Rene Obendrauf, BSc

Semi-Automated Safety Management and Safety Case Generation

MASTER'S THESIS

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme: Software Development and Business Management

submitted to

Graz University of Technology

Supervisor

Dipl.-Ing. Dr.techn. Christian Kreiner
Dipl.-Ing. Dr.techn. Gerhard Griessnig

Institute of Technical Informatics
Head: Univ.-Prof. Dipl.-Inform. Dr.sc.ETH Kay Römer

Graz, March 2016

AFFIDAVIT

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master‘s thesis dissertation.

Date

Signature

Abstract

Developing automotive safety-critical systems is nowadays a very complex and challenging task due to the high number of integrated control units and the intricate interaction of these systems. In order to cope with this complexity, it is necessary to define and set up an efficient and effective safety management that defines a consistent and continuous development process throughout the entire lifecycle and across different organizations.

The safety management process is not an independent process, it is directly connected to the quality and product development processes of the involved companies. ISO26262 (the standard for functional safety in the automotive industry) covers this safety management process with more than 100 workproducts and over 1,000 requirements. Therefore, the conversion of this complex safety management process into an existing quality process landscape and its maintenance is a time-consuming and costly task. If integrated properly in an organization, the benefits of an effective safety management are huge. It can raise the efficiency of further product developments and can therefore result in lower costs. In today's business, cost is one of the most important key factors.

Managing established safety processes is another challenge for the companies because the manual creation and maintenance of different safety-related projects and their according safety activities in context of ISO26262 is a laborious and error-prone task. To support these tasks, a tool-dependent and tool-supported environment is needed.

As a consequence, this thesis describes the design and implementation of a novel collaborative tool architecture for creating and maintaining the safety management activities of different standards throughout the entire safety lifecycle. This tool architecture is based on a common client-server architecture that allows multiple users to concurrently work on different projects within the tool.

An integrated report generator supports users by providing semi-automated documents which serve as basis for customer release documentations. The development interface agreement is a report that can automatically be generated from the tool. Additionally, the tool allows the generation of a process-based safety case which provides evidence for the fulfillment of workproducts and requirements by the integrated safety process. Thus, the provided tool environment can further help reduce time and costs.

Keywords

ISO26262; Functional Safety Management; Safety Lifecycle; Process-Based Safety Case; Development Interface Agreement; Collaborative Tool Environment

Kurzfassung

Die Entwicklung sicherheitskritischer Systeme ist heutzutage eine sehr aufwendige und komplexe Aufgabe aufgrund der hohen Anzahl an integrierten und interagierenden Steuereinheiten. Um dieser Komplexität gerecht zu werden, ist es notwendig ein effektives und effizientes funktionales Sicherheitsmanagement aufzubauen und einzuführen. Dieses funktionale Sicherheitsmanagement soll einen konsistenten und durchgehenden Entwicklungsprozess über den gesamten Lebenszyklus definieren.

Dieser funktionale Sicherheitsmanagementprozess ist kein eigenständiger Prozess, da er direkt mit dem Qualitäts- und Produktentwicklungsprozess verbunden ist. Die ISO26262 ist der Standard für funktionale Sicherheit in der Automobilindustrie und definiert einen Sicherheitsmanagementprozess mit mehr als 100 verschiedenen Arbeitspaketen, welche aus über 1.000 verschiedenen Anforderungen bestehen. Aus diesem Grund ist die Integrierung und Wartung dieses komplexen funktionalen Sicherheitsmanagementprozesses in eine bestehende Prozesslandschaft eine sehr zeitaufwendige und fehleranfällige Arbeit. Die Vorteile aus einer ordnungsgemäßen Integration sind enorm. Es kann die Leistungsfähigkeit für weitere Produktentwicklungen steigern und Kosten einsparen. Im heutigen Geschäftsumfeld ist die Einsparung von Kosten einer der wichtigsten und entscheidendsten Komponenten. Eine weitere Herausforderung stellt die Wartung eines integrierten funktionalen Sicherheitsmanagements dar. Durch die manuelle Erstellung und Aufrechterhaltung der verschiedenen sicherheitsbezogenen Projekte und die damit verbundenen Sicherheitsaktivitäten, welche in der ISO26262 definiert sind, ergeben sich viele aufwendige und fehleranfällige Arbeitsschritte. Um diese Arbeitsschritte unterstützen zu können, wird ein toolabhängiges und toolunterstützendes Umfeld benötigt.

Folglich wird in dieser Arbeit der Entwurf und die Implementierung eines neuartigen, kollaborativen Tool Frameworks vorgestellt. Dieses Tool Framework soll die Erstellung und Wartung von funktionalen Sicherheitsmanagementaktivitäten ermöglichen, welche von verschiedenen Standards vorgegeben werden. Die Architektur dieses Tools basiert auf einem Client-Server Modell, welches das gleichzeitige Arbeiten an verschiedenen Projekten ermöglicht. Ein zusätzlich integrierter Dokumentengenerator unterstützt den Anwender beim automatischen Erstellen von Dokumenten, welche beispielsweise als Basis für Berichte an Kunden verwendet werden können. Das Development Interface Agreement Dokument kann automatisch aus dem Tool generiert werden. Zusätzlich kann auch noch ein prozessbasierter Sicherheitsnachweis erzeugt werden. Dieser zeigt den Nachweis für die Erfüllung der Arbeitspakete und Anforderungen, welche im funktionalen Sicherheitsmanagementprozess eingebettet sind. Demzufolge können durch den Einsatz des bereitgestellten Tools Zeit und Kosten gespart werden.

Schlüsselwörter

ISO26262; Sicherheitslebenszyklus; Funktionales Sicherheitsmanagement; prozessbasierter Sicherheitsnachweis; Development Interface Agreement; kollaboratives Tool Framework

Acknowledgments

The contributions presented in this thesis were conducted in cooperation with the ITI - Institute for Technical Informatics, Graz University of Technology and the AVL List GmbH Graz. Foremost, I want to thank Dr. Christian Kreiner from the ITI, Dr. Gerhard Griessnig and Dr. Roland Mader from AVL for advising me how to compose and complete this thesis and for a good and fruitful collaboration.

In particular, I would like to thank my colleagues Philipp Prinz, Dominik Mößlang and Labinot Xhafa at AVL List GmbH for excellent discussions providing qualified comments, supporting the evaluation of the tool and enabling an atmosphere of friendship. Additional thanks to all people from the AVL safety team which always supports me during the performance of the thesis by providing helpful information and making good discussions.

I would like to express my greatest gratitude to my parents, Paul Obendrauf and Annemarie Obendrauf, and to my girlfriend, Melanie Harb, for their life-long support and their understanding during the completion of this thesis. Without the help of these people, the thesis would not have been possible. Thank you!

Graz/Austria
March, 2016 Rene Obendrauf, BSc

Contents

List of Figures

Listings

List of Tables

Abbreviations

API	Application Programming Interface
AVL	Anstalt für Verbrennungskraftmaschinen List GmbH
AVL-PTE	AVL Key Area - Power Train Engineering
DIA	Development Interface Agreement
E/E	Electrical/Electronic
E/E/PE	Electrical/Electronic/Programmable Electronic
EER	Enhanced Entity-Relationship
FIT	Failure in Time
FMEA	Failure Mode and Effects Analysis
FSC	Functional Safety Concept
FSM	Functional Safety Manager
FSM-P	Functional Safety Manager in Project
FSPO	Functional Safety Project Overview
FTA	Fault Tree Analysis
GSN	Goal Structuring Notation
GUI	Graphical User Interface
HARA	Hazard Analysis and Risk Assessment
HQL	Hibernate Query Language
HSI	Hardware-Software Interface
HTTP	Hypertext Transfer Protocol
IIS	Internet Information Services
ISO	International Organization for Standardization
LDAP	Lightweight Directory Access Protocol
MVVM	Model-View-View-Model
OEM	Original Equipment Manufacturer
ORM	Object Relational Mapping
QMS	Quality Management System
RDBMS	Relational Database Management System

SQL	Structured Query Language
TSC	Technical Safety Concept
TSD	Technical Safety Developer
UML	Unified Modeling Language
URI	Uniform Resource Identifier
VDI	Verein Deutscher Ingenieure
VDP	Virtual Private Network
V&V	Verification & Validation
WCF	Windows Communication Foundation
WPF	Windows Presentation Foundation
XAML	Extensible Application Markup Language
XML	Extensible Markup Language

1. Introduction

1.1. Motivation

According to the increasing complexity of automotive embedded systems (Figure 1.1), functional safety is nowadays a more and more important issue. The development of safety-critical systems require structured and well-defined workflow steps described by ISO26262, the standard for functional safety of the automotive domain. ISO26262 [9] is an adaption of IEC61508 [10] (Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems (E/E/PE, or E/E/PES)), which is the main functional safety standard of all different domains and describes functional safety as "absence of unreasonable risk due to hazards caused by malfunctioning behavior of Electrical/Electronic systems."

The work in [11] provides an overview of ISO26262 and the according lifecycle with the associated parts. The ISO26262 standard consists of ten parts that are further described in Section 2.1.3. These ten parts consist of nine normative parts and one informative part that can be used as a guideline.

ISO26262 contains a subsection entitled "Management of functional safety", which defines the automotive safety lifecycle which starts with the safety management, goes on with the development, operation, service and ends with the decommissioning of the system. It furthermore provides support for tailoring the different safety activities during a project and the according lifecycle. Part 8 [12] additionally contains processes and gives an informative example for a "Development Interface Agreement (DIA)" (Section 2.1.2), which is a document describing who (customer, supplier) is responsible for the different safety activities during the safety lifecycle.

Because of the complex development behavior of the automotive area, structured development processes and workflows are needed in order to establish an effective safety management within the whole organization. Implementing functional safety within the organization is a complex and time-consuming task because the safety management process contains over 100 workproducts [13] that consist of over 1,000 single requirements. Furthermore, it is not a particular process and coincides the quality and product development processes of a well-defined organization culture.

To cope with such a complexity, tool support for automatic generation of artifacts is needed in order to avoid faults and failures by the manual creation and maintenance of artifacts and their relations. Moreover, tool support saves time and reduces therefore the costs of a product development. Manual creation of workproducts such as an item definition or a safety plan are very time-consuming and error-prone tasks because a safety plan contains numerous safety activities that are linked to each other either as inputs or outputs. Therefore, workproducts entail a lot of dependencies and such dependencies make things unclear and complex.

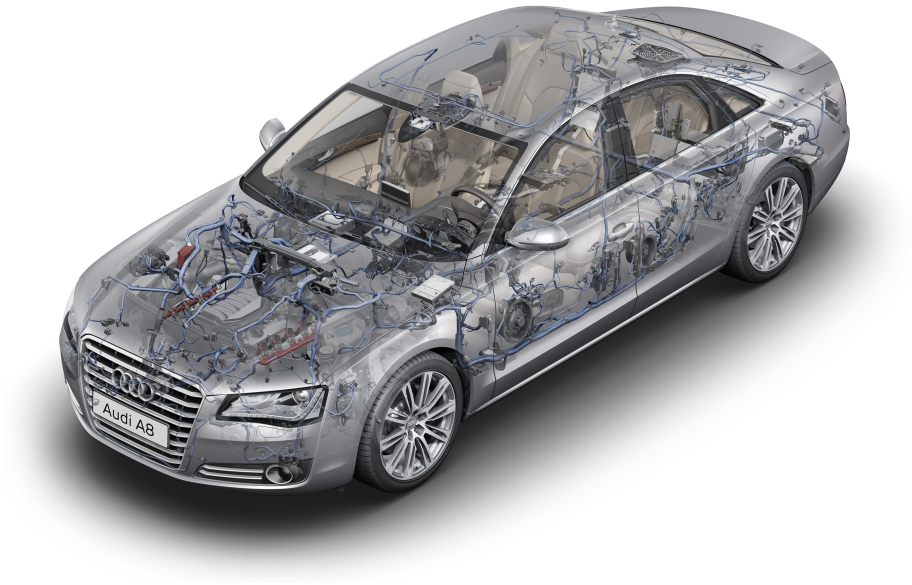

Figure 1.1.: Overview of the high number of control units and their connections in today's cars (Source: VDI[1])

The implementation of an efficient and effective safety management results in a well-defined workflow structure and provides many advantages. As the work in [14] describes, the adoption of a safety process helps reduce costs and enhances the measurement of quality as well as the communication between employees. Furthermore, it improves the performance and the predictability of processes.

1.2. Background and relation to AVL

This thesis is conducted in cooperation with the AVL List GmbH[2], which is the world's largest privately-owned company for development, simulation and testing technology of powertrains (hybrid, combustion engines, transmission, electric drive, batteries and software) for passenger cars, trucks and large engines. In the AVL safety department, a tool environment for creating and maintaining safety management with the associated different safety activities already exists and is based on a spreadsheet application.

According to the huge number of relations between these different safety activities, workproducts and requirements that can be found in the standard, the defined spreadsheet application is already unmanageable. Currently, the existing application contains formulas that have more than 1,000 characters. Therefore, nobody could further maintain and handle this sheet and, thus, a new approach is needed. One other huge problem that occurred by using the spreadsheet application approach is that there is no possibility to store and manage artifact changes within the tool environment. This means there is no

[1] https://www.vdi-wissensforum.de/de/nc/presse/details/article/komplexe-bordnetze-entwickeln/
[2] www.avl.com

database in the background that is responsible for the data storage. By storing a database in the background it is possible to track and audit changes and to open and view old versions of the project. Additionally, the manual generation of customer release documents is a very time-consuming and expensive task because there is currently no automatic conversion from big spreadsheet tables into a report format.

In order to set up a new powerful and effective safety management tool environment, AVL wants to start with a novel tool environment to cope with such time-consuming and error-prone tasks. The aim of this thesis is to better support the development process and to therefore reduce the development time and costs.
In this thesis, a newly developed "Functional Safety Management" tool will be described that allows the creation of different standard-specific and project-specific artifacts. Whereas the standard-specific artifacts describe a generic lifecycle which is described by a standard, the project-specific artifacts describe the independent project-specific lifecycle that inherits from a standard lifecycle.

Hence, a new safety management meta-model shall be created and defined for the tool within a database architecture. This meta-model shall be integrated into this new tool environment and the tool shall be able to plan and maintain the safety management tasks throughout the entire lifecycle. Furthermore, it shall provide features for the automatic creation of different reports as described in 3.4.3.

The main focus within this newly developed tool is on the following parts:

- Safety Management
- Process-Based Safety Case

1.3. Objectives

In this section, the main objectives of this thesis will be depicted. First, the AVL process shall be mapped onto the safety culture process as shown in Figure 1.2. This safety culture process defines the needed activities that shall be performed within a safety management process. Therefore, an AVL functional safety management meta-model shall be created in order to comply with the process. With this meta-model, a novel tool architecture shall be developed that transfers the existing functional safety management spreadsheet application into a database-driven tool environment. By using a database in the back-end, it is possible to store artifacts and their changes during the entire lifecycle. This allows users to save different baselines and to extract and view old versions of artifacts from the database.

An other main objective is that the tool shall allow the creation and maintenance of different projects within an integrated multi-user environment where several people can work on projects concurrently. Furthermore, it shall be possible to store different standards in the database. New projects can be created from these standards and automatically inherit the according defined standard lifecycle. Additionally, it shall be possible to create new projects from other projects and to create sub-projects for existing projects. At last, the tool shall allow the automatic generation of reports using a report generator.

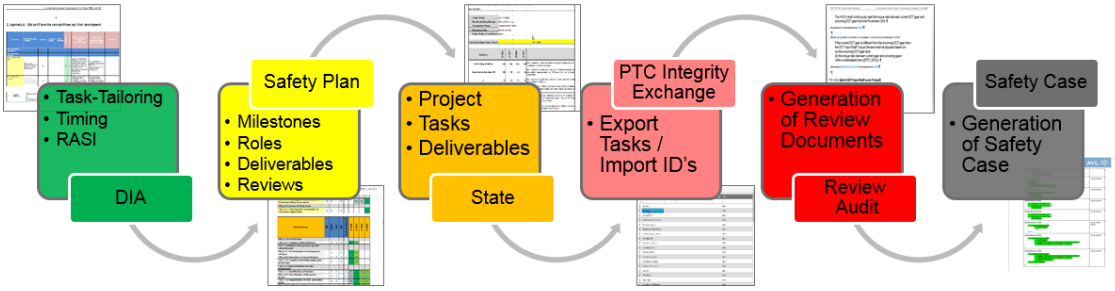

Figure 1.2.: Mapping of AVL and the Safety Culture Process

These reports shall be generated in a semi-automated way, which means that they shall serve as a basis for customer releases or for own reviews and usages. During the concept phase of the tool, the creation of an easy and powerful user interface that has a database in the back-end will be considered.

Additionally to the two defined objectives above, the tool environment shall be based on a client-server architecture which allows the integration of a multi-user approach. Therefore, the tool shall comprise a user management that permits to define different categories of permissions for users.

1.4. Outline of the thesis

The rest of this thesis is organized as follows: Chapter 2 summarizes the results of the literature research regarding collaborative safety management and the process-based safety case generation. Additionally, the ISO26262 standard and the according terms and definitions will be described in more detail. Chapter 3 depicts the defined requirements, the software architecture of the newly developed tool environment and the associated features. Chapter 4 provides an overview of the implementation of the different concepts within the tool environment. Furthermore, the tool usage will be illustrated in more detail. Chapter 5 concludes the thesis and suggests further work.

2. Related Work and Context

2.1. ISO26262 – the Automotive Functional Safety Standard

ISO26262 [9], entitled "Road vehicles – Functional safety", is the official standard of functional safety for the automotive industry. It was created for the development of safety-critical automotive E/E systems and it applies to passenger cars up to 3,500kg [1]. The standard is an adaption of IEC61508 [10], the generic standard for functional safety across all domains, and provides structured workflows during the entire product development cycle, starting with the specification, then continuing with the design of the product, the implementation and integration, the testing, verification and validation and at last ending with the operation after the product release [1]. As shown in Figure 2.1, ISO26262 consists of process-based requirements, product-based requirements and further safety-related workproducts. The work in [15] compares and describes the differences between the two standards. The following key concepts are defined in the ISO26262 standard:

- provides an automotive safety lifecycle during the entire process
- supports the tailoring of safety activities during the safety lifecycle
- provides Automotive Safety Integrity Levels (ASIL A – D) for risk classification
- uses these risk levels for deriving requirements to reduce the risks to an acceptable level
- provides requirements for validation and confirmation measures to achieve an acceptable level of safety

Figure 2.1.: ISO26262: composition of the different parts

This chapter concentrates more on Part 2, “Management of functional safety”, on Part 8, “Supporting Processes”, and on Part 10, “Guideline on ISO26262”, of the standard because these parts contain the main information that is needed for this thesis. In the following, the challenges and implementation of these parts will be discussed in more detail. First, the process model of ISO26262 will be presented and explained. The subsequent section illustrates the important and relevant terms of the standard and gives a short overview of all ten parts of ISO26262.

2.1.1. V-Model

The V-Model describes the overall process model of the functional safety standard and presents the different phases of the standard. Figure 2.2 illustrates the V-Model over all existing parts of ISO26262. As it can be seen, only Part 3 to Part 7 are part of the V-model process. The reason for that is that only these parts describe the design, development and production processes while the other parts cover different things which will further be described in 2.1.3. The left side of the V-Model illustrates the development process and the right side describes the associated verification and validation processes.

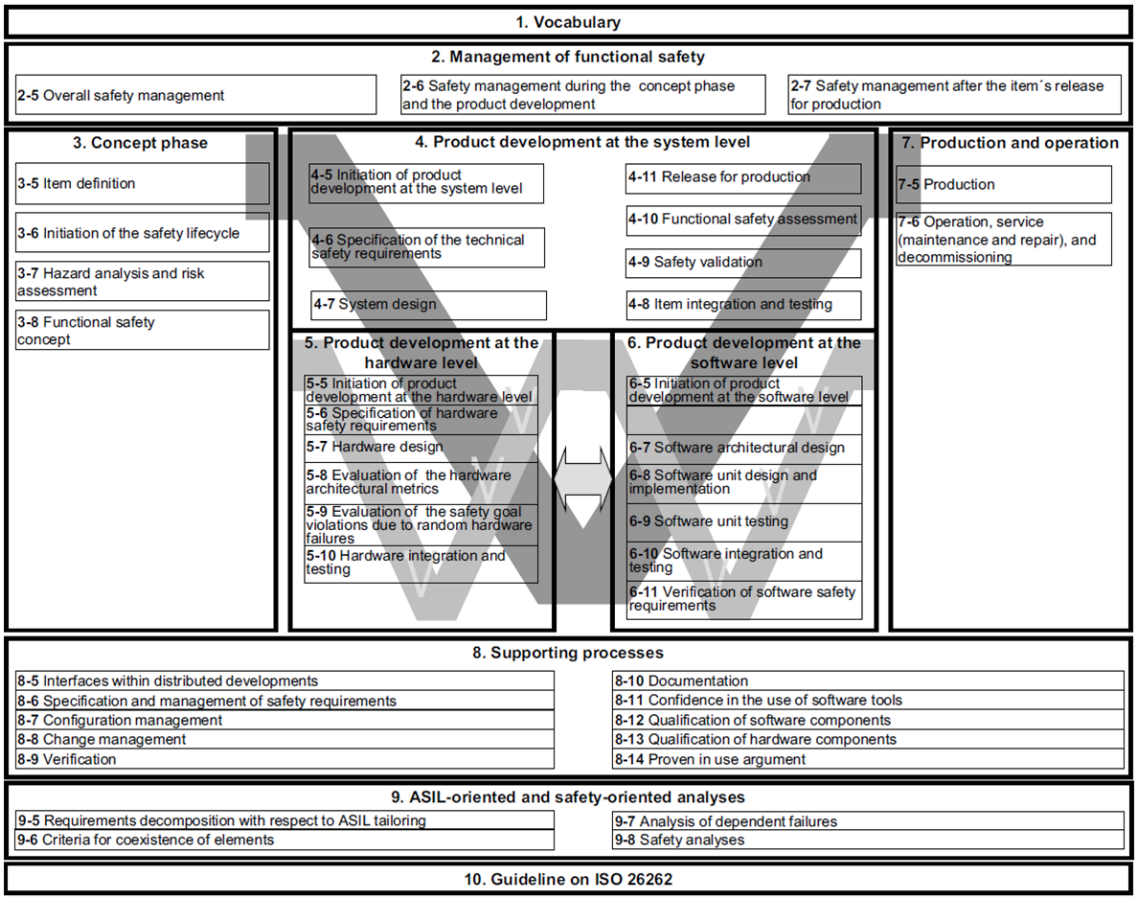

Figure 2.2.: ISO26262-integrated V-Model (Source: [1])

2.1.2. Terms and Definitions

The following parts describe the relevant and necessary terms and definitions of ISO26262 and additional literature. The first part [1] of ISO26262 provides definitions and explanations.

Safety Management

ISO2626 contains a normative Part 2 [2] concerning safety management which defines the overall safety management through the organization. The objective is to define and implement the workproducts and requirements of the safety lifecycle defined by ISO26262. Furthermore, the standard distinguishes between process-based and product-based requirements. Whereas the process-based requirements define the activities which shall be performed in order to fit the safety lifecycle. the product-based requirements define the technical aspects for creating a safe product within the safety lifecycle. Furthermore, the standard defines different roles for safety, like the "Functional Safety Manager (FSM)" or the "Technical Safety Developer (TSD)."

Functional Safety Manager

In Part 2 of the standard, a new role named "Safety Manager" will be introduced. A safety manager is responsible for the planning and coordination of the activities that are part of the entire safety lifecycle. A safety manager is coupled very closely to a project and quality manager. In the following, some main tasks of a safety manager [16] are listed:

- responsible for functional safety within the company/project
- plans and coordinates tasks
- maintains the safety plan and monitor the safety process
- checks safety activities against the safety lifecycle
- delegates tasks to persons
- trains and educates persons
- define reviewers, auditors and assessors
- plans reviews, audits and assessments
- does technical leadership
- checks organization safety process against standard safety process

Safety Culture

The term "Safety Culture" implies the implementation of all needed organization-specific tasks, processes and roles in order to achieve functional safety within the organization. Every organization that performs safety activities for the development of E/E systems should establish a safety culture. For achieving a well-defined safety culture, all team

members should be aware of the defined safety culture while the organization should provide all needed resources in order to establish this safety culture. Furthermore, these tasks and processes (safety lifecycle) should be executed in accordance with ISO26262 and the defined requirements in order to comply with the standard. The definition of safety culture within ISO26262 Part 1 (1.107) [1] is "policy and strategy used within an organization to support the development, production and operation of safety-related systems." The objective of a well-defined safety culture is the achievement of functional safety throughout different projects.

Safety Lifecycle

The safety lifecycle defines all requirements in all different phases that shall be performed in order to complete the safety activities. As shown in Figure 2.3, the lifecycle accounts for the management of functional safety through the concept phase, the product development and the release of production. The main goal is to plan, coordinate and perform the safety activities within this safety lifecycle in order to achieve functional safety.

