in Analyzer	Generic Type: (Tool or language independend type)	List of Integers
Required Properties: (Information required in interactions between steps)	List of Req.-Ids	Properties:	Req.-ID, text description, component ID	Provided Properties: (Information provided in interactions between steps)	List of Req.-Ids
Description & Interoperability Additional Constraints:		Description: all requirement data the analyzer needs for		Description & Interoperability Additional Constraints:	
Name		Name		Name	
Generic Type: (Tool or language independend type)		Type:		Generic Type: (Tool or language independend type)	
Required Properties: (Information required in interactions between steps)		Properties:		Provided Properties: (Information provided in interactions between steps)	
Description & Interoperability Additional Constraints:		Description:		Description & Interoperability Additional Constraints:	