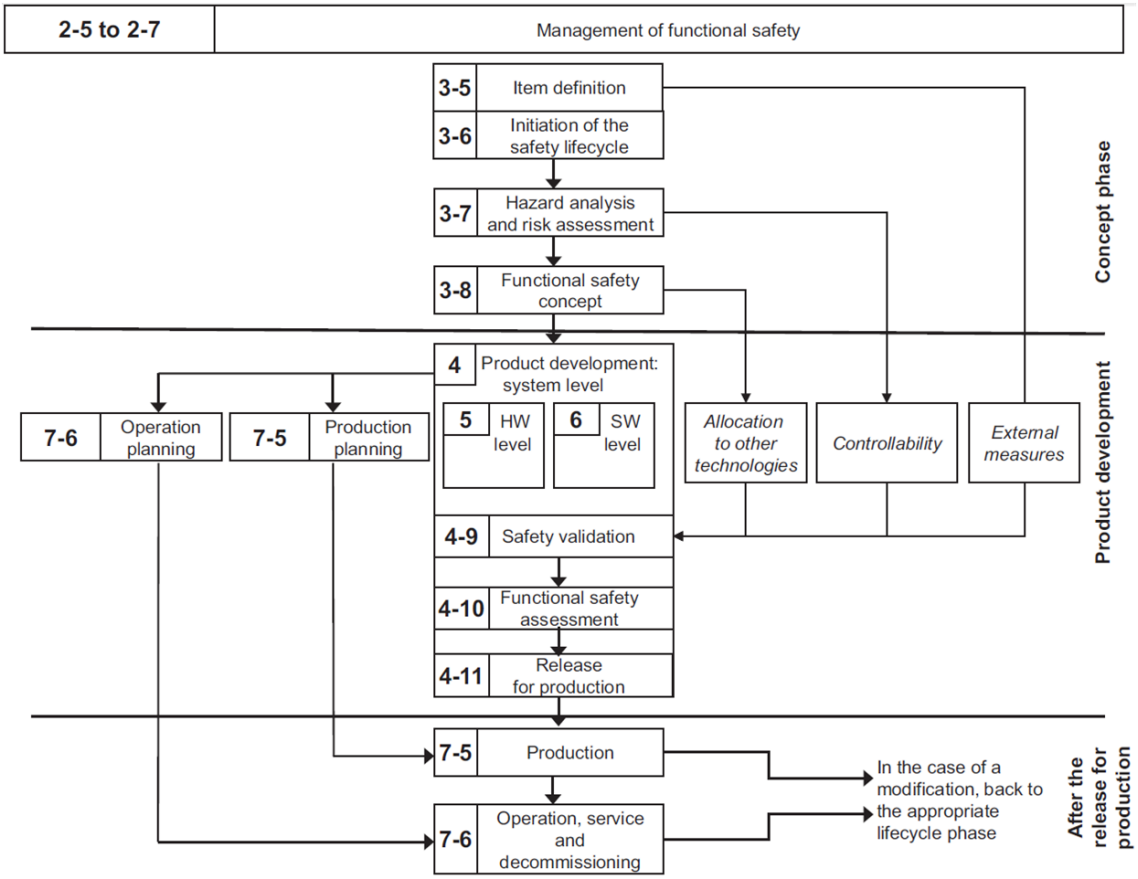

Figure 2.3.: ISO26262-defined Safety Lifecycle from [2]

As described in [2], the tailoring of a safety lifecycle should be performed at the beginning of each project, at the same time as the project plan will be generated. During the tailoring of the safety activities, all tasks that are not necessary for this project are disabled and must be linked with an rationale explaining why it is not needed in this project. As soon as the tailoring process is completed, the safety lifecycle is now ready for a specific project.

Development Interface Agreement

After tailoring of the different safety activities, the "Development Interface Agreement (DIA)" shall be performed. Input of the DIA are the tailored process activities that describe the relevant tasks to be performed within a specific project. Additionally, it determines the responsibilities of customer and suppliers for the different tasks. For this information, a defined "Responsibility Assignment Matrix"[1] is used. In general, there are a lot of different RASI matrices. At AVL, a cooperation partner of this thesis, the below RASI matrix is used.

- **(R) Responsible:**

  This role describes the person that is responsible for executing the safety activity.

- **(A) Approval:**

  This role defines the person that shall approve the implemented safety activity.

- **(S) Support:**

  This role specifies the person that shall support the responsible person during the execution of the safety activity.

- **(I) Information:**

  This role determines who will give only information.

Additionally to the RASI matrix, the development interface agreement contains the according workproducts which are results of the different safety activities, and it further contains and describes all different interfaces (processes, tools, etc.) between the customer and the suppliers. Annex B of Part 8 [12] of ISO26262 depicts an example of such a development interface agreement. The provided example is only for the purpose of illustration and therefore only informative and not normative! Hence, it may be used as a guideline. The description of the development interface agreement in ISO26262 (Part 1 – 1.24) [1] is "agreement between customer and supplier in which the responsibilities for activities, evidence or workproducts to be exchanged by each party are specified."

[1] http://project-management.com/understanding-responsibility-assignment-matrix-raci-matrix/

Safety Plan

Within the safety plan, the performance of the different safety activities of the safety lifecycle are planned and managed in detail. It is a further refinement of the development interface agreement and additionally contains the project milestones until when each activity shall be executed. Furthermore, it shows who is responsible for the different safety activities, what the according tasks for performing these activities and what the resulting deliverables are. At last, also resources for the execution of the safety activities can be defined inside the safety plan. The functional safety manager is responsible for creating and maintaining the safety plan and for checking the status of the safety activities against the safety plan. Another important thing is that the safety plan shall either be integrated into the project plan or shall be referenced in order to align the project plan and safety plan.

Confirmation Measures

A confirmation measure can be confirmation reviews, functional safety audits or functional safety assessments. Each task must be reviewed and evaluated according to the ISO26262 standard. For achieving an acceptable level of safety, the confirmation measures shall be performed. This step is also defined in Part 2 [2] of ISO26262. Furthermore, the execution of verification reviews are needed because they are required by ISO26262. In the following, the three kinds of confirmation measures will further be described.

Confirmation Review

This review checks the compliance of workproducts against the standard and shall be performed at all ASIL levels.

Functional Safety Audit

The functional safety audit is the inspection of all implemented processes with the workproducts and requirements for the achievement of functional safety. This audit shall be done by one or more persons and is only needed for items with an ASIL higher than ASIL (B).

Functional Safety Assessment

A functional safety assessment is the checking of the workproducts defined by the safety plan and furthermore the monitoring of the processes that are necessary for achieving functional safety. Such an assessment plan shall be included in the safety plan and as a result, a report of the assessment shall be generated. This report shall include, as described in [2], a recommendation for acceptance, a conditional acceptance or a rejection of the functional safety of the item. This safety assessment shall only be conducted for items with an ASIL greater than ASIL (B).

Safety Case

The definition of a safety case according to ISO26262 Part 1 [1] is an "argument that the safety requirements for an item are complete and satisfied by evidence compiled from workproducts of the safety activities during development". Basically, in the industry, there are different definitions of a safety case available.

In general the purpose of a safety case is to demonstrate that functional safety was achieved and that the system under development is free from unreasonable risk. Therefore, a safety case is based on an understandable and well-defined argument which is supported by evidence. The evidence can comprise the confirmation that all the workproducts of the safety lifecycle were created and the requirements defined by the applicable safety standard were fulfilled.

Figure 2.4 shows the relationship between the three important artifacts of a safety case. Requirements are the first artifact which are defined by ISO26262. With the artifact number two, the safety argument, a clear and understandable reason shall be given in order to show that the system is free from unreasonable risk. In order to verify these safety arguments, the third artifact, evidence, is needed. For example, a workproduct from ISO26262 can give such an evidence. This entire information should be compiled in a safety case report.

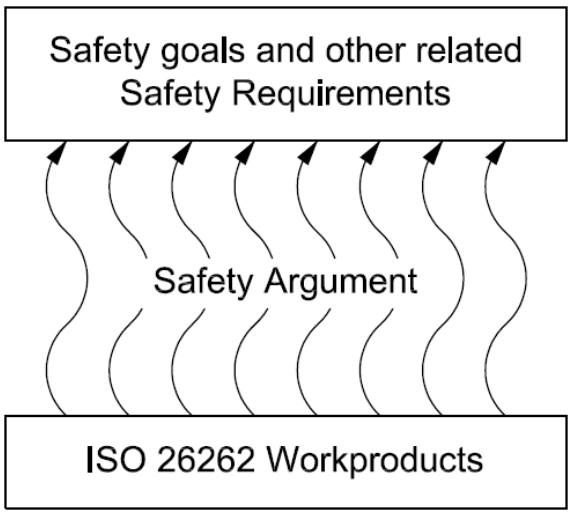

Figure 2.4.: ISO26262 Safety Case Key Elements (Source: [3])

The links between the artifacts like requirements and workproducts should always be provided, either top-down or bottom-up. Additionally, these links should be structured in a clear way and there should be no gaps which could lead to a safety goal violation.

In Figure 2.5, the relations between the different kinds of a safety case are depicted. Whereas the process-based safety case focuses on the safety lifecycle and the associated workproducts the product-based safety case focus on the system architecture and system behavior and is therefore related to the attributes of the system which was developed according to a derived ASIL. ISO26262 provides no normative part for the safety case generation. Moreover, it only gives a simple information guideline defined in Part 10 [17] of the standard.

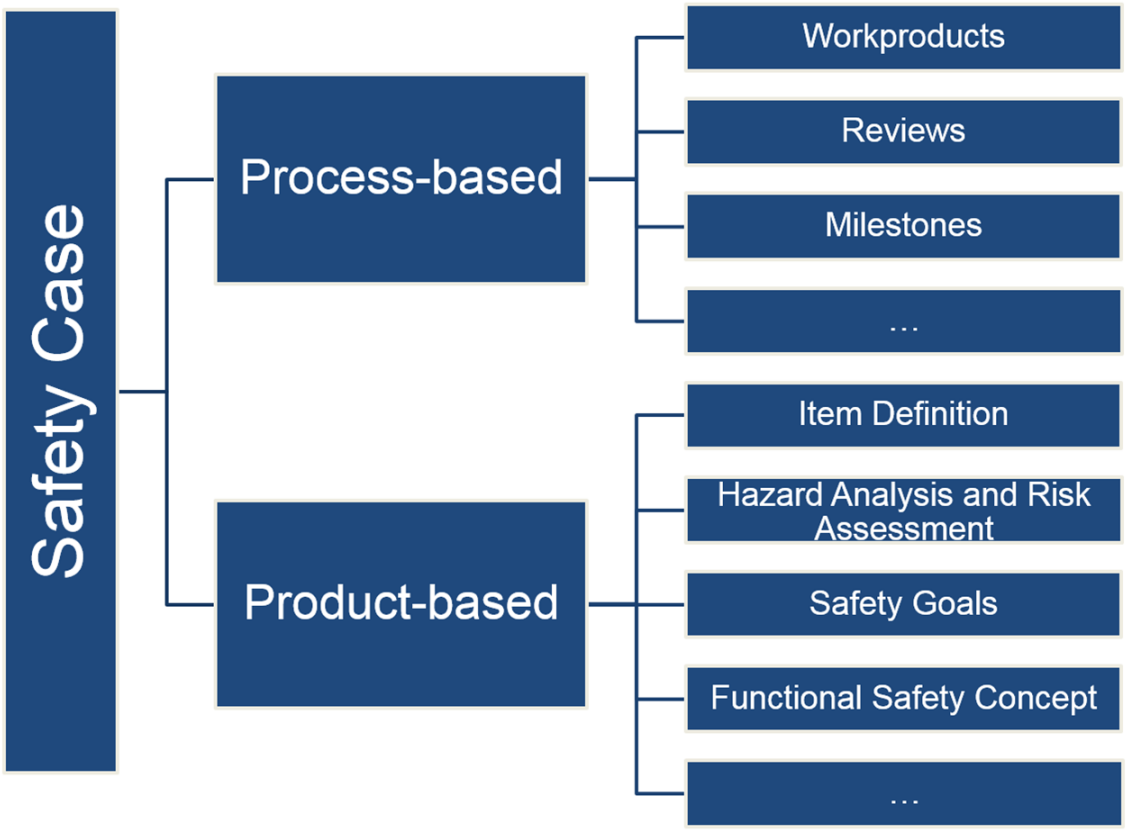

Figure 2.5.: This figure illustrates the possibility to divide a safety case into a process-based and product-based safety case. Furthermore, some example artifacts for each safety case are given

A safety case should be conducted for items that have an ASIL greater than ASIL (A), ASIL B, ASIL C or ASIL D. The brackets in ASIL (A) mean that for this ASIL it is only a recommendation to create a safety case and it is therefore not mandatory. This thesis focuses more an the process-based safety case generation and offers therefore only a short overview of product-based safety cases.

Traditionally, a safety case is mostly only a collection of simple text and therefore really difficult to read and understand. Furthermore, it is not as easy to see how the three aforementioned artifacts are coupled together. The author of [18] has developed an alternative way for demonstrating a clearly arranged way for showing a safety case by providing a specific graphical notation language. Today, this language, named "Goal Structuring Notation (GSN)," is very popular and widely used in different industries. It provides different graphical elements for requirements, claims, evidences, context and many other artifacts. In Section 2.1.2, more information about this goal structuring notation will be given.

GSN – Goal Structuring Notation

According to the huge amount of information that is elaborated within a safety case, the textual representation of such a safety case is very unclear and unmanageable by an individual person. Therefore, a better approach for managing the argumentation in a compact way is needed. For this reason, the "Goals Structuring Notation (GSN)" has been introduced. It is a powerful graphical notation language which contains a set of predefined graphical elements that can be used for conducting safety argument in a clear and manageable way. Today, the GSN is already recognized as a certain standard [19] and widely used in different industries. This section gives a short overview of the GSN standard and associated elements.

According to the work in [20], the goal structuring notation is widely embedded in the industry for creating safety cases. According to the simple graphical elements and their correlation, it is possible to define a safety argumentation less complex and better and easier to understand. The standard provides a set of core elements and modules. The elements are depicted in Figure 2.6 and their associated relations are illustrated in Figure 2.7. An example of a goal structure is given in 2.8.

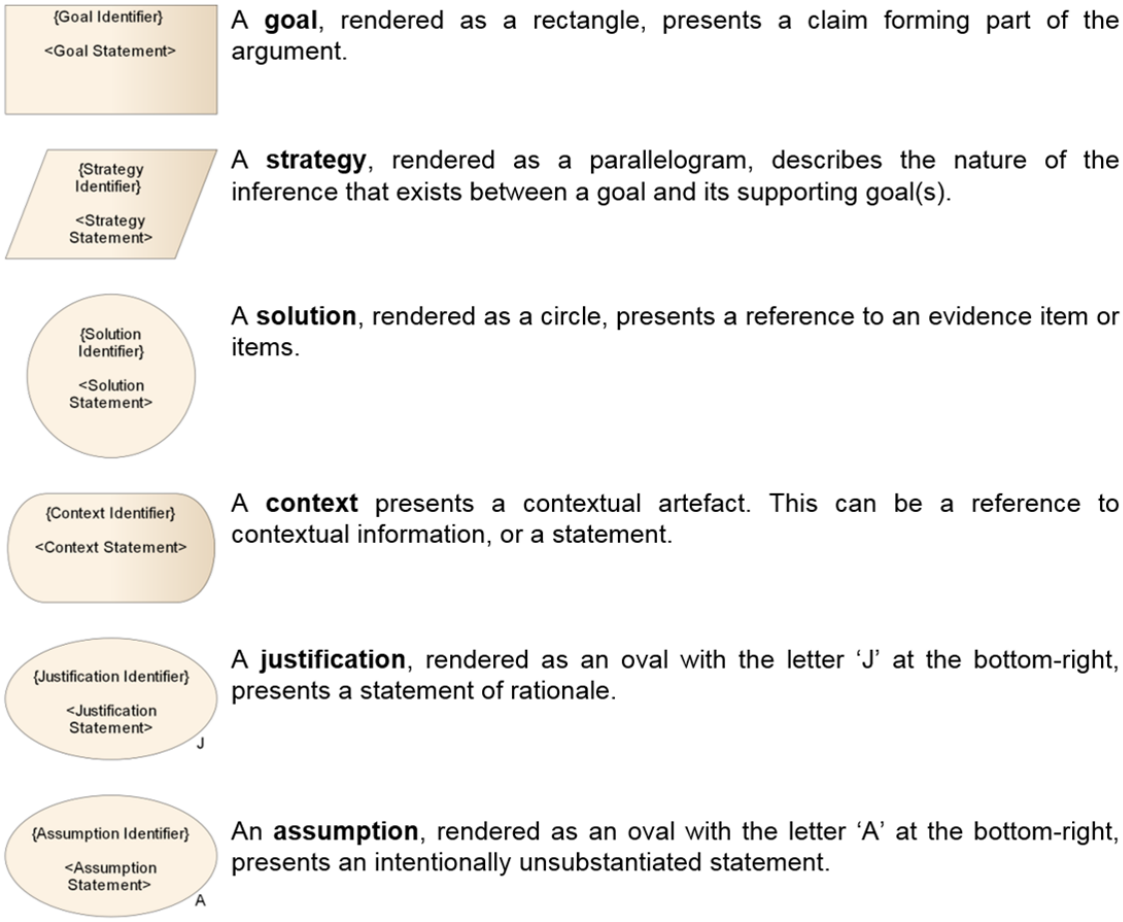

Figure 2.6.: GSN core elements defined in the GSN standard (Source: [4])

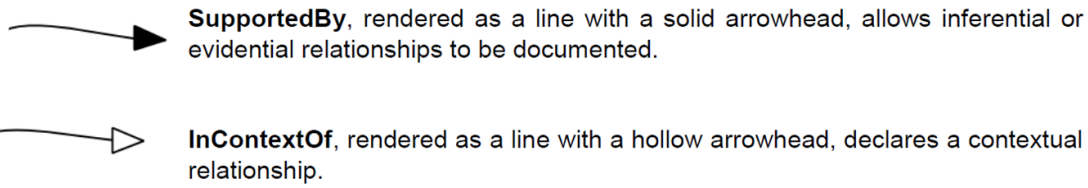

Figure 2.7.: GSN relations for connecting the GSN elements (Source: [4])

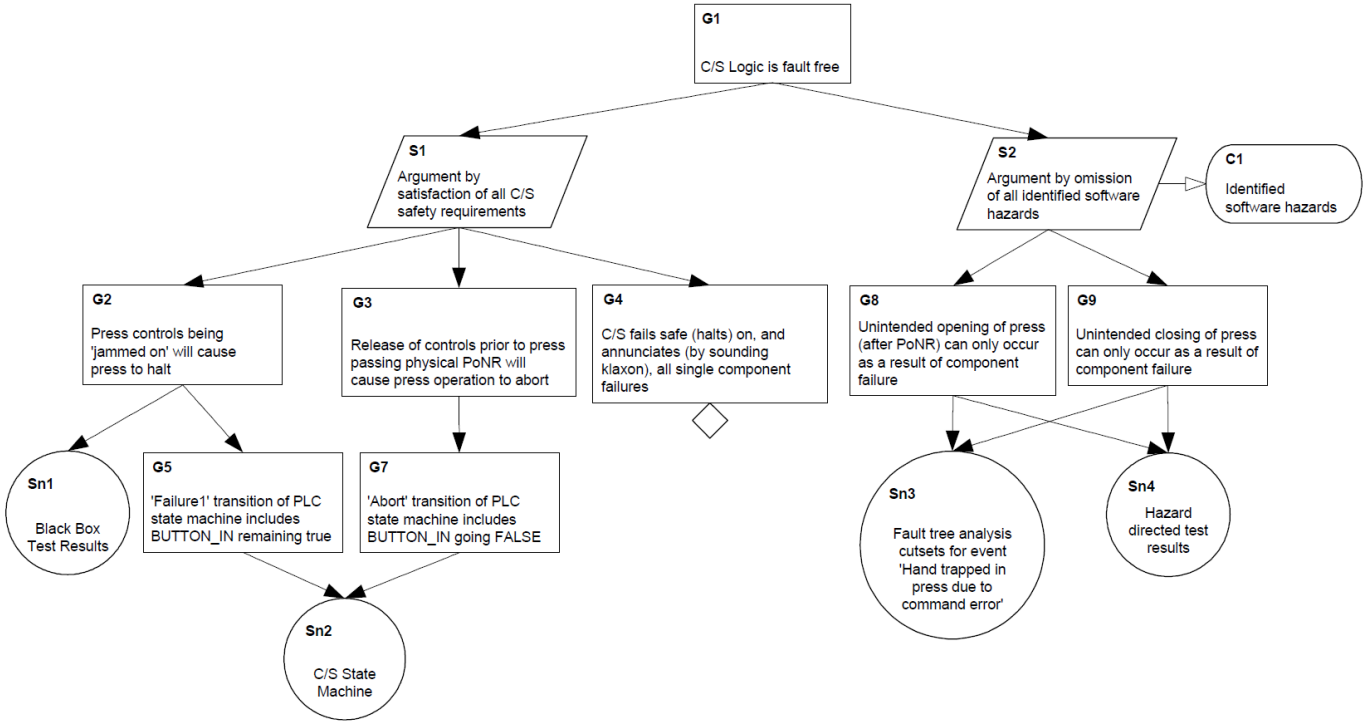

Figure 2.8.: An example of a goal structure (Source: [5])

2.1.3. ISO26262 Structure

The functional safety standard consists of ten parts which will be taken into consideration beneath. Whereas Part 1 through 9 are normative parts, Part 10 is only an informative part and can be used like a guidebook.

Part 1: Vocabulary

Part 1 [1] of the standard defines the important terms and definitions of functional safety. These terms and definitions are valid within all different parts of the standard and are very important because terms usually have different meanings in different standards.

Part 2: Management of functional safety

Part 2 [2] covers the process for the management part of the functional safety standard. On the one hand, this part defines process-related requirements, which involves the organization part, and one the other hand, it defines product-related requirements which are related to the execution of safety activities of the safety lifecycle (2.3).

This safety lifecycle starts with the safety management, covers the concept phase, development phases in system, hardware and software level and concludes with the release of the production phase. Furthermore, this part describes the necessary requirements to set up a safety culture for developing safety-critical systems according to "Automotive Safety Integrity Levels (ASIL)" and to develop a system that is sufficiently safe. Therefore, the objective of this part is to achieve functional safety throughout the organization and the entire product lifecycle.

Part 3: Concept phase

The third part [6] of the standard deals with the concept phase of ISO26262 and starts with the item definition. The item definition is the first task of the safety lifecycle which will be defined in Part 2, and the objective is to define and describe the item as well as the interfaces to other items or the environment. Therefore, a set of non-functional requirements which describe the item with its dependencies and interfaces and the boundary of the item, shall be the output of the item definition. It is important to entirely understand the item in order to perform the subsequent tasks. One important thing to mention is that this part only considers the functional aspect of the system.

The next step after the item definition is the hazard analysis and risk assessment (HARA). In this task, all possible malfunctions will be determined and the according hazards will be defined. In a further step, these hazards will be combined with operational situations (e.g city driving - rolling towards a red traffic light with a pedestrian crossing). Operational situations include the driving situation, environmental conditions and involved parties. Combining hazards with different situations results in the creation of hazardous events. After the definition of all possible hazardous events, they can be categorized according to severity (0-3), exposure (0-4) and controllability (0-3). With this safety categorization, it is possible to determine the ASIL from the ASIL table displayed in Figure 2.9.

After the determination of the ASIL, safety goals shall be derived for hazardous events with an ASIL A, ASIL B, ASIL C or ASIL D. A safety goal is a top-level requirement that must be achieved in order to avoid unreasonable risk. If the result contains a high number of safety goals, similar safety goals have to be combined as one safety goal. Additionally, more than one hazardous event can be linked to a safety goal and one safety goal can be linked to several hazardous events. If several hazardous events are linked to a safety goal, the safety goal always inherits the highest ASIL of all linked hazardous events.

Table 4 — ASIL determination

Severity class	Probability class	Controllability class		
		C1	C2	C3
S1	E1	QM	QM	QM
	E2	QM	QM	QM
	E3	QM	QM	A
	E4	QM	A	B
S2	E1	QM	QM	QM
	E2	QM	QM	A
	E3	QM	A	B
	E4	A	B	C
S3	E1	QM	QM	A
	E2	QM	A	B
	E3	A	B	C
	E4	B	C	D

Figure 2.9.: ISO26262-defined ASIL classification table corresponding to [6]

When the safety goals are determined, the next task, the "Functional Safety Concept (FSC)," can be performed. This task contains the derivation of functional requirements for each safety goal and the allocation to the respective components or elements. The process of the aforementioned task is illustrated in Figure 2.10. The functional safety requirements contain the safety measures which shall be implemented for preventing safety goal violations.

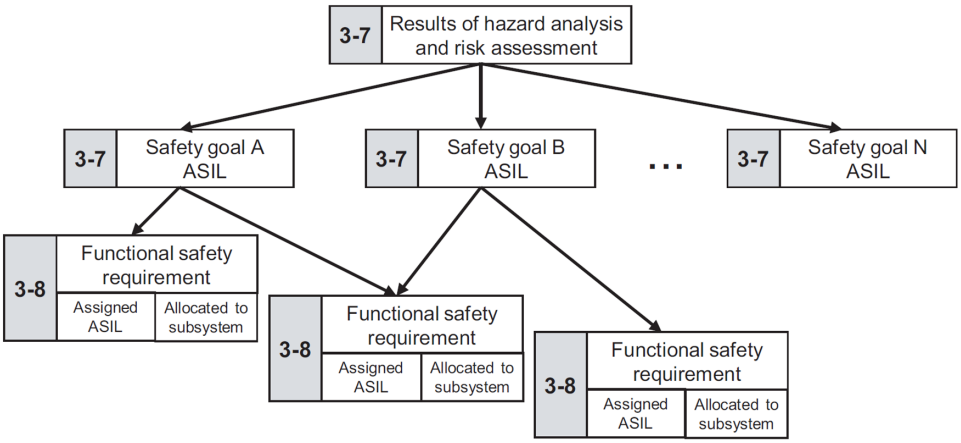

Figure 2.10.: Overview of the Functional Safety Concept process corresponding to [6]

Part 4: Product development at the system level

The next part of ISO26262 is the product development at system level. This part, Part 4 [21], contains the following activities:

- identifiying requirements for the initiation of product development at system level
- specification of the technical safety requirements
- creation of the technical safety concept
- system design with Hardware-Software-Interface (HSI) integration
- item integration and testing
- functional safety assessment
- product release

At the beginning of this part, the different safety activities of the safety plan shall be identified and planned according to the provided safety lifecycle. Additionally, the validation and integration plan shall be added to the defined safety plan. Hence, the next step is to derive technical safety requirements from the functional safety requirements. These technical safety requirements contain for example information about how a safe sate can be achieved during the occurrence of a fault or how such a fault can be detected and mitigated inside a specific system. The set of these technical safety requirements is contained in the "Technical Safety Concept (TSC)."

After the creation of the TSC, the system architecture shall be designed and the according technical safety requirements shall be allocated to these hardware and software systems. For further development, the "Hardware-Software-Interface (HSI)" is created in order to specify the interaction between the different hardware and software components. After the product development of the hardware and software level, the item can be integrated into the system and shall be validated in order to show that the safety goals from the hazard analysis and risk assessment have been fulfilled. The product development on hardware level and product development on software level will further be described in Part 5 (2.1.3) and Part 6 (2.1.3) of ISO26262. The functional safety assessment is an additional task to perform the check whether the system is free from unreasonable risk. The final task after the functional safety assessment is the product release and further decommissioning.

Part 5: Product development at the hardware level

The main focus in this part [22] is on the hardware level. Therefore, hardware safety requirements shall be identified and the according hardware architecture shall be designed. Different methods for the verification of the hardware system with the according ASIL levels are listed within this standard part. For the verification process, different metrics for the evaluation of the hardware architecture are provided in order to show that a specific ASIL can be achieved. These metrics contain the single-point fault metric, latent fault metric, the diagnostic coverage and some others. As, they are very technical metrics, they

will not further be covered in this thesis. More information on these metrics can be found in Part 5 section 8.4 ([22]) of ISO26262. The other chapters of this Part 5 contain the integration of the hardware system and the according validation and verification.

Part 6: Product development at the software level

Part 6 [23] describes the software level of the product lifecycle. Similar to Part 5, the software safety requirements will be specified and the according software architecture will be designed. Just like the hardware safety requirements, the software safety requirements will be derived from the technical safety requirements. The remaining sections of this part covers the software implementation, provides information about the different software unit testing methods and describes the integration of the software components into the system with the associated software verification.

Part 7: Production and operation

Part 7 [24] covers the production task for safety-critical items that shall be integrated into road vehicles. Furthermore, it also covers the operation, the service, which contains the maintenance and repair work, and at last the decommissioning. First, the safety plan shall be adapted with the according production plan that contains the related safety activities for this part. A further goal is to achieve functional safety during the entire process as mentioned above.

Part 8: Supporting processes

The supporting processes part [12] of ISO26262 contains the process description for the interfaces between customer and suppliers. A workproduct of the first safety activity is for example the development interface agreement which is further described in 2.1.2. This safety activity defines the different task responsibilities and processes between the customer and the supplier during the development process.

The objective of the safety management activities is to create consistency and efficiency throughout the entire safety lifecycle defined by ISO26262. Therefore, Part 8 defines the management of safety requirements as a process of managing requirements, obtaining agreements on these requirements, obtaining commitments from those implementing these requirements and maintaining traceability.

Furthermore, this part contains processes for the configuration and change management. Therefore it deals with the question how to ensure workproducts and with the changes of the safety-related workproducts throughout the entire safety-lifecycle. Moreover, the standard part contains the process description for the verification and the documentation and the process description for tool qualification in order to gain people's confidence in the usage of such tools.

Part 9: Automotive Safety Integrity Level (ASIL)-oriented and safety-oriented analyses

The goal of Part 9 [7] is to reduce the derived ASIL by the usage of ASIL decomposition and therefore by integrating redundancy into the system. This part further contains requirements and rules for this decomposition process with the according descriptions and methods. The ASIL decomposition process is illustrated in Figure 2.11. Additionally, Part 9 also covers the different safety analyses methods in order to detect and investigate possible faults/failures.

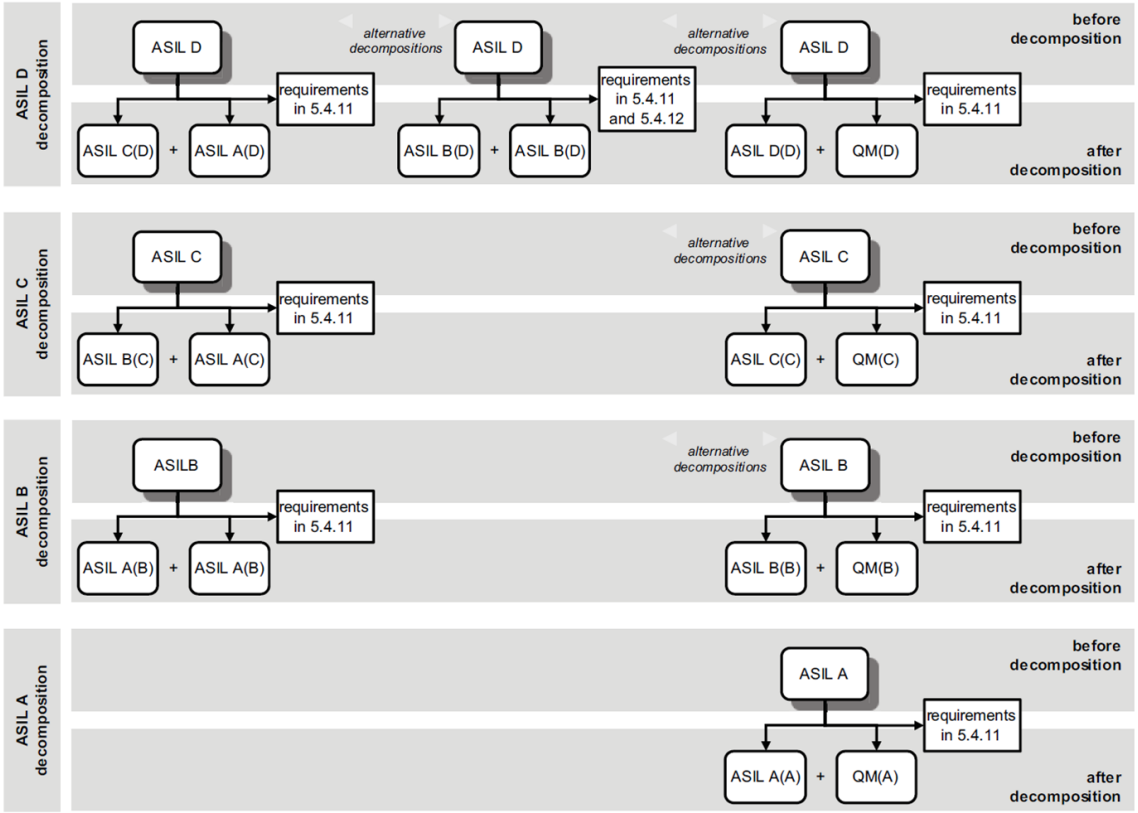

Figure 2.11.: The different rules for the ASIL decomposition (Source: [7])

Part 10: Guideline on ISO 26262

The last part [17] of the standard is an informative part and not a normative part. It covers some examples according to the standard and shows further instructions. Therefore, this part can be used as a guidebook for usage of ISO26262-related processes. Moreover, it covers the relation between ISO26262 and IEC61508 and provides information about safety management and safety case generation. The organization and interaction between the different parts is illustrated in Figure 2.2.

2.2. Related Work

In the following section, approaches for integrating and establishing effective safety management within the organization and possibilities of how to argument these processes in a structured and acceptable way will be reviewed. The first subsection shows attempts for implementing safety management throughout the entire safety lifecycle defined by ISO26262. Additionally, frameworks for these implementations will be depicted. The second subsection gives an overview of approaches which will be part of a process-based safety case generation.

2.2.1. Safety Management

The work analyzed in [13] explains that the ISO26262 standard provides a shift from a "Quality Management System (QMS)" to a new safety-oriented working culture. QMS describes production control-oriented development processes which only address if the product is right. Whereas QMS focuses more on the right product, the ISO26262 process focuses on developing the right product. For this shift, ISO26262 determines development process requirements that shall support the implementation of an effective safety management throughout the different departments within the company.

In order to accomplish this shift, the defined work mentions four barriers for implementing ISO26262. The first one is that business decisions are based on costs, which means that on the one hand, companies shall save costs, and on the other hand, they shall maintain the same product quality during the development cycle. Therefore, by implementing functional safety, the management will be disputed, because safety management brings a lot of benefits but needs more money and time for its implementation.

Moreover, a badly established safety culture takes a lot of time and leads to a high number of failures during the development lifecycle. To deal with this problem, ISO26262 provides workproducts that can be classified according to (1) project-dependent and (2) process-dependent workproducts. To overcome this first barrier, these workproducts must be implemented effectively and must be understood by all developers.

The next issue is that in a normal working culture the end product can be retrieved without defining any requirements. That means that QMS does not cover the left side of the V-model, which describes the requirement definitions. The V-Model can simply be described as a product assurance plan that provides a guideline for the product development lifecycle. Furthermore, ISO26262 defines coordinated workproducts which are linked in a consecutive way. The result of a workproduct serves in most cases as an input for the next workproduct.

The third problem is the gap between qualitative and quantitative product reliability assessment. Whereas QMS can only improve the quality of a product by conducting a qualitative FMEA (Failure Mode and Effects Analysis), ISO26262 has also defined a quantitative FMEA method for the evaluation of the different FIT (Failure in Time) rates of the components. These FIT rates are part of the safety metrics which are defined in Part 5 [22] of the standard.

The last issue is to establish and manage the safety confirmation measures. ISO26262 introduces a new role named safety manager which is responsible for planning, delegating

and tailoring different safety activities, creating and monitoring the safety plan, maintaining the safety lifecycle, verifying the safety assessment plan and writing the safety case for the process and product confirmation.

The goal of the automotive industry is to overcome these obstacles and to shift the development process from a qualitative way to a quantitative working culture. While ISO26262 has already taken the first three steps, it does not define the way how distributed developers shall work together. This means that there is a gap between the different developers and their particular working culture.

At last, the paper proposes "Safety Management Review" network for ISO26262. This network already exists in the aerospace and aviation industry and shall (1) close the gap between different departments, (2) shall find safety anomalies early in the product lifecycle and (3) shall establish an effective safety management based on field data.

[25] presents an enhancement of the CESAR safety framework [26] for supporting the different phases during the development of customer projects according to ISO26262 and a new methodology for the documentation of this development process. The different phases are (1) definition, (2) management, (3) monitoring and (4) the validation of the project according to the development process described by the standard. One of the main problems during the development of automotive embedded systems is that this process is mostly distributed over different organizations. Although ISO26262 provides a detailed development process and supports these different methods, it is a challenge, because using these methods creates up to 100 workproducts that have to comply with over 1,000 requirements in order to fit the standard.

Additionally, the paper describes three different challenges for the management of functional safety according to ISO26262. The first challenge is the planning of the different safety activities during the project planning which shall be conducted in parallel to the project plan. Identifying and well-considered planning of all safety activities establishes a good working culture amongst different organizations and groups. The planned activities result in the safety plan.

The second challenge concerns the provision of guidance of safety experts throughout the project development phase. Efficient exchange of information is necessary for retrieving all relevant requirements for defining the activities.

The last challenge is providing confirmation of the processed safety activities according to ISO26262. An important step in this challenge is the mapping between the workproducts defined by ISO26262 and the company specific deliverables. At the end, an evidence for each implemented requirement shall be provided.

In addition, the work presents a tool enhancement for Microsoft Office Excel. This enhancement contains the development interface agreement (DIA), the safety plan and the allocation of recommendations for one or more workproducts. DIA is the first document created during project planning and contains all tailored safety activities and the according responsibilities of the customer and the suppliers. A detailed planning of each safety activity is included in the safety plan. The allocation of the recommendations combine the requirements with the different workproducts and provide information about the creation of these specific workproducts. As depicted in the work above the recommendations can be "Input", "Output", "Refined" or "Link". With this information, the developer gains an overview of the methods that shall be used. Additionally, this information can

be used as a checklist for the correct execution of the safety activities defined by ISO26262.

In [27], an approach for integrating a graphical approach for functional safety management (FSM) is given. Although the work describes a different standard than ISO26262, namely IEC61511 [28], which is the functional safety standard for safety instrumented systems for the process industry sector, the safety management approach is very similar to the ISO26262 approach. Both standards inherit from IEC61508, the main standard for functional safety over all different domains. IEC61511 also contains a safety lifecycle that covers workproducts and requirements.

The authors explain the usage of a graphical tool that can be used right from the beginning of a project in order to gain benefits like multi level access, automatic report generation and remote review and control. The integration of functional safety management into a company is a very time-consuming and challenging task. If it is not integrated into the overall process landscape, which is very costly, the company faces just additional costs but no benefits. As a matter of fact, the paper points out that three of the top five main oil and gas companies do not have a functional safety management strategy.

First, the authors describe an approach with which an organization chart shall be created within the proposed tool in order to have a general overview of all involved employees and their skills and roles. This can be done by simple dragging and dropping boxes from the toolbox into the graphical view. In addition, it is possible to add some resources, like files, pictures and other artifacts. Furthermore, the authors also mention that a software tool shall be very intuitive in order to have the focus on the definition and creation process of functional safety management.

The used tool allows the definition of standard-specific and company-specific activities and the linkage of those artifacts. By using the approach of combining boxes that represent safety activities, a safety plan and a verification plan can be created. For this verification, the tool provides bullet points which are connected to defined reviewers and can automatically be reported by the tool. This integrated notification mechanism shall ensure a fault-free execution of the different process-specific tasks. As every person has access to the tool, they always get to know the current state and gain an overview of the project status.

Although the tool shows a lot of drawbacks, like the not user-friendly graphical user interface and the poor visualization of the boxes and connections, it is a way towards an electronic and collaborative way of working and provides more benefits than the usage of natural based paper environments. The conclusion of the authors is that the usage of such a graphical tool is an innovative way for collaborative safety management and saves time and extra costs.

A modeling approach for the definition and formalization of development process is depicted in [14]. The authors of the proposed work describe a process modeling approach which is based on SPEM (Software and Systems Process Engineering Meta-model) that is created by OMG [29]. For the appliance of the SPEM approach, an Eclipse-based library named EPF (Eclipse Process Framework Composer) is used. This library is grounded on the CESAR [26] project and provides a modeling framework for the planning and definition of workproducts, roles and tasks defined by ISO26262.

The work in [27] additionally highlights that the workproducts are strongly coupled.

This input/output relation can be modeled within the provided library. The users get always a consistent and complete overview of the project and the associated workproducts. The proposed library has a powerful plugin mechanism where new plugins can be integrated easily.

The authors have already developed plugins for the different method and task configurations. Hence, it is possible to integrate only the needed plugins according to the project type. If the project focuses on a hardware system, only the necessary plugins may be included. Thereby, all needed task and process elements will automatically be integrated into the project. In the aforementioned example, the software plugin can now be excluded.

Moreover, the work describes an associated methodology that contains a tailoring approach for the different tasks. The authors describe the term tailoring as an adaption of the development process to the project-specific process. The plugin mechanism mentioned above can be used as first tailoring process during the project set up. The tool already includes ASIL packages that contain the associated methods and the linked recommendations. Finally, the library contains a view mechanism that allows a graphical representation of the workflows and all available methods. The contribution of the work is on the one hand the modeling approach for the tasks, roles and workproducts and on the other hand the detailed description of the workflows and integrated safety activities.

In contrast to the works in [13], [25], [27] and [14], the work described in this thesis focuses on the development of a collaborative software tool that provides support for establishing consistent and complete safety management throughout the entire safety lifecycle by allowing users to create and store different standards within a software tool. Furthermore, it is possible to create projects from standards that inherits the associated standard lifecycle and store the data in a provided database. The inherited standard safety lifecycle contains the safety activities, workproducts, methods and requirements.

2.2.2. Process-based Safety Case

Within this section, papers about processed-based safety case generation according to ISO26262 are presented. In general two different safety cases can be distinguished. The first one is the process-based approach and the second one is the product-based safety case approach. In this thesis, the focus is only on the process-based safety case generation, which is why some existing approaches will be described.

The author in [30] focuses on a process-based argument safety case and on the question of how to generate such process-based arguments directly from process structures. A process-based argument verifies that a process has been applied correctly to fit the standard. The second major contribution is that the work concentrates on a reuse approach of such process-based arguments in order to compile the information in an understandable manner.

The work presents a model-driven method where process-based arguments can (semi-automatically) be derived from process models. Hence, this approach minimizes the work and thus saves time. Additionally, for modeling static processes the authors use the language SPEM. They also use a model-driven approach and a goal structuring notation (GSN). A safety case as described in this work consists of two main parts.

The first one is the process part that describes the safety lifecycle and the associated

workproducts. The product part describes the safety mechanisms implemented in the product.

The second one is the product part which describes the product development according to a derived ASIL level. Both parts shall be developed at the same time because they have interferences. Basically, the system is only safe if the process and the product are appropriate.

In ISO26262, the process-based arguments shall be provided within the functional safety audit. The provided approach for model-driven safety certification starts with the process modeling. After the definition of the process, the process-based arguments are generated with an transformation engine according to the standard. Now, this generated process-based argument will be checked by an expert, and if there remain problems, a new iteration will be started. The work also points out that human knowledge of safety argumentation is always needed. The framework shall only provide support for the creation and formalization of the arguments and cannot conduct the thing autonomously.

Within the approach, the author convert the process artifacts like the tasks, roles and workproducts into a subset of goal structuring elements that can be linked to each other. The author additionally provides a model-driven solution for semi-automated process-based safety argumentation according to a defined set of rules and the appropriate transformation engine.

In [31], an overview of functional safety compliance and assurance with safety cases is presented. Within the functional safety standard of the automotive industry, there is an explicit requirement (6.4.6.2) defined in Part 2, “Management of functional safety” for creating a safety case.

A safety case is a way to show that all workproducts and associated requirements have been fulfilled by linking requirements and evidences with so-called safety arguments. Whereas ISO26262 provides no requirements for the way how to conduct a safety case, it provides only further explanations and recommendations in the informative Part 10 of the standard. This was further described in Section 2.1.2.

A product-based safety case is required for an item which has a safety goal with an assigned ASIL (Automotive Safety Integrity Level) higher than ASIL A. As mentioned above, the requirement 6.4.6.2 of ISO26262 is basically a requirement for the process-based safety case that shall ensure that all processes are performed and fulfilled correctly.

The main elements of such a safety case are requirement, evidence, argument and associated context. The requirement defines the needed safety functionality and objectives to ensure safety. The second main element provides information from studies or different analyses. To be compliant with the requirements, the argument addresses the underlying evidences, and the last element provides information on the argument basis. For conducting a safety case in the automotive industry, the work presents an approach with the goal structuring notation language. It is a graphical notation language for presenting safety arguments. More information on the goal structuring notation can be found in 2.1.2. Within the goal structuring notation, several basic elements exist that contain the aforementioned main elements. The basic elements of the goal structuring notation can be linked by two different relations. The first one is the “supported by” relation and the second one is the “in context of” relation. Both relations were shown in Figure 2.7. More detailed information can be found in the standard [32].

Furthermore, the paper uses this graphical approach in an architectural framework that

provides a safety case for an automotive product. In this framework, product-based and process-based safety cases are closely coupled together. For the product argument, an evidence is needed and because also the evidence must be trusted, a process argument is needed as well in order to provide a trustworthy evidence.

As an example, the paper illustrates the scope of a safety management argument which contains the workproducts (1) safety plan, (2) project plan, (3) safety case and (4) confirmation measures. Therefore, these workproducts are process-related and not directly related to the product safety.

Within this thesis, the developed tool provides a mechanism where the entire safety lifecycle can be defined and maintained within the tool. Therefore, the relations between the different artifacts are defined and stored using the tool. The tool allows the basic generation of a process-based safety case. In this approach, the safety case contains the different arguments and shows the traceability of these elements. The result is a clear overview which shows which of the required safety workproducts are already available and which are not.

2.3. Existing Functional Safety Management Tools

There are already tools available which enable performing safety management and all modules needed by ISO26262. In the following, two of those tools will be listed and described in short.

2.3.1. ENCO Safety Office

The ENCO Safety Office[2] tool is one of the already existing safety management tools. It is a collaborative safety tool that provides a lot of different technologies for different system analyses (FMEA, FTA), requirements engineering, safety plan generation and review handling. All provided features can also be used as a standalone tool and are therefore independent. They provide interfaces for a popular requirements management system, namely the PTC Integrity tool[3]. Regarding safety management, they offer the creation of a safety plan with associated milestones. Furthermore, they provide interfaces for different external task management tools. A report generator is also integrated into the tool.

[2] http://www.enco-software.com/
[3] http://de.ptc.com/product/integrity

2.3.2. Vector PREEVisison

One of the existing major tools for safety management and ISO26262-related issues is the Vector PREEVisison[4] tool. This tool provides support for requirements engineering, designing of architectures, integration of software development processes and many other safety-related workflows. The Vector company has integrated all major key workproducts that are required by ISO26262. The key workproducts are listed beneath.

1. Item Definition
2. Hazard Analysis and Risk Assessment
3. Functional Safety Concept
4. Technical Safety Concept
5. Hardware-Software Interface
6. Qualitative and Quantitative Safety Analysis
7. Verification and Validation
8. Safety Case

All these different areas are integrated into the tool. The tool has additional interfaces to a lot of external tools, like Microsoft Office Excel. Because the company does not provide an evaluation license, the tool could not be evaluated. Vector only provides trial licenses which are quite costly. Therefore, only a short overview and no evaluation of this tool is available.

[4] http://vector.com/vi_preevision_de.html

3. Design of the Functional Safety Management Tool

This chapter provides a detailed overview of the defined tool requirements and the conceptual formulation of the newly developed safety management tool. The goal of this tool is to offer a simple and powerful graphical user interface for creating and maintaining standards and projects with their associated safety lifecycles at the AVL[1] company. Therefore, the tool shall support consistency and completeness in spite of the huge number of different artifacts provided within a specific lifecycle. Furthermore, it shall be possible to extract these artifacts by an available report generator (3.4.3). In addition, a database for storing the different artifacts shall be introduced. At the beginning of this chapter, the main requirements and the overview of the defined tasks for completing this thesis will be listed and explained. The subsequent sections will give an overview of the commonly used technical concepts. Furthermore, the tool-specific features and integrated concepts will be demonstrated in detail. Finally, the chapter concludes by showing the tool and software architecture of the collaborative safety management tool and the consisting libraries. The work in [33] describes the detailed software architecture of the developed functional safety management tool.

3.1. Requirements

In cooperation with the AVL List GmbH, a set of main requirements are prepared during the tool design phase. These main requirements define the features and concept that shall be integrated into the newly developed tool. For a better overview of these requirements, in this thesis, they will be subdivided into user requirements, database requirements and further tool-specific requirements. Figure 3.1 shows this partition.

The user requirements define the features that shall be implemented for the tool usage. Database requirements inherit from these defined user requirements and define a set of database-specific requirements and the associated database architecture. This database architecture represents the meta-model of the developed tool environment. Further requirements that are needed for the concept of the tool architecture are grouped as tool-specific requirements.

The goal of this tool is to integrate a newly developed safety management tool architecture into the AVL infrastructure for supporting safety engineers and safety managers with consistence and complete workflows during their daily work. Additionally, a semi-automatic generation of reports shall support engineers by avoiding the manual creation of specific release documents. Creating release documents is always a very time-consuming

[1] www.avl.com

and laborious task by bringing the data from different sources together. All further tool objectives were listed above in Section 1.3.

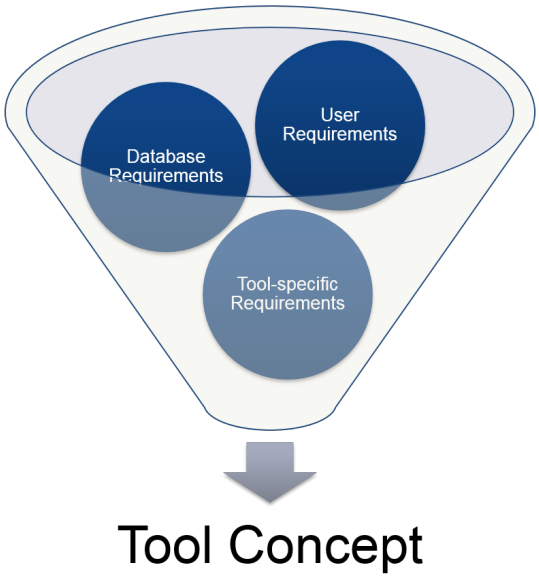

Figure 3.1.: The different requirement sets of the Functional Safety Management Tool concept

The created company specific user requirements document by AVL can be found in Appendix A.1.

3.1.1. User Requirements

- RQ: A simple login mechanism shall be provided.
- RQ: The tool shall provide a simple user management. For the tool usage and the appropriate features, the following three different roles shall be defined.
  1. Global Admin - Is responsible for creating and maintaining the generic artifacts and for attaching access rights to new users.
  2. Project Admin - Is responsible for creating and maintaining projects and can delegate the different tasks to users.
  3. Regular User - Is able to open/change projects depending on their access privileges (read, write).
- RQ: The tool shall allow to describe and maintain the safety lifecycle defined by specific standards. The safety lifecycle can be defined by the following structure:
  1. Cluster (e.g. Part 2: Management of functional safety) - Group of Activities in order to create a simpler structure
  2. Activity (e.g. Overall Safety Management) - Group of mate Tasks

3. Task (e.g. Establish Safety Culture) - A basic unit that can be performed within a process

- RQ: The tool shall allow the creation and storage of different standards with their associated generic artifacts. Only the global admin shall be able to maintain the generic standard part and the associated artifacts. All artifacts of a standard are listed beneath:
  1. Methods
  2. Safety Lifecycle (Clusters, Activities, Tasks)
  3. Workproducts
  4. Requirement
- RQ: The tool shall allow the creation and storage of different projects. Each project is based on a defined standard and inherits the standard specific lifecycle. Furthermore, a project is described by the following artifacts.
  1. Project-specific tasks
  2. Deliverables
  3. Project milestones
  4. Project-specific roles
  5. Project methods
  6. Reviews
- RQ: Additionally, it shall be possible to create and maintain different sub-projects of projects. Sub-projects can have the same artifacts as main projects, which is why redundant information shall be avoided.
- RQ: A project branching mechanism shall be integrated into the tool environment. Branching a project means the creation of a new project based on an existing one. More information regarding branching will be shown in 3.3.2.
- RQ: It shall be possible to create project baselines. Baselines can be created at any time and store the current state of a project in the database. Furthermore, in the future, it shall be possible to open baselines and compare them with other baselines or with the current project version. An overview of the term baselining can be found in 3.3.2.
- RQ: The tool shall provide a report generator (3.4.3) that allows the semi-automated generation of the following documents:
  - Development Interface Agreement (DIA)
  - Process-based Safety Case

  Additionally, the report generator shall allow opening existing template documents. It shall be possible to create particular templates with project-specific static content and it shall furthermore be possible to extend these templates with project-specific data. The templates shall be stored on the server and can be selected within the client-side report generator.

- RQ: The tool architecture shall be based on a collaborative client-server architecture. Therefore, the tool shall allow multi-users to concurrently work on same projects. The design of this feature will further be illustrated in Section 3.3.1 and 3.3.1.
- RQ: For storing artifacts, the tool architecture shall comprise a database management system. The created database architecture will be described in 3.5.2 and the associated database requirements will be listed in 3.1.2.
- RQ: The tool shall provide wizards for the creation of new standards and projects.

3.1.2. Database Requirements

- RQ: The database shall be an open source database. Therefore, the evaluation according to the defined requirements concludes with the usage of the MySQL (3.2.3) database.
- RQ: The database shall be a relational database where data can be selected, inserted, updated and deleted.
- RQ: The database shall be able to audit changes. For each table in the database schema, an audit table shall be generated in order to track these changes with an associated revision and time stamp.
- RQ: The database shall be able to create users and delegate rights.
- RQ: For creating database backups, it shall be possible to automatically create database dumps every night. Furthermore, the database shall allow the import of data for updating an existing database schema.
- RQ: The database shall be multi-user applicable. It shall be possible that more users can concurrently read data from tables while only one user writes on a database table.
- RQ: For the interaction between the tool and the database, an "Object Relational Mapping (ORM)" (3.2.5) shall be used. Therefore, the database shall provide a library that can be integrated into the ORM and furthermore, the ORM shall provide a library that supports the selected database.
- RQ: It shall be possible to integrate the database into the company existing IT infrastructure.
- RQ: The database shall provide an easy handling and a good support.
- RQ: The database shall consume low memory and shall be high-available and easily scalable.

3.1.3. Tool-specific Requirements

- RQ: The tool shall provide a notification mechanism. If one user changes a project, then all other users working on the same project shall be notified. Therefore, a broadcasting (3.3.1) concept shall be integrated into the tool environment.

- RQ: Wizards shall be created in order to support users during the creation of new projects, either from a standard defined template project or from any other existing project.
- RQ: A locking mechanism shall be integrated into the tool. This locking mechanism shall be supported by the ORM and shall allow the concurrent work on the same projects.
- RQ: The report generator shall allow the sending and fetching of templates from and to the server. Only the global admin is able to store a template on the server while all other users are able to fetch these templates. Nevertheless every user can fetch a template from the server and can create his own.
- RQ: The tool login mechanism shall be combined with the AVL LDAP "Lightweight Directory Access Protocol" server for user authenticating. Therefore, no password storage is needed in the provided tool database.
- RQ: It shall be possible to reuse text modules in the tool. Therefore, an automatic recommendation mechanism shall be integrated which allows an auto-completion of text inputs.
- RQ: A test-framework shall be integrated into the tool environment.
- RQ: The client-server architecture (3.3.1) shall contain two different WCF "Windows Communication Foundation" 3.2.2 services. The first one shall be responsible for the authentication and authorization and the second one shall be responsible for the data exchange between the client and the server.

The above listed requirements are only the overall main requirements of the newly developed tool and will further be processed in the following sections.

3.1.4. Conceptual Formulation

The goal of this thesis is to provide a proof of concept tool for safety management and safety case generation. Therefore, a new tool environment will be designed and integrated into the AVL IT infrastructure. This new tool shall allow the creation and maintenance of process-based artifacts and shall ensure consistency and completeness of the existing artifacts. Furthermore, it shall provide a robust framework for planning and performing an effective and efficient safety management despite the huge number of information. With an integrated database architecture, the tool shall avoid redundant information by storing artifacts in a clear and well-defined way. Therefore, the database shall provide the meta-model for the safety management tool. With an automatic data tracking, all changes during the lifecycle of a project will be stored in the database and can be retrieved in the future. At least, by providing a report generator, the tool environment shall be able to directly export reports from the tool.

3.2. Common Technical Concepts

This section gives a short overview of the commonly used technical techniques and their appropriate concepts. These techniques are used in order to develop a tool architecture that allows a client-server based communication, the storage of artifacts in a database and the graphical representation of these artifacts.

3.2.1. WPF – Windows Presentation Foundation

For the graphical user interface, the WPF[2] library is used. This library is part of the Microsoft Visual Studio tool and allows the modern development of business desktop applications. There are two main reasons [8] why this library is used instead of the default "Windows Forms[3]" library.

The first reason is that WPF strictly distinguishes the creation of the user interface layer and the associated source code layer. Therefore, it is possible that a specific graphical designer can create the graphical user interface with a different program and without any knowledge of a programming language. The graphical user interface can be created with an editor that allows the extraction of an XAML (3.2.1) file which is based on XML[4] (Extensible Markup Language).

The second reason is that there the is possibility to bind data between the graphical interface and the source code. Within an XAML file it is possible to define variables in tags that will automatically be bound to a variable in the source code. Therefore, it is not necessary to test the graphical elements in WPF and a better software architecture can also be created. More advantages of [8] will be listed beneath.

- Description of graphical elements is clearly arranged and compact
- Graphical elements are independent and can be integrated into all other elements (e.g. a button can contain another button)
- Styles can be defined for all elements
- WPF allows the usage of triggers. With triggers it is possible that graphical elements react on events and change therefore any properties
- It is possible to draw 2D and 3D graphics with the WPF library
- Animation, audio and video elements can be integrated into the view
- Provides powerful library for document editing

As mentioned before, WPF is based on an XML file that connects the graphical elements with source code properties. These files are called XAML (Extensible Application Markup Language) files and will further be described in the next section.

[2] https://msdn.microsoft.com/de-de/library/aa970268(v=vs.110).aspx
[3] https://msdn.microsoft.com/de-de/library/dd30h2yb(v=vs.110).aspx
[4] https://www.w3.org/XML/

XAML

The graphical design in WPF is based on simple XAML files. This file is similar to a standard XML file and therefore contains tags with appropriate properties. These tags represent the object tree of the graphical user interface. On the one hand, XAML can be used for building desktop applications and on the other hand, it can also be used for building web applications. Therefore, it is very flexible and can easily be exchanged within different kinds of applications. As an example, a WPF graphical user interface and the associated XAML code can be found in Appendix B.

Data Bindings

A further powerful concept in WPF are dependency properties. This feature allows the linkage between different items without writing additional codes. Will such an item be changed, all connected items will automatically be updated too. It is a very powerful feature for element styles and data bindings in the WPF concept. A dependency property is constructed like an observer pattern and contains a notification mechanism where all listing items will be notified whether the dependency property recognizes a modification. If they will be notified, they will automatically update their values.

Data Binding is the mechanism that uses dependency properties for connecting two different properties. Figure 3.2 illustrates this connection where the target side and the source side are connected by a binding path. Furthermore, the connection can also be established by a dependency property and a normal property.

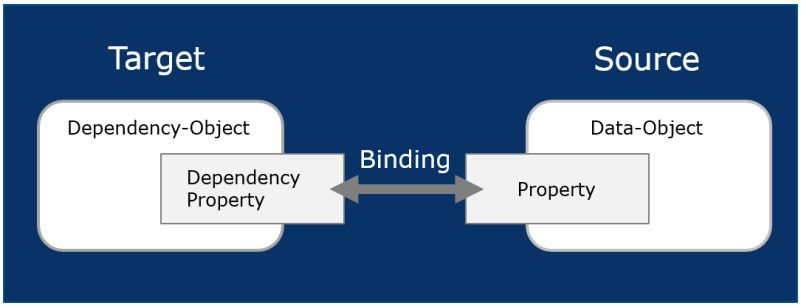

Figure 3.2.: Overview of a WPF Data Binding (Source: [8])

With this simple mechanism it is possible to create powerful and complex bindings through the different XAML and source code concepts. It is also possible to bind complex data types like lists or classes.

3.2.2. WCF – Windows Communication Foundation

With WCF it is possible to create service-oriented applications, which means that WCF can be used for communication purposes between a client and a server. Therefore, WCF in our tool environment, is the foundation for the communication-based services and the according data exchange between two different end points. Amongst the important and powerful features[5] of WCF are the following:

- Service Orientation
- Interoperability
- Multiple Message Patterns
- Data Contracts, Security
- Multiple Transports and Encodings
- Transactions
- Extensibility

WCF consists of three key concepts [5] as described in the following:

Address:
Each WCF service provides a "Uniform Resource Identifier (URI)", which can be accessed by clients.

Binding:
A Binding defines the different communication parameters for the WCF communication.

Contract:
A Contract provides all methods that can be accessed within a WCF service.

The developed tool environment provides two separate WCF services. The first service is responsible for the authentication and authorization of a user and the second one provides all methods that are needed for the data exchange between client and server. Both services have their own addresses and different bindings. Whereas the authentication service is based on a general "Hypertext Transfer Protocol (HTTP)", the data exchange service is premised on a "wsdualhttp" binding. This binding creates a duplex HTTP channel that allows the server to send notification messages back to the client. This feature is necessary for integrating the broadcasting (3.3.1) mechanism.

The authentication service authenticates a user over the company's internal LDAP "Lightweight Directory Access Protocol" server (3.3.1). Therefore, only a user with a valid AVL account is able to log on to the safety management tool.

[5] `https://msdn.microsoft.com/en-us/library/ms731082(v=vs.110).aspx`

3.2.3. RDBMS – Relational Database Management System

This section provides fundamental information about the applied "Relational Database Management System (RDBMS)". One of the main requirements that was established during the concept phase of the tool development was that the used database management system must be an open source database and able to integrate the database into the AVL infrastructure. Therefore, there were not many RDBMS options available. A very popular open source database is the MySQL[6] database, which is selected for the developed tool environment. MySQL provides a "Community Edition" which is free for open source developers. This version contains a graphical user environment for the creation and interaction with the database schema, is called "MySQL Workbench" and will be described in short in 4.1.1.

In our tool environment, a database management system for storing and manipulating the different artifacts and their relations. Additionally, data changes during data manipulations are stored inside the database. As explained in [34], MySQL has a very simple service layer but nevertheless provides a lot of very powerful features. Table 3.1 offers some advantages and disadvantages [34] of the MySQL database architecture. Furthermore, MySQL provides a lot of interfaces to all current existing programming languages, like C, C++ or C#. The database system is based on a client-server architecture and allows the definition of different users for accessing different databases in the workbench.

For the tool environment two different databases are created. The first database, named "fsmtrain", is the training database of the tool environment and is only used for development purposes. The second database is called "fsmprod" and is the productive database in the tool environment. An example cutout of this database schema will be provided in Figure 3.12. Both database are created directly out of an "EER (Enhanced Entity Relationship)" diagram and contain the same database tables. A short overview of the EER diagram can be found in Appendix C.

Advantages	Disadvantages
simple and compact handling	sometimes MySQL has some stability problems
versatile and flexible	contains a lot of add-ons and is therefore confusing for beginners
less memory consumption	it has some limitations concerning performance and fault tolerance and some very profound issues
highly available and scalability	security drawbacks
open source and good support	
a lot of documentation and online tutorials	

Table 3.1.: Advantages and disadvantages of an MySQL database system

[6] https://www.mysql.de

3.2.4. Infragistics

For improving the creation of a powerful novel application, an external library named Infragistics[7] with the WPF developer toolkit will be integrated. This library contains a lot of helpful features that already provide filtering, sorting, multi-column trees and many other enhanced features. In the following, the features which will be integrated into the safety management tool will be depicted in more detail and in Appendix E, a list of all existing features is included.

- **XamBusyIndicator**

  Provides visual notification that a control or an item is currently loading or processing data. This extension is useful because our application always sends requests to the server before, the server fetches data from the database and sends back an appropriate response. To make the user attentive that something is happening within the tool, this feature is very helpful. Furthermore, it can be used to give the users an overview of the current progress state of a work (e.g. during document export).

- **Drag and Drop Framework**

  This extension offers a powerful drag and drop feature that can be applied to any existing element (e.g. Images, Controls, etc). Furthermore, it provides channels for the drag elements (source) and channels for the drop elements (target) which specify where the elements can be dragged. Therefore, it is really simple to apply different targets for different source elements. Elements can additionally be animated during the dragging.

- **XamCarouselPanel**

  The XamCarouselPanel allows the graphical visualization of elements according to a defined path. With a navigation bar or with the mouse wheel it is then possible to scroll through these elements defined on the path. In our tool, this feature is used for providing predefined workflows. The path defines the order of the different workflow steps and contains buttons with which the different modules can be accessed.

- **XamComboEditor**

  Inside grids, the XamComboEditor can be used for viewing a more powerful drop-down list. This feature is more flexible than the standard combo editor and additionally provides a multi-select feature or the possibility to add checkboxes to the drop-down items. The XamComboEditor can also defined templates for the contained element. These templates determine the visual representation of the drop-down list and can for example select text with images or any other graphical elements and colors.

[7] http://www.infragistics.com/

- **XamContextMenu**

  The XamContextMenu provides pop-up menus by clicking on graphical elements. It allows adding icons to the different context menu items and provides a simple mechanism where the clicked elements can be retrieved in order to manipulate them. Additionally, the context menu items can be cascaded into a deeper level of menu items. It is also possible to define only single elements where the context menu can be opened.

- **XamDataCards**

  The XamDataCards are a graphical representation of objects like cards. These cards can show additional properties and are selectable. In our tool, this view is used for providing an overview of all existing projects. Furthermore, by double-clicking on such a card, the detailed project overview can be opened.

- **XamDialogWindow**

  For providing pop-up windows, the XamDialogWindow is used. It allows the creation of modal and non-modal dialogs with additional blur effects. The window can be minimized, maximized and closed by the users and allows a simple interaction with the view arranged behind.

- **XamDockManager**

  With this extension, it is possible to manually place the different elements on any specific part in the view. Furthermore, these elements can be pinned and unpinned in the view. Therefore, each user can hide and show only the graphical elements which are needed in order to simplify complex graphical user interfaces.

- **XamGrid**

  All views that contain a table are built up with this extension. It provides a powerful datagrid with a lot of extensions like filtering, sorting, grouping, merging and many others. Additionally, every graphical element can be used as column inside this grid. Another powerful extension is that the grid allows real-time checking and real-time updating of embedded data.

- **XamMultiColumnComboEditor**

  This is an extension of the aforementioned XamComboEditor and allows drop-down lists with several columns. In our tool environment, it is used for selecting a specific standard. Therefore, it is needed to show the standard name and the associated version in the drop-down list. In principle, the drop-down list is a grid view that can be built up individually.

- **XamRichTextEditor**

  The XamRichTextEditor is used in order to provide users with a feature where images, tables, hyperlinks and formatted text can be inserted. It is really useful for description fields inside the developed tool. Furthermore, it offers the possibility to open and show content from Microsoft Office Word or HTML documents and allows the export to these formats. An automatic undo-redo functionality and zooming mechanism is also comprised in this feature.

- **XamTreeGrid**

  With this view, it is possible to show hierarchical data in a tree. In addition to the general tree, this extension is able to show more than one column within the tree view and allows the dynamical switching of the tree between the different columns. Within the developed tool, this feature is used very often, for example for the creation of the safety lifecycle. By an integrated drag and drop mechanism, the different elements can be dragged from or to the tree. As within all trees it is also possible to expand and collapse the child nodes and to filter them.

- **XamThemeManager**

  All aforementioned extensions have the same style that is defined by the XamThemeManager. It is a powerful feature that allows the dynamical changing of different predefined themes. In the tool settings of the newly developed tool, there is the possibility to individually change the appearance of the tool. Furthermore, it is also possible to create new themes based on individual taste.

3.2.5. ORM – Object Relational Mapping

An object relational mapping will be introduced in our tool architecture in order to connect the server and the database. Therefore, an ORM allows the linkage between different systems by simple additional configuration. Because interacting with a database is a very complex and challenging task, the manual creation of database statements is very time-consuming and error-prone. By connecting the server logic with the database logic through an ORM, only a configuration between the system must be created while the library is responsible for all database interactions. The configuration maps instances of classes directly to database tables and vice versa.

In the provided tool environment, "NHibernate[8]" (3.3.3) will be used as ORM. It is an open source library that was developed for the .Net framework. NHibernate additionally provides an extension called NHibernate Envers[9] (3.3.3). This extension is also an open source library and allows the automatic tracking of data changes during the entire project lifecycle. It is responsible for all insertions, updates and deletes and creates the appropriate revision entries with the time stamp in additionally created audit tables.

[8] `http://nhibernate.info/`
[9] `https://bitbucket.org/RogerKratz/nhibernate.envers`

3.3. Tool-specific Concepts and Design

This section provides fundamental information about the design of the different tool-dependent concepts and their appropriate features. First, the client-server architecture will be designed which includes the design of the communication services, the multi-user feature and the broadcasting mechanism. Then, the client concepts branching and baselining will be defined in detail. Finally, the integration of the object relational mapping library will be illustrated.

3.3.1. Client-Server Concepts

By providing a client-server architecture as displayed in Figure 3.3, it is possible that more people can work in a concurrent and collaborative way on the same projects. For designing such an architecture, some things must be kept in mind. The connection between the client and the server must be encrypted in order to provide a confidential data exchange. Another issue, is that a client which will access the server must be authorized and registered on the server. The server needs this information because if one user changes some artifacts on a project while another client also works on the same project, the tool shall inform the other client that something has changed within the project. Therefore, a broadcasting mechanism as described in 3.3.1, which notifies all registered clients with a defined message is integrated. For the authentication and authorization of a client, a particular WCF service is created that handles the user authentication and authorization on the server. Both concepts will further be described in Section 3.3.1.

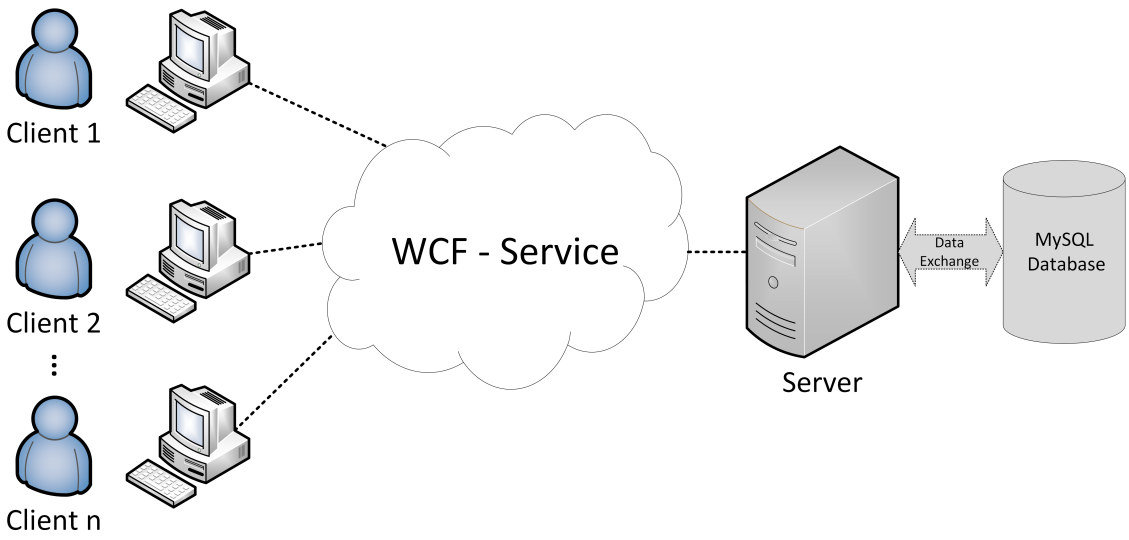

Figure 3.3.: This figure shows an example client-server architecture with an integrated database management system. An arbitrary number of clients can access the database by the provided service.

The client side is developed as a WPF desktop application (3.2.1) and the client-server connection is based on the WCF (3.2.2) communication. WCF is mostly used for the development of distributed systems within service-oriented architectures. The communication via this service is based on SOAP/XML messages.

On the server side, a database will be integrated into the back-end which is able to store the different artifacts that are sent from the client. Additionally, the changes will be tracked on the server by the integrated ORM. The ORM automatically stores the revisions in the database.

One powerful feature that is included in the tool architecture is the baselining concept (3.3.2). With baselining it is possible to make a snapshot of the current state of a project. Furthermore, in the future, it shall be possible to open such baselines and compare them to other baselines or to the current project state.

Authentication Service

The provided authentication service is responsible for the user authentication and authorization. The service has a connection to the company-specific LDAP server. By using the LDAP authentication of the company, it is not possible to store the password in the provided database. Therefore, password hashing and password storage in a safe manner is not needed while integrating these security issues are very costly tasks.

If the user is authenticated, the service automatically checks whether the user exists in the database and extracts the associated user permission which is stored inside the database. Only the global admin has the permission to create new users with associated rights in the database. Therefore, if the user does not exist in the database, the authorization fails and the application shows an according error message. Otherwise, if the user exists in the database, the authentication service creates a response that contains an encrypted cookie with the associated user permission. With this cookie, the user is now able to exchange data between the client and the server via the data exchange service. The data exchange services needs a valid cookie in order to allow the user the execution of the called method on the server.

Data Exchange Service

Whereas the first provided service is only responsible for the authentication and authorization, this service is in charge of the data exchange between the client and the server. Once a user is successfully authenticated and authorized, the user is able to access data in the database. If the global admin also provides the user with write access, he or she is additionally able to manipulate data within projects.

The service provides different methods for inserting, updating and deleting data in the database. An example service method is illustrated in Listing 3.1. This method automatically checks the user permission of the current user with the “PrincipalPermission” attribute. This attribute will automatically be set by the request which contains the cookie and therefore the appropriate user permission. If he or she is not allowed to execute the called method, an exception will be sent back to the user. If he or she is allowed to execute

the method, the called method returns the ID of the newly created item.

Inside the called method, a “dataSessionContext” object is created. This context object is the interface to the integrated ORM and contains the interacting database methods. Therefore, the service has no knowledge of the database connection. It only provides the user with a simple interface and the complex business logic lays in the data access layer which is stored in the background. A second example method for fetching an item with an existing item ID is depicted in Listing 3.2.

```
[PrincipalPermission(SecurityAction.Demand, Right = "GlobalAdmin,
    ProjectAdmin,WriteUser")]
int FSMServiceContracts.IFSMService.SaveItem(Item item)
{
  IDataContextManager dataSessContext = _dcFactory.CreateNewContext();
  try
  {
    return dataSessContext.SaveItem(item);
  }
  catch (Exception ex)
  {
    throw new FaultException<FSMServer.Faults.ProcessFault>(
      new FSMServer.Faults.ProcessFault("Could not save item!);
  }
}
```

Listing 3.1: Overview of the save method provided by the data exchange service

```
[PrincipalPermission(SecurityAction.Demand, Right = "GlobalAdmin,
    ProjectAdmin,WriteUser")]
Item FSMServiceContracts.IFSMService.GetItemById(Entities entity, int
    id)
{
  IDataContextManager dataSessContext = _dcFactory.CreateNewContext();
  try
  {
    return dataSessContext.GetItemById(entity, id);
  }
  catch (Exception ex)
  {
    throw new FaultException<FSMServer.Faults.ProcessFault>(
      new FSMServer.Faults.ProcessFault("Could not fetch item!);
  }
}
```

Listing 3.2: Overview of the get item by ID method provided by the data exchange service

Multi-User Integration

As mentioned at the beginning of this chapter, the tool shall allow different users to interact with it on the same projects. Because the client application will be distributed to all users, they can work with the individual applications on the same database and therefore on the same projects. For integrating this feature, some points however have to be considered.

For introducing a multi-user approach, the tool has to deal with some concurrency problems. It may be the case that two users update the same project artifact at the same time. What happens if the first user updates a field and the second user updates the same field as well? The first value will, thus, be overridden and the first user has no knowledge of this update. To avoid such a situation, the tool needs a multi-user approach.

The database interaction in the provided tool architecture is done by the introduced object relational mapping (3.2.5) library. This library already provides features for avoiding such situations by introducing specific concurrency controls. The library allows two different approaches. The first is based on the principle of the quickest wins. In this case, the user who calls the method first updates the value and the second one gets an exception. In the second approach, the object relational mapping library locks the current database table with a specific table entry and no other transaction can access this table during the update. For this approach, the database schema has to be updated with the necessary lock flags.

In our tool architecture, the first approach is used because it needs no database updates and the locking can directly be done in the code. Additionally, because a broadcasting mechanism is integrated into the tool architecture the error message can be sent back to the client. Thereby the user is informed that he or she cannot update the artifact because another user was quicker. This user must now refresh the data and then he or she is allowed to update the artifact in the database. Therefore, no direct locking of database tables is necessary.

As mentioned before, for integrating a multi-user feature, a notification mechanism for data changes is needed. This mechanism is named broadcasting and will further be explained in 3.3.1. Summarized, it can be said that the broadcasting mechanism checks changes within project artifacts and sends notification messages to all users which are currently logged in. These users get a message saying that they should refresh their content in order to fetch the latest changes. Hence, it is important that all users are registered on the server so that the latter is able to send a message to the clients through the provided duplex HTTP channel.

Broadcasting

Because the tool environment allows several users to work on the same projects, a broadcasting mechanism is needed. This feature notifies all users whether some changes have occurred in the database. If two users work on the same project and one user updates a field in the database, the broadcasting mechanism automatically sends a message to the other registered users. With this feature, the tool shall guarantee that the users always work on the newest data and therefore, the problem of interferences between user changes shall be avoided.

Currently, only a simple version of broadcasting is implemented where all registered users are informed that they should refresh their data. In the future, this feature shall be further improved so that only the persons who work on the same projects receive the message. To illustrate the feature, Figure 3.4 shows the integrated broadcasting architecture. There are three different clients registered (black dotted arrow) on the server. Client one sends an update request (green arrow) to the server and the ORM changes the specified table entry in the database. If the update was executed successfully, all other clients will automatically be notified by the server (red arrow). For each notified client, a message will pop up saying that he or she shall refresh the values because another client has changed something in the database.

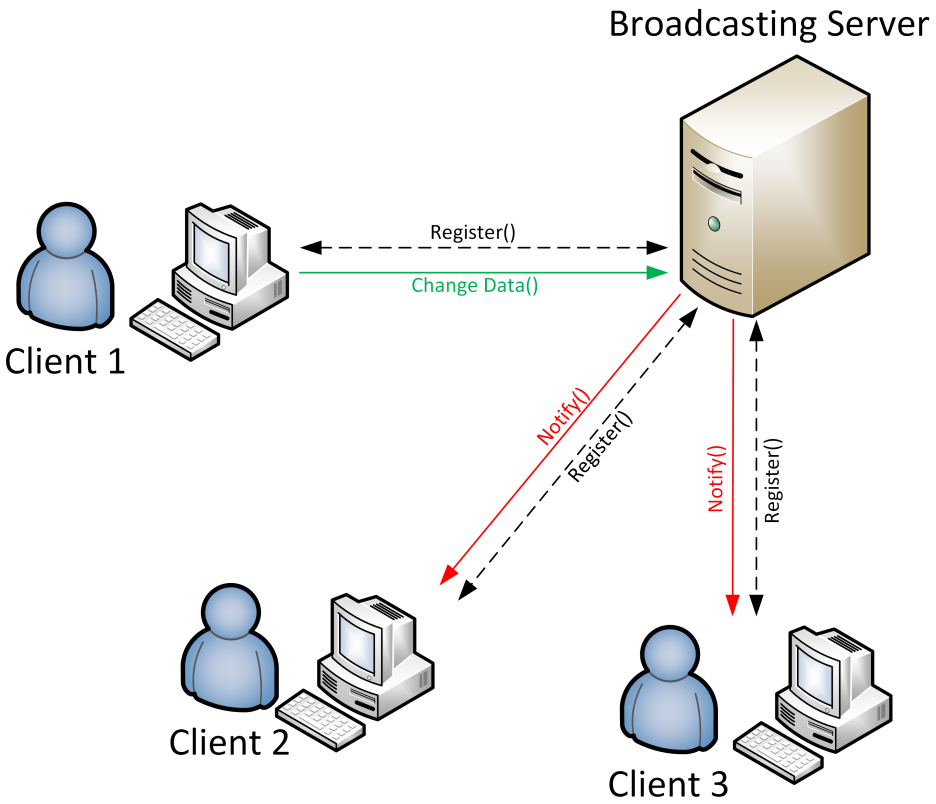

Figure 3.4.: Example broadcasting architecture with three associated clients

3.3.2. Client Concepts

This section explains the design of the client specific features. These features contain a report generator that allows the automatic generation of reports directly from the tool and furthermore the baselining and branching feature.

Report Generator

A report generator shall be integrated into the tool architecture. This report generator shall be able to fetch templates which are stored on the server. Additionally, a global admin is able to send a template from the client to the server in order to add new templates and hence, for each project, a particular template can be created by a user. The user needs to send the template to the global admin, and once he or she confirms the template, he or she is able to upload it.

Furthermore, it shall be possible to attach project-specific data to these templates. A template shall contain a cover sheet and some project-specific static contents like the introduction or the abbreviations. The report generator shall be able to add project-specific data from the tool to these templates in order to create a preliminary release document.

Baselining

The term baselining is defined as a snapshot of the current state of a project at an point of time. Therefore, with the baselining feature, it shall be possible to save an arbitrary project state in the database. Such a framework is needed because it shall be possible to open and export such baselines from the database in the future. Thereafter, it shall be possible to compare these baselines to an arbitrary project state or to another baseline and extract the differences to see what data have changed in the different versions. This is a very powerful feature because the manual checking of these differences is very time-consuming. Automatically receiving this information brings hence a lot of benefits. The differences can for example additionally be sent to the customer for quickly showing which data have changed since the last release.

Branching

Branching is another important approach where new projects can be created from template projects. At the beginning, if no project exists, a new project can be created only from an existing standard. If one project already exists, it shall be possible to branch a new project from the existing one. By introducing a wizard, the tool provides a simple mechanism to branch projects or standards. Additionally, it is possible to create links between the branched project and the new project. Therefore, they can for example share the same deliverables within different projects. This is a very powerful feature for creating sub-projects. Sub-projects mostly have some equal deliverables and therefore it is not necessary to create redundant information within the database. One thing that is very important is that a branched project always inherits the same standard as the original project. In addition, branching is also possible for standards and new standards can be created from an existing standard.

3.3.3. Server Concepts

In the following, the features for the server architecture will be explained in more detail. The server contains the object relational mapping library which further interacts with the database behind. Furthermore, the server contains the data tracking library which is needed in order to track and save data changes.

ORM - NHibernate

As mentioned in 3.2.5, NHibernate is an open source library and is responsible for the database mapping and the associated data exchange. The book [35] describes NHibernate as an "adaption of the powerful Hibernate that was originally developed for JAVA". Furthermore, the book depicts that "this feature is already used within over thousands of productive applications." It provides a simple access layer for the database while a developer does not need to have knowledge of the database logic behind. The main architecture of NHibernate can be found in Figure 3.5.

For using NHibernate, the library must be configured according to the used RDBMS. This configuration contains a lot of different properties like the connection string and the mappings for all existing entity tables. Each table that shall be mapped to an entity must also be configured in a special way. In the proposed tool environment, for each table a particular entity class and a particular entity configuration file is created. An example of such a configuration can be found in Appendix D.

Therefore, one table from the database is exactly mapped on one entity class in the source code layer. NHibernate contains different strategies for the configuration of entities. In this work, the general and powerful XML configuration is used because it is very simple to read and everyone should understand XML files. This approach supports all existing configuration properties and is therefore widely used. The only drawback of this approach is that it is very easy to include some typing errors in an XML notation as this configuration provides all necessary features and is used in most applications. After the configuration is specified, a session factory element can be created. This factory element only exists once per database connection and is therefore thread-safe.

In our defined software architecture the "Session-Per-Request" pattern is used. This pattern determines that exactly one WCF service call contains one session object which will automatically be created by the service call. This session can be created from the provided session factory element, thus, contains therefore all configurations and is only valid until the service call is completed. By using this pattern, we avoid the case that sessions will be created through different WCF calls and define a clear way of processing a client request. Sessions are not thread-safe, which is why only one session per request shall exist.

As a consequence, a particular session element is created for each WCF service call which provides the methods for accessing the database artifacts. For this purpose, a session contains the following database manipulation methods (insert, update, delete) for the entities:

- **Save**
  For storing a new entity in the database, the save method is used.
- **Update**
  For updating an already existing database entity, this method is used.
- **SaveOrUpdate**
  If the user does not know whether the entity already exists in the database, this method is useful. It automatically checks whether the entity exists and calls the associated save or update method.
- **Delete**
  This method can be used for deleting already existing entities from the database.
- **Refresh**
  With the refresh method it is possible to bring changes from the database to the memory. For example, if another user has updated an entity, the refresh method is really helpful in order to fetch the changes from the database.
- **Merge**
  Merge is very similar to the update method and is used if an entity is already present in a session element. This method is not used within our session-per-request pattern and will not be further investigated.
- **Persist**
  The persist method is very similar to the save method. The only difference is that with this method, an entity can only be saved within existing database transactions.

All these existing methods can be called within each session element and are embedded in a database transaction to ensure consistent and concurrent database accesses. A transaction allows the procession of multiple statements within one unit and can automatically roll-back all statements if an error occurs during this execution. Additionally, it is possible to lock database calls inside these transactions, which is why a more smooth multi-threaded environment could be provided.

For the relationships between entities, NHibernate supports the one-to-many, many-to-one and many-to-many relationships and provides cascades that automatically process connected entities in order to offer a consistent and complete database schema.

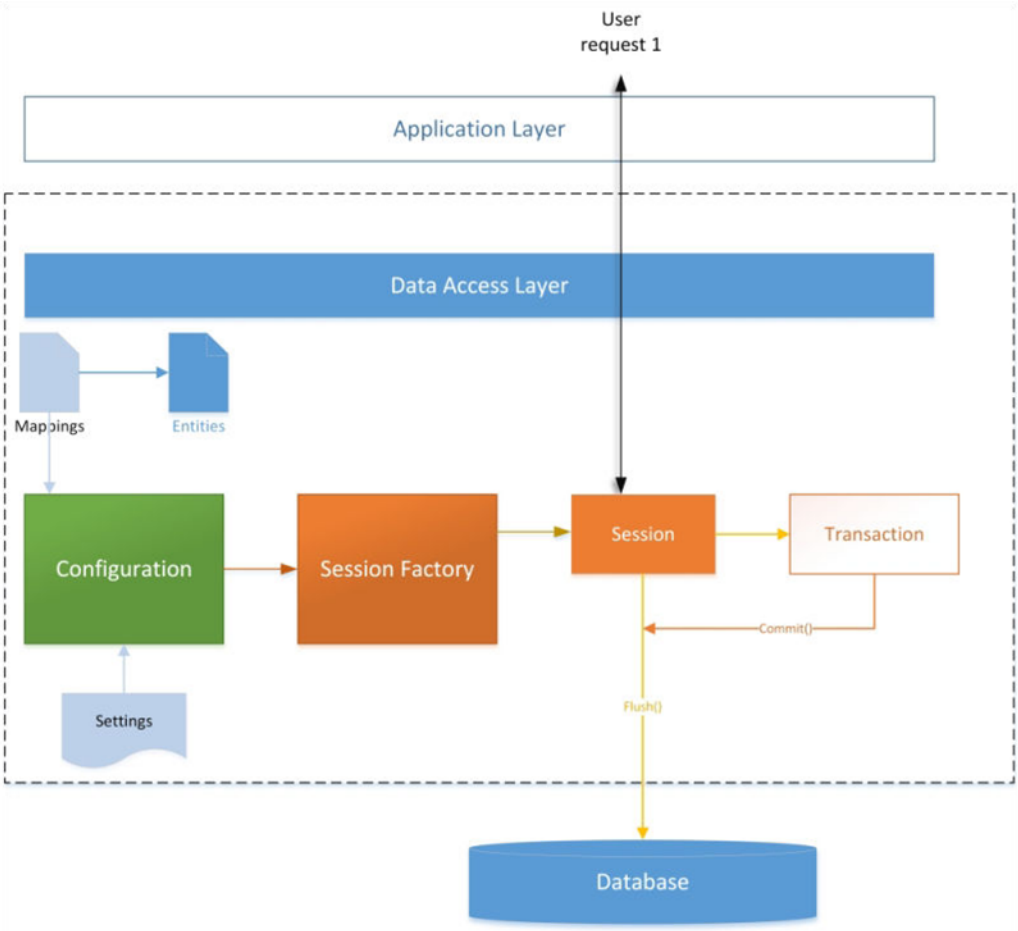

Figure 3.5.: The NHibernate architecture with the SessionFactory and the associated sessions over the different layers (Source: [35]). The figure additionally shows the connection between the different NHibernate artifacts.

Retrieving data from the database with NHibernate

For retrieving data from the database, NHibernate provides different possibilities. In the following, some famous possibilities will be described in more detail. A first approach is named "Hibernate Query Language (HQL)". This approach is very similar to the general database statement syntax. Therefore, the only difference is that the HQL approach does not use the database table names inside the statement. Instead, it uses the entity class names that are defined within the mapping files. This approach is not very recommendable because it is also necessary to write complex database queries. By introducing an ORM, this complex query creation shall be avoided. If only simple database statements are needed, this approach, however, is helpful.

"Criteria API" is another approach that is provided by the NHibernate framework. It is a simpler approach that only requires little knowledge of the database query language.

It allows the fetching of entities by using the provided "ICriteria" software interface. This interface only needs the entity type or name that shall be fetched from the database. Additionally, it is possible to add some restrictions (e.g. search criteria on table properties) to this interface in order to filter the result. The main problem with this approach is that the restrictions have to be written in common strings, and hence, it is really easy to include errors in the search query.

A more powerful approach is the "QueryOver API", which allows the building of dynamic queries during runtime. It is generally not based on a common string approach like the approach before. Nevertheless, it cannot be avoided to write common text into the queries, particularly if complex queries are needed.

The final approach is the "Linq" approach. It is a common querying method in .Net applications and is therefore also compatible with the NHibernate framework. However, this approach only works with generic types and has a similarity to a general SQL syntax.

In our tool environment, a combination of approach number two and three is used. The first approach is too complex and does not improve our work. Because the "Linq" approach only works with generic data types, it cannot be used within the provided tool environment because generic types are not supported by WCF services. Instead, we use a combination of the "Criteria API" and the "Query Over API". Listing 3.3 shows a defined SQL query in all different approaches. All of the above mentioned approaches allow the usage of powerful joins.

```
SQL: select * from User where Name = "Max";

HQL: var user = database.Session.CreateQuery("select e from User as e
                where e.Name = 'Max'");

ICriteria API: var userQuery = database.Session.CreateCriteria<User>();
               userQuery.Add(Restriction.Eq("Name", "Max"));
               var user = userQuery.List<User>().UniqueResult<Item>();

Query Over API: var user = database.Session.QueryOver<User>()
                                   .Where(u => u.Name == "Max")
                                   .List<User>().FirstOrDefault();

Linq: var user = (from e in database.Session.Query<User>()
                  where e.Name == "Max"
                  select e).FirstOrDefault();
```

Listing 3.3: Overview of the different NHibernate data retrieve approaches

Lazy Loading

NHibernate provides a powerful feature named lazy loading. This feature is responsible for loading only the information from the database that is currently needed. This feature is activated by default and can be dropped individually for each entity by setting the specific property in the entity configuration file. A great benefit by using this feature is that it decreases the amount of data exchange and therefore increases the performance of the tool.

This feature can also be activated for collection entries. Hence, only the needed elements of a list will be retrieved from the database.

One problem that occurred during the integration of NHibernate into the developed tool environment is that it does not support the lazy loading feature in combination with a WCF service. The reason is that the lazy loading only works if a one session-per-application pattern is used.

Lazy loading always needs an open NHibernate session in order to fetch the specific data that is currently needed. For example, if an user entity which contains a list of projects is fetched from the database the detailed information of the projects will not be fetched at this point as well. But if the list will be accessed later on, NHibernate automatically loads the needed projects from the database with the before created session.

In our tool, this is not possible because our tool architecture is based on a session-per-request pattern and therefore the session is always closed after the completion of a request. This means that, if the session is already closed, the lazy loading feature does not work properly. In order to fix this problem in our tool environment, a specific lazy loading implementation is integrated.

This new layer is hence integrated between the NHibernate layer and the database layer and is responsible for the correct loading of data from the database. During the first call, it loads all needed data from the database and stores them in this additional layer. The response then contains all needed data.

Now, if data must be re-fetched, the new layer will automatically check whether it has already been fetched from the database and returns the values if they exist in the layer. The integrated layer is constructed similarly to a cache layer and speeds up the application. With this layer, it is possible to enable lazy loading within a WCF service.

If such a layer does not exist, each database call must eagerly fetch all connected data from the database. Because there are a lot of relations between the existing tables in the database, the database is a large cyclic graph. Therefore, fetching such a huge amount of data every time decelerates the tool a lot.

NHibernate Envers

For tracking data changes, NHibernate provides a powerful open source add-in named NHibernate Envers[10]. This extension is responsible for all database changes and automatically saves the changes and the associated revisions in predefined database tables. Furthermore, it provides a powerful querying mechanism for fetching old revisions. As a consequence, it is possible to retrieve old history data either by revision numbers, by a user who has changed the data or directly by an entity type. So it is really easy to get the changes out of the database.

Integrating this extension is just as well really simple because it only needs some additional configurations within the NHibernate configuration. With a few lines of code, it is possible to integrate the powerful add-in. Nevertheless, it provides a lot of different configuration properties that allow the refinement of saving strategies or the integration of other database-specific configurations. All existing properties can be found in the NHibernate Envers documentation[11].

In the provided tool environment, a simple strategy is used that creates a new revision entry for each database change. To do so, NHibernate Envers needs two things. First, it needs a specific revision table in the database which contains the revision ID, the time stamp and the user who changed the property. Furthermore, for each revisioned entity, a particular audit table is needed. An integrated tool extension that is provided by NHibernate Envers is able to automatically create these additional tables with the defined configuration.

No manual database configuration is needed to get this library working. After creating the additional tables, all entities that shall be audited must be configured once in the basic NHibernate configuration. That is it! Now, the extension automatically creates and stores data changes that will be made on the different entities. It is also very easy to extend the library in order to save additional information in the provided revision table. With little effort it is possible to create a powerful auditing environment that allows on the one hand the automated data tracking and on the other hand a simple data retrieving of historical data. For retrieving historical data, the extension provides a special data retriever class that allows the querying of historical data.

There are some drawbacks within this extension. It does not work with legacy database and has currently some problems with one-to-one relations. Because these things are not used within our tool environment, the library can be integrated without problems.

[10] `https://bitbucket.org/RogerKratz/nhibernate.envers`
[11] `http://envers.bitbucket.org/`

3.4. Collaborative Safety Management Tool Architecture

This section describes in detail how the aforementioned features and concepts are integrated in the newly developed safety management tool and the associated environment. Figure 3.6 illustrates the entire tool architecture with the different key concepts. The client side application is a standalone desktop application and is connected to the server with two different WCF services. The first service interacts with the AVL LDAP server and is responsible for the authentication and authorization of the users. The other WCF service is responsible for the data exchange between client and server and additionally provides a broadcasting mechanism. The server and the database are connected via an object relational mapping library. This ORM is responsible for the data access on the database layer by providing a server side business logic. For the secure data exchange connection between client and server, an HTTP cookie is used.

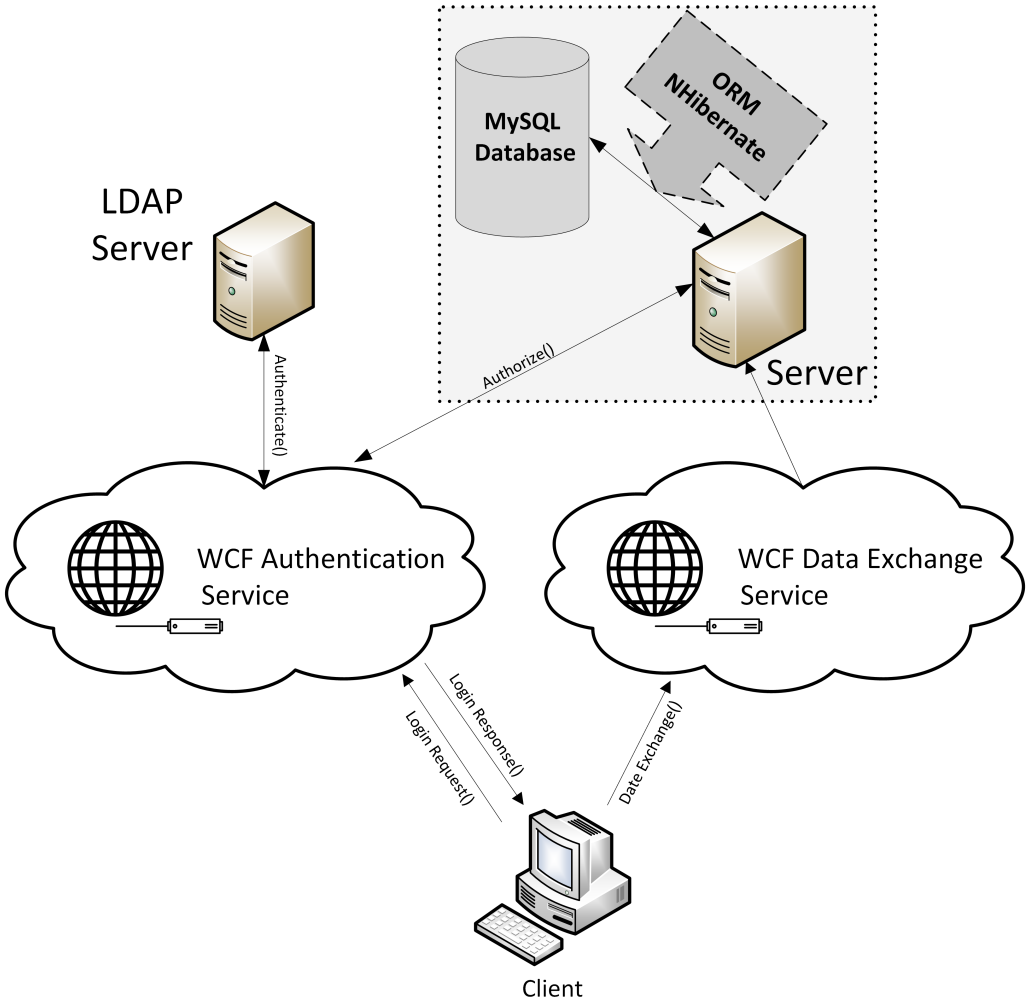

Figure 3.6.: This figure shows the overall system architecture of the developed Functional Safety Management Tool. Additionally all existing features and their interactions are displayed above.

After the user has successfully been authenticated and authorized, a cookie will be created by the server and will be sent to the client. This cookie comes as encrypted ticket within the HTTP message header from the service and will be stored in the client's memory. Additionally, this cookie contains a time stamp and a period of validity in order to guarantee a valid time range. Now, if the client sends a request via the data exchange service to the server, this cookie will be attached to the message header. On the server, the cookie will automatically be decrypted and the information will be stored in the current "Principal Permissions". This attribute is a C# specific attribute and can be defined above a method body signature. By using this attribute, it will automatically be checked whether the cookie is valid and the user has the necessary rights in order to execute the called method. If the latter is not the case, an exception will be sent back to the client. One important thing to mention is that the user only authenticates his identity once and then the received cookie will be saved until the client closes the connection to the services by closing the tool or when the valid time expires.

This section gives an overview of all existing safety management parts that will be integrated into the safety management tool. These parts are responsible for the creation and maintenance of standards and projects. Furthermore, in this section, two processes illustrate how an interaction between the client and the server can look like are illustrated. These two sequences are depicted in Figure 3.7 and Figure 3.8. Both sequences show a typical interaction of the single features within the tool architecture. Moreover, the design of the report generator and the appropriate reports will be shown.

3.4.1. Safety Management Tool Parts

To provide a safety management application in the tool, the client architecture is subdivided into two different main parts. The first part is the standard-specific part and contains all generic artifacts and relations provided by a standard. For the standard-specific part, different modules are created. The different modules will further be explained in 4.2.1. The second part is the project-specific part and contains only the artifacts that are needed for creating the different projects and the appropriate artifacts. Furthermore, this part contains the relations to the inherited standard-specific artifacts. All modules are listed beneath, while detailed information of all project-specific modules will be provided in 4.2.2.

The enumeration shows the different modules for both existing parts

- Standard-Specific Part
  - Methods module
  - Workproduct module
  - Requirement module
  - Task module
  - Lifecycle module
- Project-Specific Part
  - Deliverable module
  - Deliverable responsibility module
  - Milestone module
  - Project role module
  - Tailoring module
  - Project-specific lifecycle module
  - Project task module
  - Review module

3.4.2. Safety Management Tool Processes

Process 1: Get item from database

In Figure 3.7, the process of fetching an item from the database is shown. The first step in this process is to start the tool by double-clicking on the provided executable. Now a login screen appears and the user must enter his or her company specific username and password. By confirming the data, the authentication service will be invoked and checks whether the user can be authenticated within the AVL infrastructure. What is more is if the user is authenticated by the AVL LDAP server, the authorization of the user will be invoked. The service sends a request to the server and the latter checks whether the user is defined in the database and may rightfully access the tool. If one valid right is found, the service creates a cookie and stores the username with some additional data inside this cookie. Now the cookie is encrypted and will be sent back to the client. In the next step, the tool is able to create the request for fetching data from the database.

After the user opens a specific module, data can be fetched from the server. Therefore, the cookie will be stored in the message header and attached to the server request. If the user has the permission to execute the called method with his or her special right, the ORM creates a new database session on the server. This session will be used to access the database and to fetch the needed data. If no error occurs during the database calls, the data exchange service will send back a response with the fetched data. Before sending this response back to the client, the database session will be closed in order to provide no memory leaks and no multi-threaded issues. The user receives the data and can proceed further with the data.

Process 2: Save item in the database

The provided graphical user interface of the safety management tool also allows the creation of new entities and to store them in the database. Therefore, this process (Figure 3.8) describes the according steps. The beginning of the process is exactly the same as in the process mentioned before. If the user is authenticated and authorized correctly, a new entity that was created within the safety management tool can be sent through the data exchange service server. Therefore, the user calls the provided save method from the data exchange service and delivers the item as method argument. Now the service checks again whether the request has a valid cookie and the user is allowed to perform the save method on the server side. Because of differently provided user permissions within the tool, not every user is allowed to save data in the database. If the user is allowed to execute the method, the object relational mapping is invoked and is responsible for the relevant further steps for saving the entity in the database.

If everything was executed without an error, the ORM returns the ID of the newly created database entry to the server. The server attaches this ID to the response and sends it back to the client. Now, the user is able to manipulate this entity. The ID of an item is very important because without an ID the item cannot be linked to other items and processed further by the ORM.

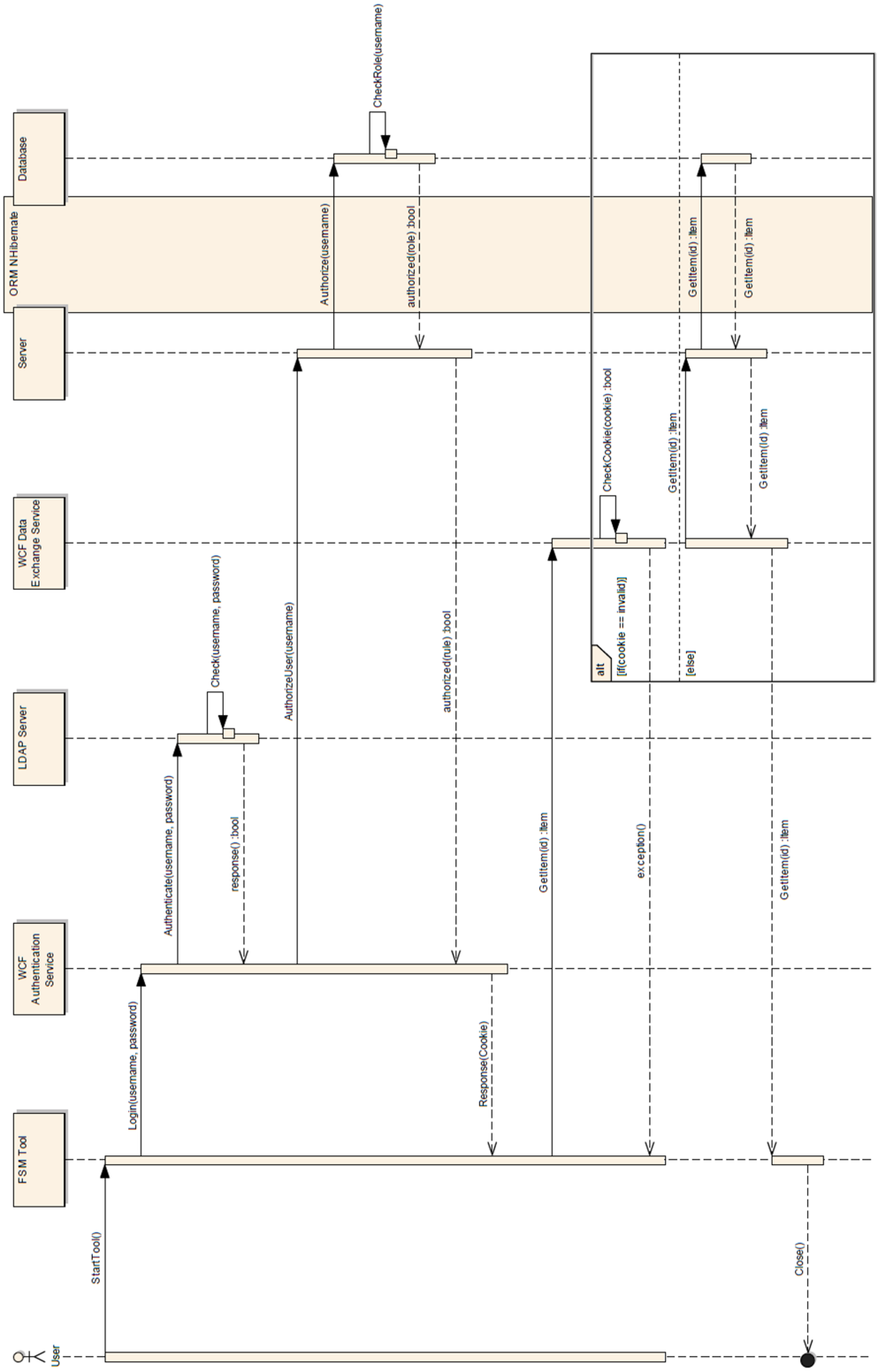

Figure 3.7.: The figure shows the way how data can be fetched from the database through the provided tool environment. Furthermore, all detailed steps and the according actions are displayed.

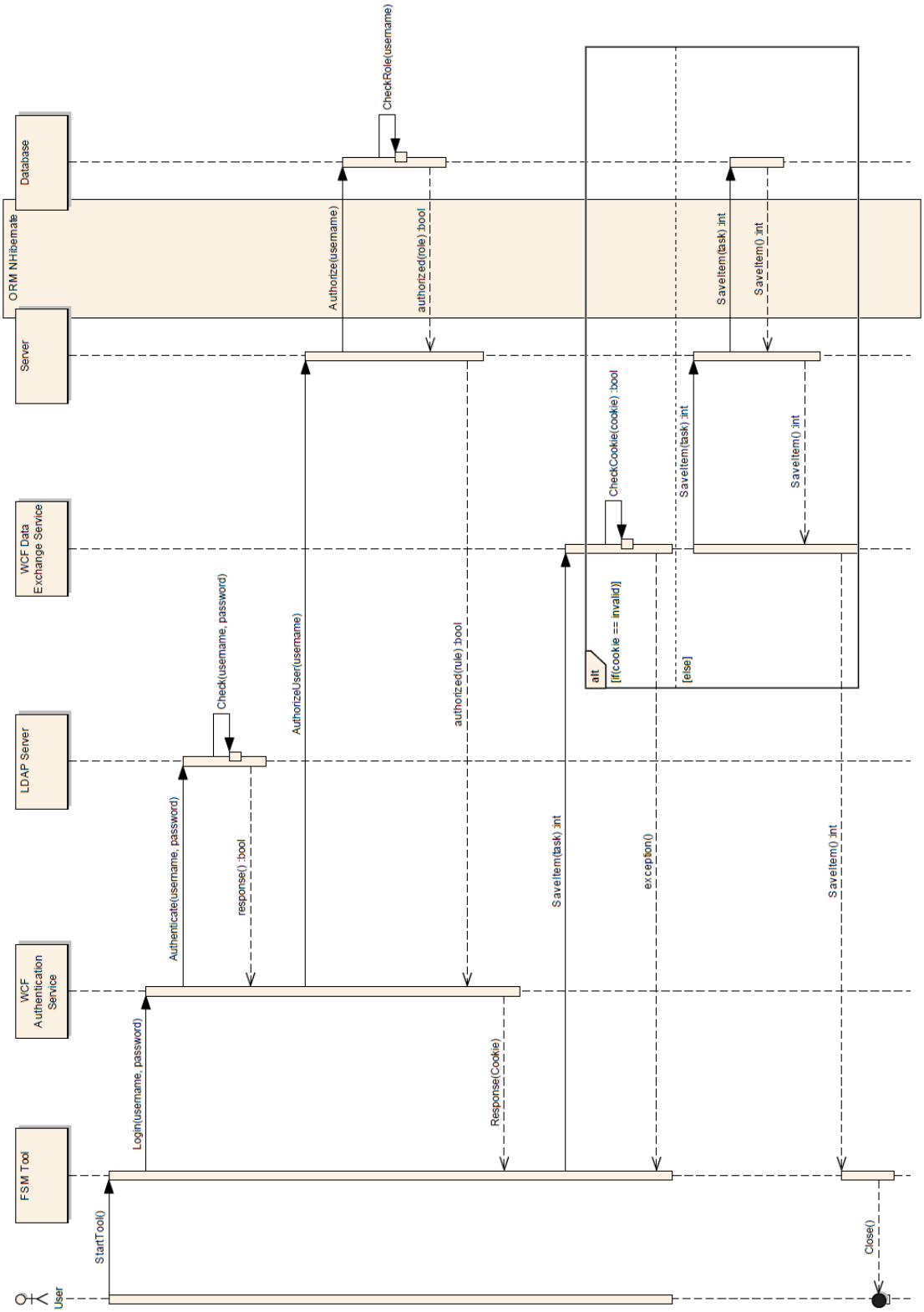

Figure 3.8.: The figure shows the way how a new item can be stored in the database through the provided tool environment with all detailed steps and according actions

3.4.3. Safety Management Tool – Report Generator

For extracting the provided information from the tool and putting it into predefined templates, a report generator is integrated into the tool environment. This report generator shall provide a simple API that allows the quick enhancement of already existing reports and the smooth integration of new ones. Therefore, the report generator shall be able to interact with Microsoft Office Word files. Additionally, it shall provide methods which can be used to insert and update Microsoft Office Word specific artifacts like headers, tables and others.

In this thesis, the focus is on two different reports. The first one is the development interface agreement report and the second one is a report for a preliminary process-based safety case generation. For both reports templates exist which already contain some static content. This static content consists of the cover sheet and provides some introductions and general chapters. Now, the report generator shall be able to open these predefined templates with the associated static content and shall be further able to add dynamic content afterwards. The dynamic content shall therefore contain the project-specific information from the database.

The report generator shall be integrated into the tool environment and shall allow users to select existing templates for the different reports. These templates shall be stored on the server and the provided tool environment shall allow users to fetch specific templates from the server. Furthermore, these templates shall be storable and extendable on the client side. The implementation of the report generator can be found in 4.2.3.

3.4.4. Process-based Safety Case Design

Within this section, the safety case design within the tool environment will be investigated in more detail. As described in 2.1.2, a safety case is a way to show that a system is free from unreasonable risk and therefore implemented according to a defined standard. A safety case can be subdivided into two different safety case types.

On the one hand, a process-based safety case exists and on the other hand a product-based safety case exist. Product-based safety cases shall show that the product is safe. In this thesis, the focus is on process-based safety case generations which shall ensure that the process is safe fits the standard.

Therefore, within the tool, it shall be ensured that all necessary artifacts for the process are linked in a clear and understandable way. It is very important to show that the traceability of the artifacts is given. Additionally, the report generator (3.4.3) shall be able to export these artifacts and the traces and put them into a preliminary process-based safety case template.

This safety case shall show the traces between the following artifacts:

- Project-Specific tasks
- Deliverables
  - Safety-Relevant flag
  - Customer Input flag
  - Customer Output flag
  - AVL intern flag
  - Name
- Project milestones
  - Alias
  - Name
  - Deadline
  - Main milestone flag
- Reviews
  - Person
  - Path to review
  - Review-completed flag

The above artifacts are necessary for creating the process-based safety case in our tool environment. The project-specific tasks are all basic units which can be performed in order to fit the standard. Furthermore, each of these tasks contains an arbitrary number of deliverables that shall be created during the execution of the task. Deliverables are further linked to different project milestones. These traces are the preconditions for creating reviews. For a review, an arbitrary number of persons can be linked to a combination of deliverables and project milestones.

These persons shall perform a review up to the time defined by the project milestone with an externally provided tool. After the creation of a review, the tool allows the storage of the path to this review and furthermore the adding of a description. Furthermore, for each reviewed deliverable, a version and deliverable type can be inserted. After the review is completed, it can be marked as completed within the safety management tool. Therefore, it can be ensured that the necessary steps in the safety lifecycle have been performed.

After the reviews have been integrated in the tool, the report generator can be used to extract the process-based safety case. It contains tables that show the above listed artifacts and their traces between them to ensure that all necessary tasks have been performed in order to fit the standard.

For creating a process-based safety case, all the above artifacts and their connections are needed which can be created and maintained within the provided safety management tool.

3.5. Software Architecture

The software architecture of the developed tool consists of the different features (Figure 3.6) which are described in the aforementioned sections. In this section, the associated software architectures of the integrated features will be depicted. The section starts with a short overview of the client software architecture. Then, the software architecture of the server with the integrated ORM and the database architecture will be considered.

3.5.1. Client Software Architecture

The provided desktop application is developed with WPF (3.2.1) and is based on a "Model-View-View-Model (MVVM)" pattern. The concept of this pattern is shown in Figure 3.9 and contains three areas. It strictly disconnects the views from their associated business logic defined in the source code. Therefore, the view has knowledge of the view-model and, furthermore, this view-model contains an instance of the business logic model. Whereas the view-model and the view are connected via data bindings (binding, commands, or events), the view and the model are not connected.

The main benefit by using this pattern is that the view does not additionally need to be tested because of the decoupling. Therefore, the model that contains the business logic can be tested alone with different testing methods like unit testing.

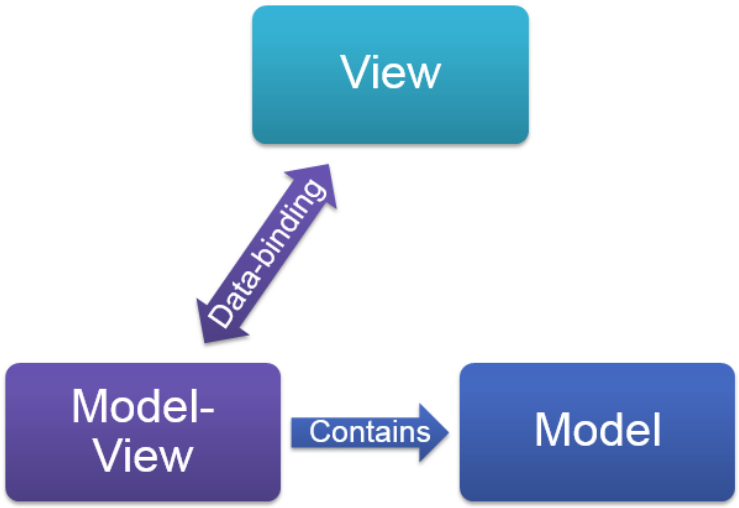

Figure 3.9.: Overview of the Model-View-View-Model pattern and its connections

The client provides the full graphical user interface and integrates the two different WCF services. As described in 3.4.1, the architecture of the client application is based on two different parts. Whereas the first part is responsible for the standard-specific artifacts, the second one is responsible for the project-specific artifacts. For each part, a specific graphical user interface is created (4.2). Within these views, a user is able to access the associated modules. Therefore, for each defined part, a particular view landscape consisting of different modules is created.

Furthermore, the client application is based on a project workflow mechanism. For each defined standard one template project has to be created at the beginning. This template project can then be used to establish new projects (3.3.2). The benefits of this approach are that the reusability and the consistency can be improved and the generation of additional artifacts can be avoided. The provided project workflow framework is illustrated in Figure 3.10. It can be seen that for each defined standard a template project shall be created at the beginning. From this template project, new projects can be developed. Furthermore, for the created projects it is possible to create associated sub-projects as well as baselines of an arbitrary state of the project. Now, a powerful mechanism inside the architecture effects that each created project can be treated as template project, which allows the creation of similar projects in a short time and it helps promote the consistency and completeness of projects. Because sub-projects can have the same deliverables and project milestones as the main project, the database does not contain redundant information.

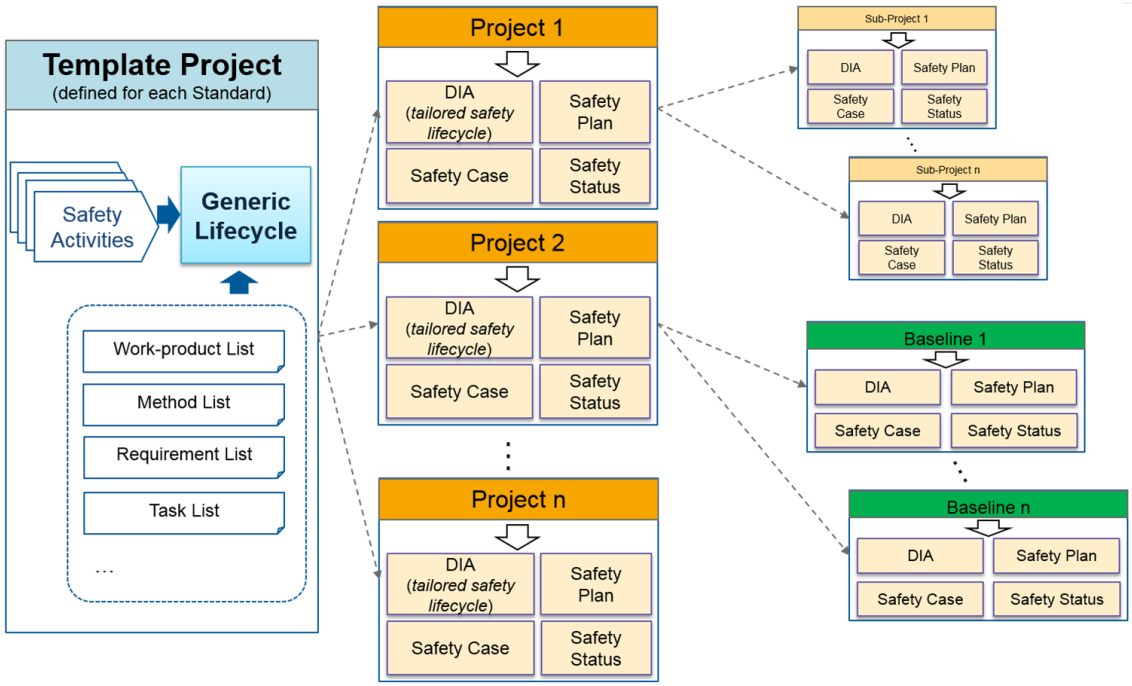

Figure 3.10.: The project workflow starts with creating a generic standard and continues by creating template projects. These projects can be used for creating further (sub-)projects or project baselines.

3.5.2. Server Software Architecture

The server consists of five collaborating libraries which are displayed in Figure 3.11. The main library is the "FSMServer" library. It contains both WCF service configurations and the associated authentication and authorization classes.

The "FSMServiceContracts" library defines an interface for the existing WCF service methods. The according implementation of these methods takes place in the "FSMServiceImplementations" library. In addition, this library contains the object relational session manager which is responsible for creating and closing the different sessions.

The fourth provided library is the "NHibernateHelper" library. It contains the ORM library and stores all needed database and entity configurations and provides it provides the functionality for accessing and retrieving data from the database. Furthermore, it includes the library for the data tracking and is therefore also responsible for checking data changes and storing them in the defined database tables. This library and an overview of the associated architecture is provided in 3.5.2.

The last library is the unit test library which contains the testing framework. This library is not included in Figure 3.11 because it is connected to all other libraries. A more detailed overview of the testing framework can be found in 4.3.

The aforementioned libraries are connected with each other and represent the entire server architecture. For achieving a working architecture, an additional database system and database architecture is needed, which will further be explained in 3.5.2.

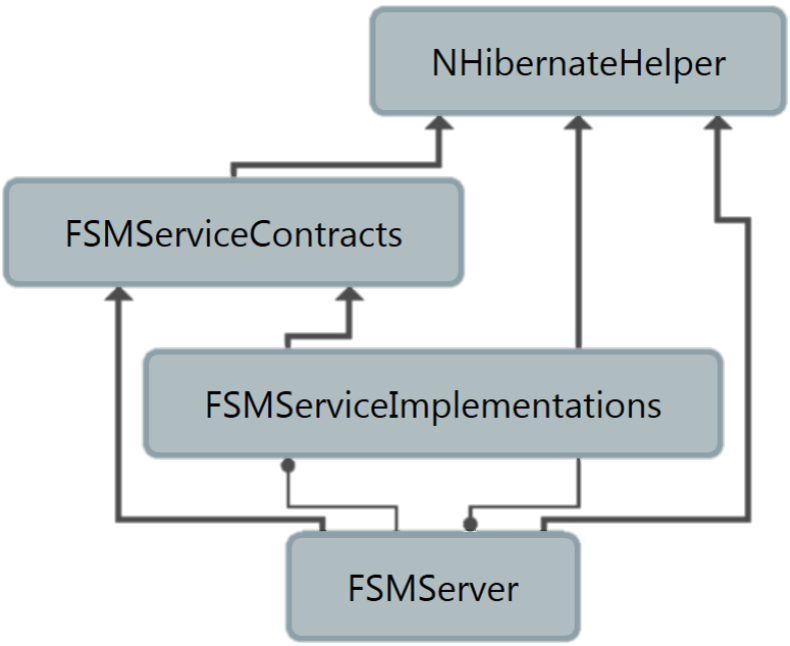

Figure 3.11.: Dependency graph of the existing software libraries within the server architecture

ORM Software Architecture

The object relational mapping library is the heart of the provided server architecture and is responsible for the entire database interaction. It consists of all table entities and the associated entity configuration files. Furthermore, it contains the database configuration and the configuration for the data tracking library. The entity configurations are needed in order to establish a valid connection between the database tables and the associated source code objects. An example configuration is displayed in Appendix D. The database table fields and the class properties must also be mapped and configured in detail to have a working connection.

Figure 3.12 shows a cutout of the integrated database architecture as well as the structure of the generic part and consists of 16 different database tables. These tables also exist in the object relational mapping architecture as separate classes. All of these classes are derived from one base class that contains the ID of the entity. ID is the only database field that exists through all different entities and can therefore be stored in a base class. The following list illustrates the 16 existing tables. For each table a consistent entity class is created which stores all table properties as public class properties.

- Cluster
- Activity Cluster Mapping
- Activity
- Task Activity Mapping
- Task
- Task Input Task Mapping
- Method
- Method Task Mapping
- Workproduct
- Workproduct Task Mapping
- Requirement
- Requirement Task Mapping
- Requirement Safety Level
- Deliverable Workproduct Mapping
- Degree of Recommendation
- Method Attributes

The tables above consist of entity tables and mapping tables. The mapping tables are responsible for mapping two different entities. In the tool architecture, specific classes for these mappings are created. One mapping class instance therefore contains one source entity and one target entity. This mechanism is used for many-to-many mappings and provides benefits. One benefit is that the source and target entity contains a list of these mapping classes and it is really easy to check whether an entity is connected to another entity. On the source code layer, this mechanism facilitates the control of the relations. Additionally, it is also possible to track changes on the mapping tables. Therefore, all relations will be saved and historical relations can be retrieved from the database.

Database Architecture

The last section of the design chapter describes the developed database architecture. A cutout of the provided database schema is depicted in Figure 3.12. The diagram is developed within an EER diagram (Appendix C) and contains all necessary entity tables and mapping tables. For each entity, a specific table is created and the appropriate properties are added. What is more, for each many-to-many relation an individual table is created which maps two entities. Furthermore, the database architecture is integrated into two different databases as described in 3.2.3. The main database architecture contains of four different parts which will briefly be described in the following.

Standard Part

This part contains a standard table for defining different standards with their appropriate attributes. Additionally, mapping tables between the standard and the standard-specific artifacts exist in this part. For the requirement, method, workproduct, cluster, activity, task and requirement safety level artifacts own mapping tables are created. These tables are only responsible for storing detailed information of specific standards and of their relations to the standard-specific artifacts which are listed in 3.5.2.

Generic Standard-Specific Part

The generic part of the database architecture consists of the standard-specific artifacts and their attributes. Therefore, it contains all tables that describe a specific standard and consists of methods, requirements, tasks, workproducts and the associated safety lifecycle artifacts. This part is illustrated further in Figure 3.12.

Project-Specific Part

The last part contains all relevant artifacts that are needed for creating and maintaining different projects in the tool as well as the mapping tables that link a project to a specific standard. Furthermore, it contains all project-specific tables and their appropriate attributes. These tables store artifacts that allow the creation and storage of project-specific data and their relations. The project-specific part contains tables for project-specific tasks, project milestones, deliverables, project roles, project methods and reviews.

General Part

The general part contains all other tables that are needed for the user management, for baselining and for providing automated text modules like rationales. These text modules are used for providing an auto-completion of input text fields. The user management tables define the person who is able to access the tool with his or her AVL credentials. In addition, the mapped user permission will be saved and the tables save all necessary values for restoring historical data.

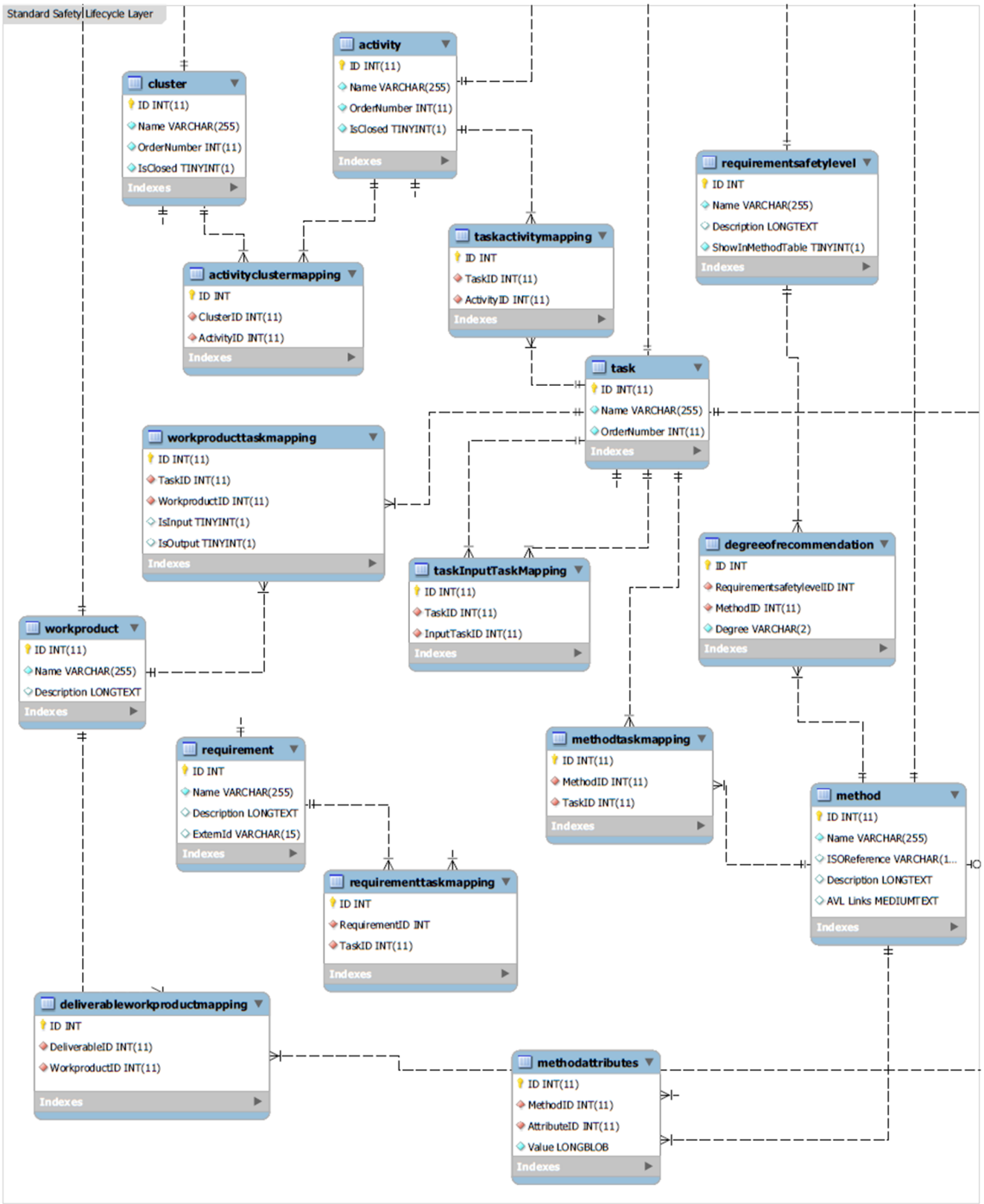

Figure 3.12.: Cutout of the database architecture that shows the generic standard-specific part. This part contains all necessary artifacts for defining a specific standard within the tool.

4. Implementation of the Functional Safety Management Tool

This chapter describes detailed information about the used tool-chain and the implementation of the different safety management features of the design phase. Additionally, it displays the graphical user interfaces that were created within the tool environment and the appropriate application. At the end of the chapter, the established testing framework will be depicted in detail.

4.1. External Tools and Libraries

4.1.1. Tool-Chain

For establishing a working environment between a huge number of modules, a tool-chain is needed. The used tool-chain consists of two external tools. The first one is the "Visual Studio" (4.1.1) tool, which is a powerful development framework provided by Microsoft. For the development of this safety management tool architecture, the Visual Studio 2013 professional version is used. The second tool is the "MySQL Workbench". It is a graphical database management tool that allows the creation and handling of database tables and their according relations. These two external tools are necessary for the development of the tool environment.

Furthermore, the tool-chain contains of two external libraries. These libraries are integrated into the visual studio tool and are necessary for the development of powerful graphical user interfaces and for the data exchange between client and server. Infragistics (3.2.4) is the first used external library and provides a lot of different powerful graphical elements. These elements already offer all needed and helpful features like filtering, sorting, grouping and many other. The second library is an "Object Relational Mapping (ORM)" library. This library is responsible for creating and configuring the database interaction as well as the data exchange between server and database and automatically establishes the needed database statements.

Visual Studio

This external tool[1] is developed by Microsoft and provides a development environment framework for creating different applications like WPF or WCF. It allows the creation of graphical user interfaces with simple drag and drop elements from a provided toolbox. By dropping the graphical element into the view, the associated source code is created automatically. Therefore, the tool is on the one hand a designer tool and allows on the other hand the direct implementation of the appropriate business logic.

[1] `https://www.visualstudio.com/`

MySQL Workbench

The MySQL workbench tool is the second external tool. It provides a graphical user interface for the database creation and database handling. Additionally, it allows an external interaction with the database and contains different views on the database as well as a graphical extension named EER diagram (Appendix C). This extension allows the graphical interaction with a database schema and the appropriate database table generation. Inside this diagram, it is also possible to create and maintain the different table relations as it is also possible to query the database within this extension.

4.2. Implementation of the collaborative Safety Management Tool Environment

This section describes the implemented safety management tool and the entailed features. In the following, the different parts and the associated modules will be depicted in detail. First, the standard-specific part will be examined by illustrating the appropriate modules and explaining the included features.

The goal is to show how a standard can be integrated and how the associated safety lifecycle can be created. After the standard has been integrated into the database, the project-specific part will be depicted in more detail. It will be illustrated how a new project can be created and how the project-specific modules can be accessed. Furthermore the creation of project-specific artifacts will be shown as well as how the two provided reports (4.2.3) can be generated within the provided tool.

The developed tool exists as simple standalone application, which is why it is not necessary to install the tool. The tool can simply be started by double-clicking on the provided executable. One very important thing in this context is that the tool only works with a valid internet connection and with a valid connection to the AVL infrastructure. This means it can only be used within the AVL company or with a connected “Virtual Private Network (VPN)” tunnel.

After the tool start-up, the logon window appears which is shown in Figure 4.1. By entering the AVL username and AVL password, a user can be logged in within the functional safety management tool. After the user has been authenticated and authorized by the tool environment, the main screen (Figure 4.2) appears. This screen contains all existing projects and sub-projects that are accessible by the user and his or her associated right. Is the user authorized as global admin, he or she is now also able to open the standard-specific part of the tool by clicking on “Standard|Standard View” in the menu bar. If he or she is not authorized as global admin, he or she is not able to see this menu bar.

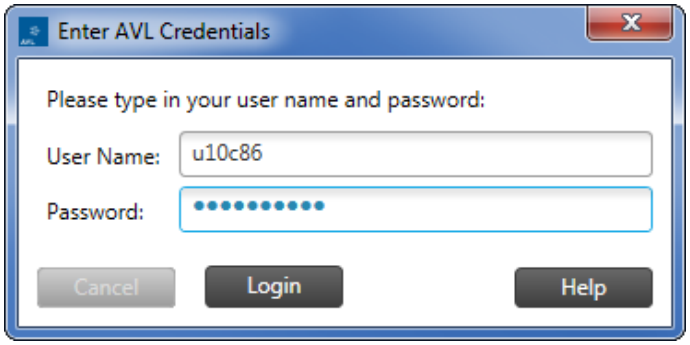

Figure 4.1.: Login window of the Functional Safety Management Tool

The main screen may display the list of projects in different ways as illustrated in Figure 4.3. Optionally, each user can choose based on his or her fondness a particular view of the representation of the projects. Furthermore, the screen shows the main properties of each project and sub-project. It is an interactive view and, therefore, a project can be opened by double-clicking on a specific project. In the following, the standard-specific part will be depicted first and then the project-specific part. As mentioned before, this part is only accessible by a global admin.

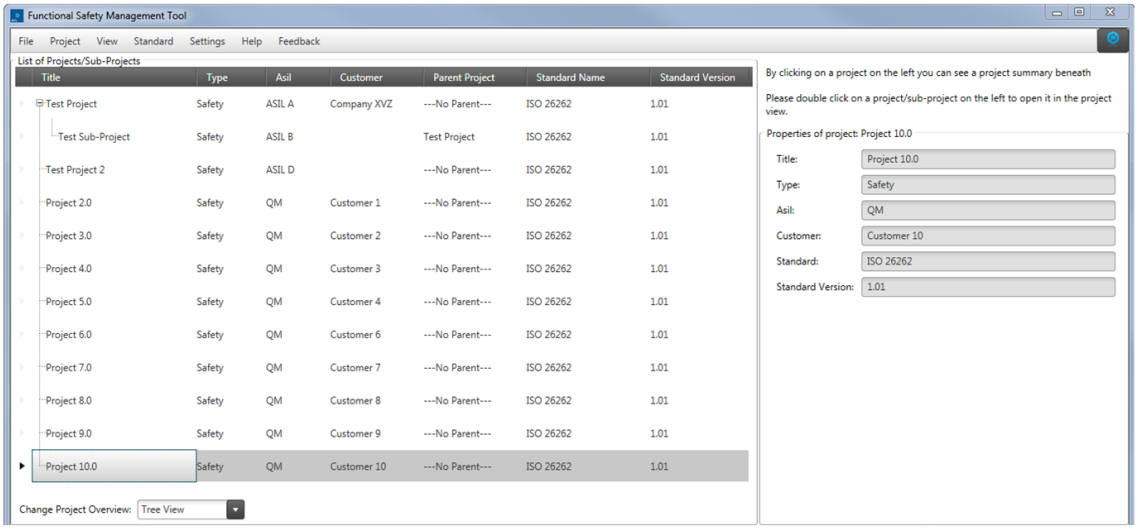

Figure 4.2.: After the start of the tool this view appears and lists all accessible projects. Additionally, the project details are shown on the right side.

Figure 4.3.: A user can choose between four different views for listing all projects. The figure shows all four views which are a tree view, a carousel view, a card view and a grid view.

4.2.1. Standard-Specific Part

By opening the standard view, the standard-specific part with the generic artifacts of the tool can be accessed. This view is illustrated in Figure 4.4 and contains a list of all standards which exist in the database. Each standard can be selected and shows the associated information on the right side of the view. This information contains the name, the version, the associated requirement safety levels and the description of the selected standard. At the bottom right corner, a carousel view that shows the provided generic workflow for a standard is displayed. It can be used to access the different modules that are integrated into the standard-specific part. These modules are relevant for creating and maintaining the generic parts of the standard. The carousel view contains five buttons that can be selected in order to access the different modules (3.5.1). The carousel view additionally provides a navigation bar that allows scrolling within the different workflow steps. In the following, these different modules with their associated views will be explained.

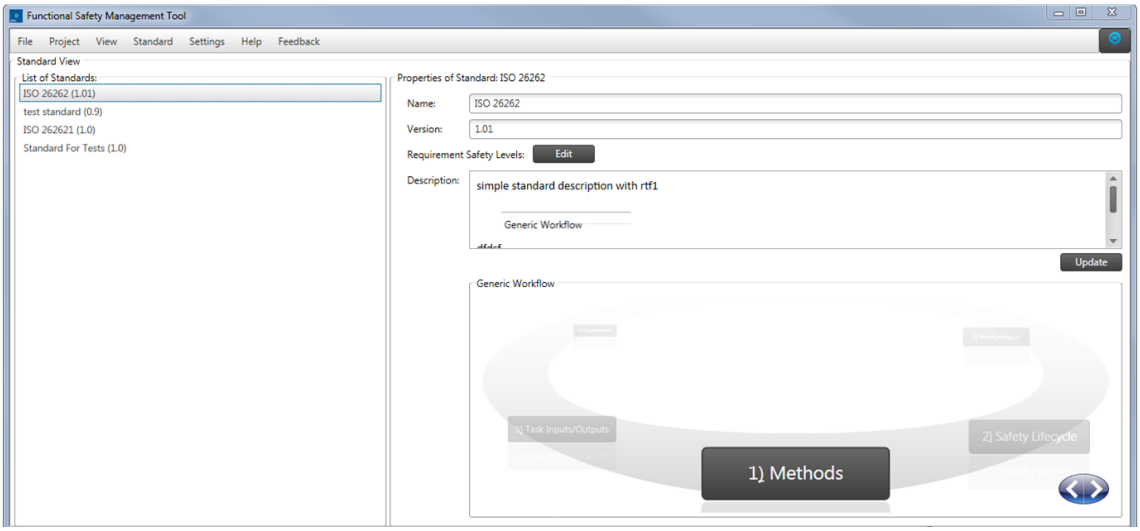

Figure 4.4.: The provided standard view lists all existing standards with their appropriate artifacts and the associated generic workflow. The workflow contains buttons which can be clicked for accessing the different standard modules.

Methods Module

The methods view is the first module that can be accessed within the carousel view from the standard view. This view allows the creation of different methods that exist in a standard. The tool provides an input field for the method name, the method description, the ISO reference which points to the location where the method can be found in the standard and the AVL links field. The AVL links field contains the path to the specific AVL method description. Each standard method comprises a defined degree of recommendation which specify at which safety integrity level the method shall be applied. The tool therefore enables entering that degree of recommendations for the different methods. The method view is illustrated in Figure 4.5 and consists of a table which lists all existing methods.

The view allows the entering of new methods by clicking on the provided button. As a consequence, a new context menu appears where the information can be fed in.

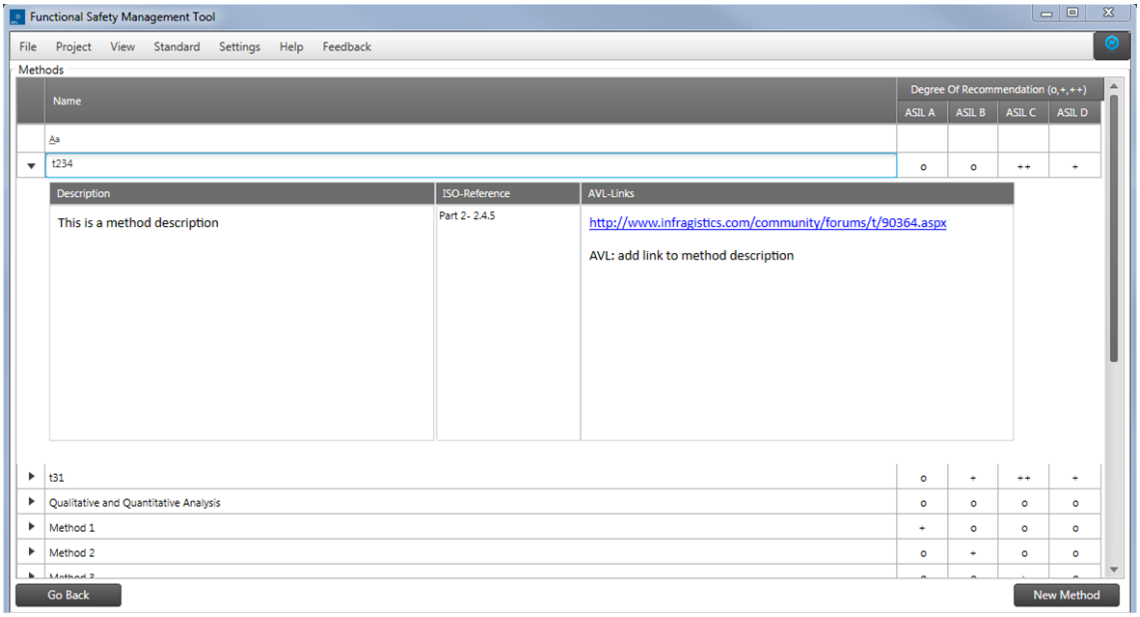

Figure 4.5.: The methods part contains all existing methods and corresponding information. Additionally, the degree of recommendations are displayed.

Workproducts Module

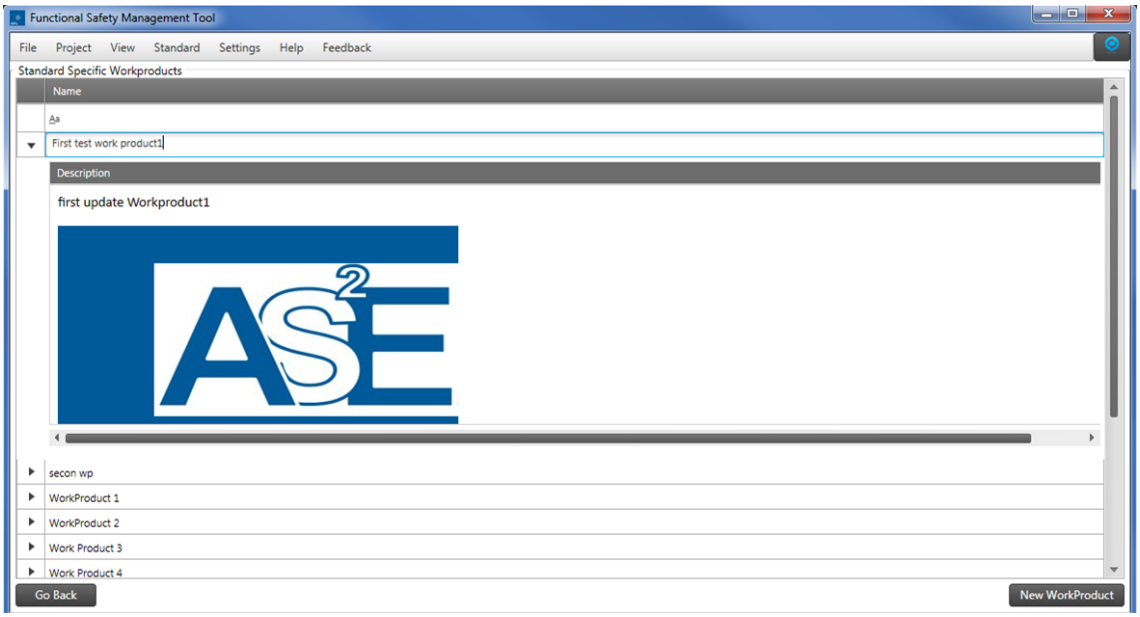

Figure 4.6.: The workproducts module lists all existing workproducts.

This view allows the creation of standard-specific workproducts with the appropriate information. These workproducts contain a specific name and an additional description. Figure 4.6 shows an overview of the workproducts view. In this view, it is possible to create new workproducts by clicking on the new workproduct button. The context menu that appears after clicking the button contains a text field for the name and a text field for entering a description.

Requirements Module

In the provided requirements module view, it is possible to create requirements that are defined in a standard. These requirements contain a name, description and an additional field where an external ID can be stored. This external ID can for example be the ID of the requirement from an external requirements management tool. Furthermore, the view allows the connection between requirements and tasks. Therefore, a tree with all generic tasks is shown on the right side. By dragging and dropping a requirement from the left requirements table to the right task tree a new mapping between these two items can be created.

The requirements view is depicted in Figure 4.7, which additionally shows the dependency feature. Furthermore, this view allows the creation of new requirements by clicking on the new requirement button. By doing so a predefined context menu that allows the insertion of requirement-specific information appears. As an additional feature, it is also possible to show all requirement mappings within the task tree. These mappings are highlighted in the tree as illustrated in Figure 4.7.

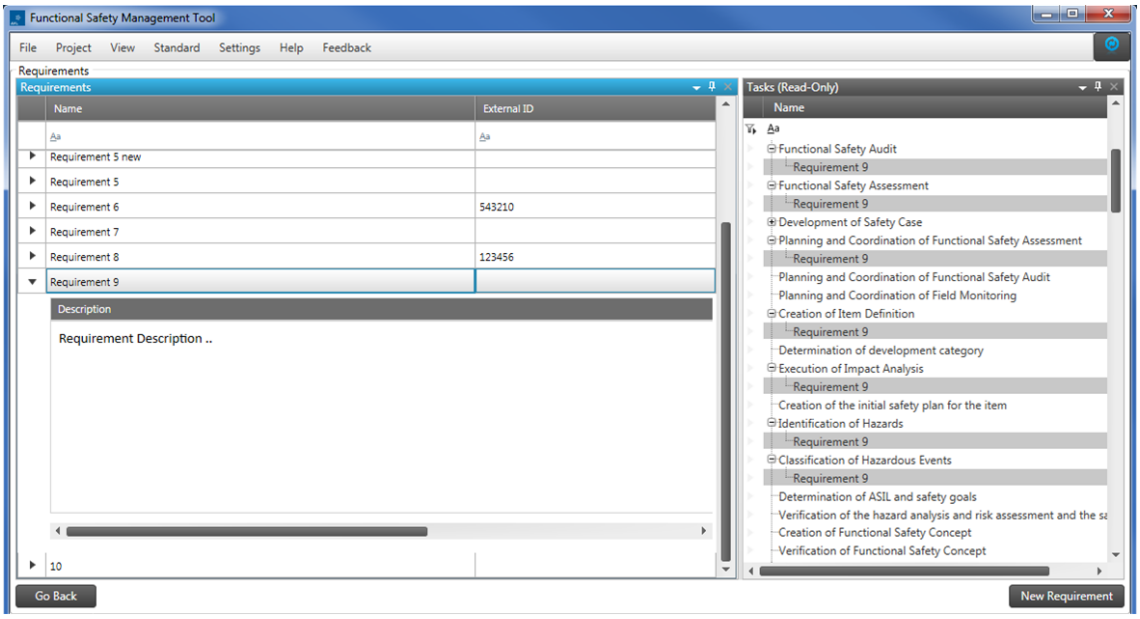

Figure 4.7.: The requirements module lists all existing requirements with their descriptions and links to external requirements. Furthermore, the mappings to the different standard-specific tasks are shown on the right side.

Task Input Module

Within the task input view, it is possible to connect the existing workproducts to specific tasks. These workproducts can be defined either as input or output workproducts by dragging and dropping a workproduct from the right side into the boxes in the middle of the view. The input workproducts of a specific task are listed in the upper center of the view. If the input workproducts are shown because of connected input tasks they have a different background color. Output tasks are listed in the lower center of the view and the box is marked with a different color.

Additionally, it is possible to define input tasks for all existing tasks. These tasks are listed in the middle of the view. For defining an input task, the target task should be selected and the input task can be dragged into the box in the middle of the view. By defining an input task for a specific task, all associated output workproducts are automatically copied to the target task as input workproducts. Therefore, complex dependency graphs can be created. The view is illustrated in Figure 4.8.

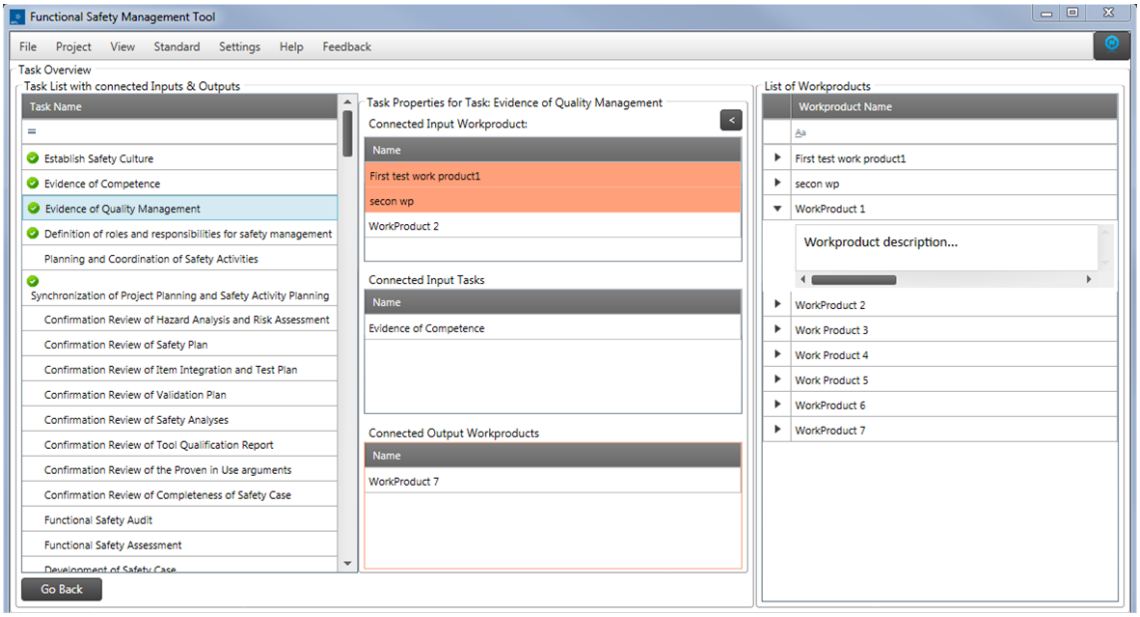

Figure 4.8.: The task input module shows all existing standard-specific tasks with the connected input and output workproducts. Furthermore, it also displays the connected input tasks. If a task has a linked output workproduct a green tick appears.

Safety Lifecycle Module

The last provided module allows the creation of the standard-specific lifecycle. The lifecycle can be built up with three different elements which can be linked together. The first element is a task. A task is a basic unit that can be performed by a person or an organization. Similar tasks can be grouped as activities. An activity is the second element that can be created within the lifecycle. Furthermore, it is also possible to group activities as clusters. The third element in the safety lifecycle is a cluster. These elements can be built up as a tree; an example lifecycle can be found in Figure 4.9. For providing a better overview, three different trees exist in the view. The first tree contains the entire safety lifecycle tree which comprises all clusters with their connected activities and all activities with their connected tasks. The second tree only contains all activities with their connected tasks while the last tree contains only the list of all existing tasks. Each tree is interactive, and therefore, the trees are synchronized. Updating an element within one tree automatically updates the according element within the other trees.

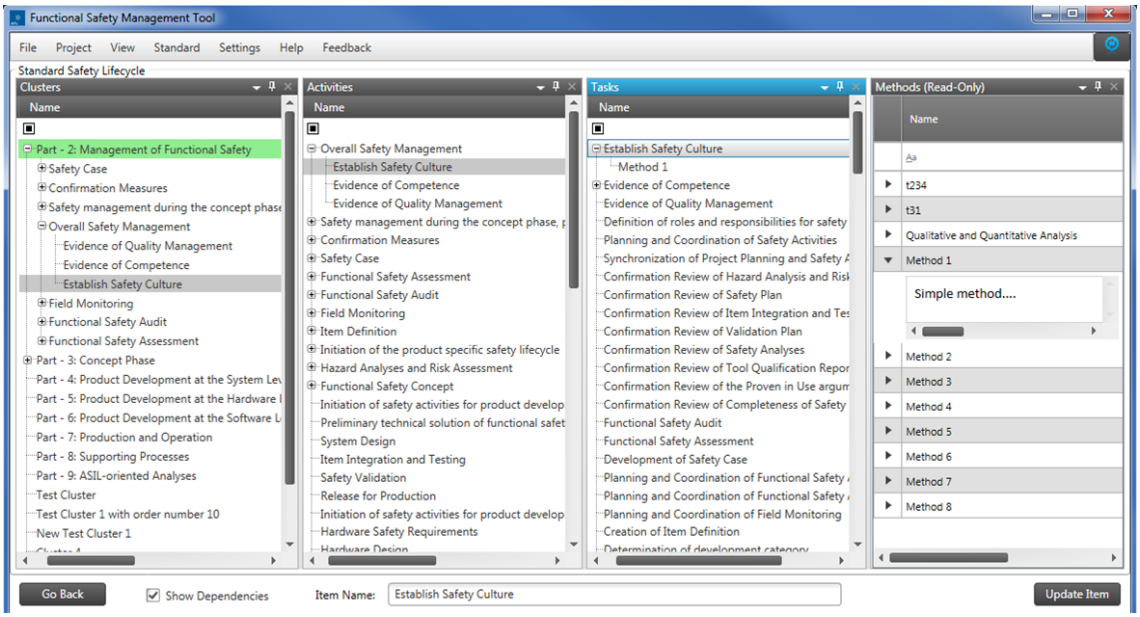

Figure 4.9.: This module represents the entire safety lifecycle and allows the viewing of the dependencies between the different artifacts.

An additional feature in this view is that it is possible to map methods to different tasks because it is a complex view it is possible to hide own trees within the view. Furthermore, to see which elements are connected, the “show dependency” checkbox can be selected. If the checkbox is selected and an element in any of the trees is selected, all equal elements in the other trees will be selected automatically and the dependencies between all elements can be seen. Additionally, it is possible to view all method task mappings in the task tree.

The aforementioned views are responsible for the creation of the generic standard-specific artifacts and are only accessible by a global admin. The second implemented part contains the project-specific modules. By double-clicking on a project in the main view (4.2), the project-specific part can be opened.

Figure 4.2 shows the main view of the project-specific part. It contains all project properties like the tile of the project, the type of the project, which standard the project belongs to, a project-dependent ASIL, the customer for the project and the project description. Furthermore, all sub-projects are listed in the right upper corner of the view. On the right lower corner, the carousel view which contains all project-specific modules is shown.

4.2.2. Project-Specific Part

By creating a new project it is mandatory to select an existing standard and enter some project-specific information or an existing project. During the creation of a new project, a lot of artifacts will be generated automatically. First, for all existing standard-specific tasks, project-specific tasks will be created and linked to the generic tasks. These links are very important in order to see which project-specific tasks belong to which standard tasks. It is possible to create a new project-specific task which then can be linked to a standard-specific task. Now, both project-specific tasks are linked to the same standard-specific task.

Furthermore, all existing workproducts will be instantiated as project-specific deliverables. Also for these artifacts the traces remain. Because the workproducts are already linked to the standard tasks, the deliverables are automatically linked to the project-specific tasks too. After creating a new project deliverable, it can be linked to a standard-specific workproduct. Therefore, the deliverable will automatically be linked to the project-specific task.

At last the same mechanism is integrated for the methods. Therefore, all standard methods are instantiated and appropriate project-specific methods will be created. Additionally, the links between the tasks and the methods will automatically be created in the project.

Through the creation of a new project, numerous links between the different artifacts will automatically be created and, hence, no user needs to be worry about them. By the automatic creation of such a huge number of complex relations, a lot of time can be saved.

Figure 4.10 illustrates a list of all existing sub-projects in the right upper corner of the view. A sub-project can be accessed and opened by double-clicking on it. In the right bottom corner of the project view, a carousel view displaying the project-specific workflow is shown. Similar to the generic workflow in the standard view, the project workflow contains different buttons for accessing the project-specific modules. The modules are listed in 3.4.1 and described further in the following. Additionally, on the left side, all project-relevant information is given and the two reports can directly be created within this view.

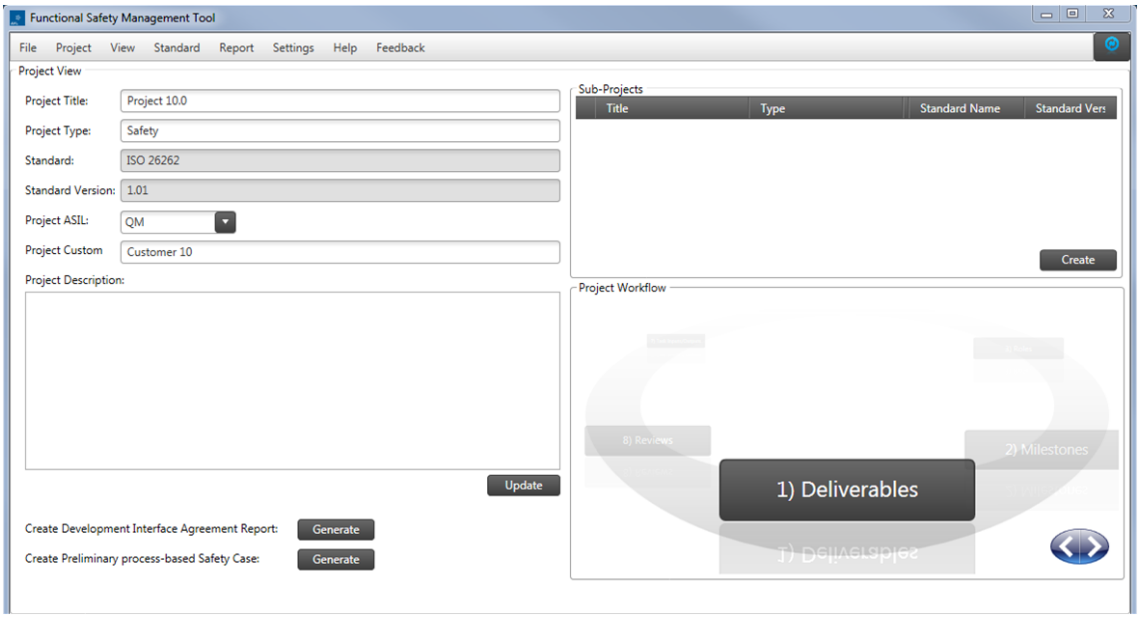

Figure 4.10.: Project view that shows all project-relevant information and provides the project-specific workflow. Additionally, it shows the list of existing sub-projects.

Deliverable Module

The deliverable view contains all existing deliverables that are necessary within a project. As mentioned before, they will automatically be established during the project creation. It is not necessary to manually connect the created deliverables to the project-specific tasks because these links will also be created automatically. The view is depicted in Figure 4.11 showing on the left side a tree which lists all existing generic workproducts with the connected project-specific deliverables. On the right side, the deliverable table is shown. This table contains all existing project deliverables and their associated information like a name or whether the deliverable is either an AVL input deliverable or an external deliverable. An external deliverable can either be sent to the customer or received from the customer. An AVL input deliverable means that the deliverable will not be sent to the customer and is therefore only for internal usage. Additionally, a deliverable can be set as safety-relevant deliverable but does not have to. Furthermore, a deliverable contains a description field and a field where the path to the deliverable can be inserted. For the deliverables, also a completed flag exists which indicates whether the deliverable is already available and reviewed. Besides the linkage feature, this view allows the creation of new deliverables. If two technical safety concepts are needed in a specific project, only one will automatically be created during the project creation. Now, a new deliverable shall be created with the appropriate name. The new deliverable shall then be linked to the according workproduct by dragging and dropping the deliverable from the right table into the left tree. After that, two deliverables are linked to one generic workproduct. It is also possible to create a deliverable that is not linked to any workproduct. In this case, the

deliverable must manually be linked to an associated project task (Figure 4.14).

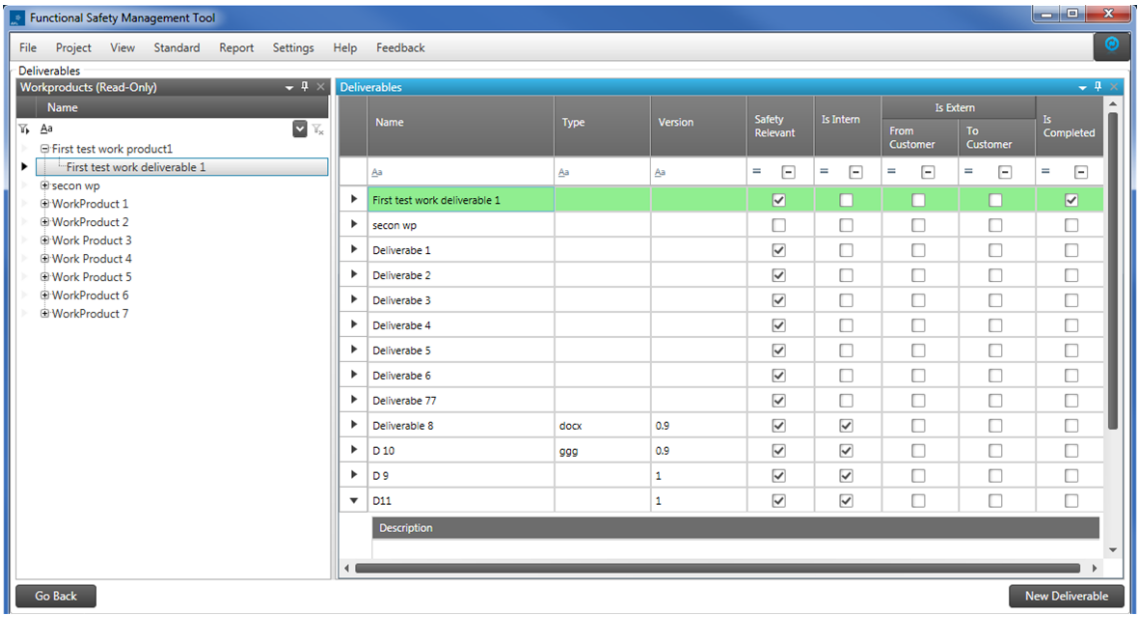

Figure 4.11.: The deliverable module allows the creation of project-specific deliverables with their appropriate information and the linkage to workproducts.

Project Milestone Module

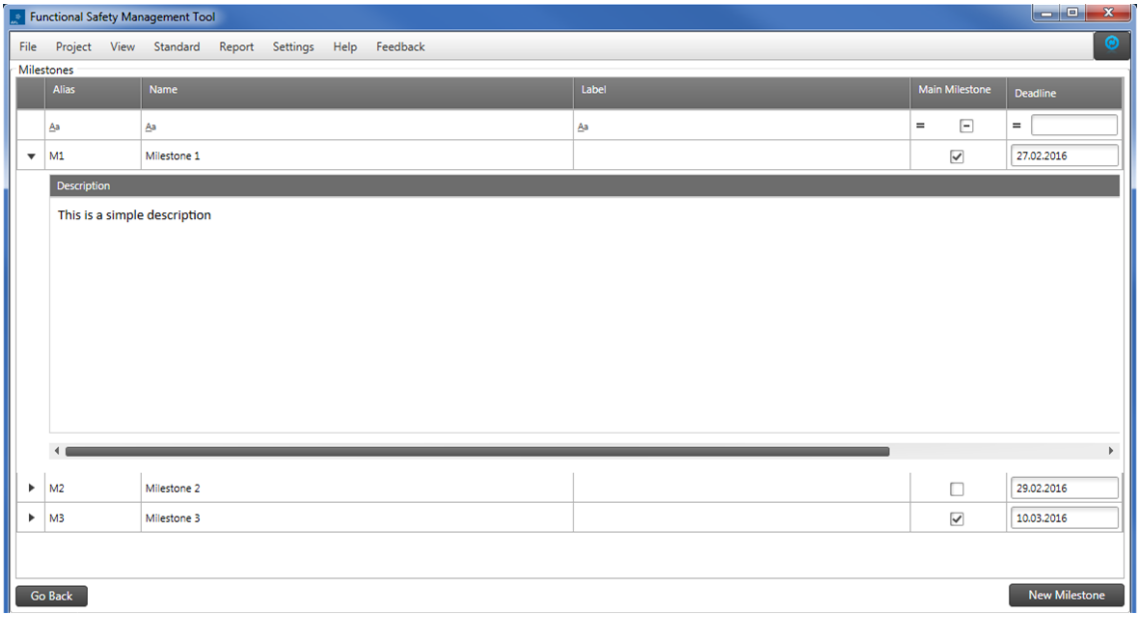

Figure 4.12.: The project milestone module shows all existing milestones with their corresponding information.

For defining project-specific milestones, this view can be used. It contains a table that lists all existing project milestones with the appropriate properties. These properties are the name of the milestone, the alias, an arbitrary label, the description and the deadline of the milestone. It is also possible to define each project milestone as main milestone. In this case this milestone is processed as master milestone and displayed in the diagram interface agreement. These main milestones can be linked to project-specific tasks and then defined for each mapping responsibilities between the AVL and the customer.

The project milestone table is depicted in Figure 4.12. The functionality for adding and removing milestones is also integrated into this view. A new milestone can be created by clicking on the defined new milestone button.

Deliverable Responsibility Module

The deliverable responsibility view is suitable for linking the existing project milestones to different deliverables. The view is illustrated in 4.13 and shows on the left side the deliverable tree with the connected project milestones and on the right side the existing project milestone list. By dragging and dropping a milestone from the right tree into the left tree, a specific project milestone can be linked to the target deliverable.

After the creation of the mapping, it is also possible to enter mapping specific values. Hence, for each mapping between a deliverable and a project milestone an AVL responsibility and customer responsibility can be selected. Furthermore, another field for entering information for deliverables which is sent to the customer exists.

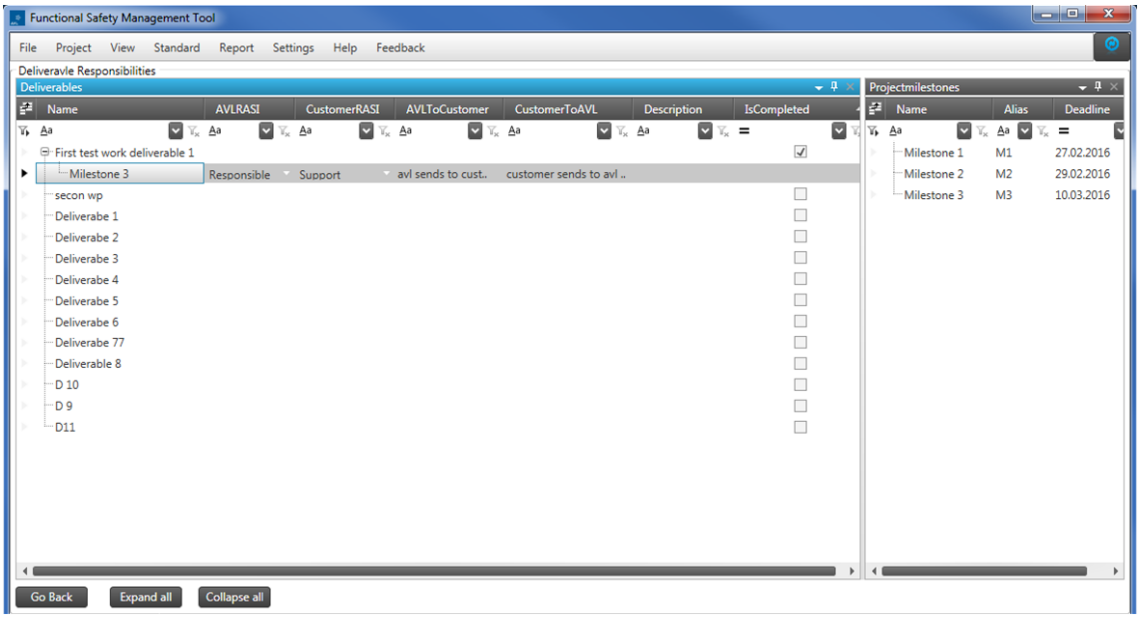

Figure 4.13.: In this module project milestones can be linked to deliverables and furthermore for these mappings responsibilities can be defined.

In addition, a field for entering information for deliverables which are retrieved from customers exist too. It is also possible to add a description for each single mapping. Finally, a mapping contains additional fields in which the version, the type and the path of the deliverable for the linked project milestone can be stored.

Project Task Input Module

This view is very similar to the task input view (Figure 4.8) of the generic standard-specific part. The difference is that within this view only project-specific artifacts are handled. As it can be seen in Figure 4.14, the left list contains all project-specific tasks and at the right side, all existing project deliverables are listed. Both lists are read-only lists, and in this view, it is only possible to create traces between the artifacts. In contrast, the view of the standard part contains the generic tasks and generic workproducts.

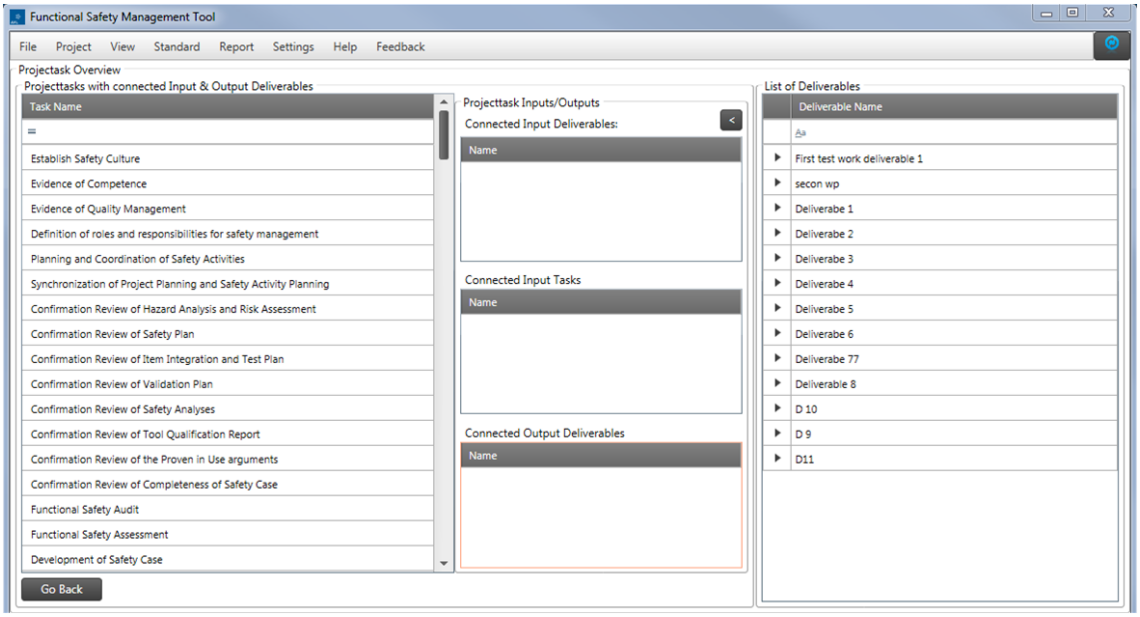

Figure 4.14.: This module allows the linkage of project-specific tasks and input or output deliverables to project-specific tasks.

As mentioned above, if a deliverable has no connected generic workproduct, it must manually be linked to a project-specific task, which can be done in this view. The view moreover shows the already connected deliverables which are developed during the project creation.

A deliverable can be linked either as input or output deliverable to any existing project-specific task. Additionally, it is also possible to define input tasks for project-specific tasks. This is only possible for tasks which are not linked to a standard-specific task because the standard-specific task already contains this connection within the standard. The generic connections between tasks cannot be updated in the project context.

Project Role Module

In the project role view (Figure 4.15) it is possible to create project-specific person role mappings. Only the global admin or the project admin is able to create these artifacts. First, the different organizations can be created in this view. These organizations are global organizations and therefore visible throughout all projects. Within the view it is also possible to create persons. Additionally, these persons are visible throughout all projects and therefore global. Such a person has a name field, an organization field and a description field. The organization can be selected from the above defined list and is hence linked to the person.

Additionally, it is possible to create project-specific roles like a Functional Safety Manager or a Technical Safety Developer. These roles can also have a description. These roles are global as well and exist thus for all projects. Now, if all persons and roles of the current project exist, they can be linked.

In the left table of the view, the project-specific mappings between the persons and roles can be created. A new mapping can be created by selecting a specific person in the provided drop-down list. In addition, a role must be selected from the other provided drop-down list. By clicking the "Add Mapping" button the new project-specific mapping is created. One important thing to mention is that one person can also have different roles in the project. Therefore, for each mapping, a new table entry shall be created. As mentioned before, the list of organizations, persons and roles is visible throughout all different projects and just the mapping of them is project-specific and only visible within one project.

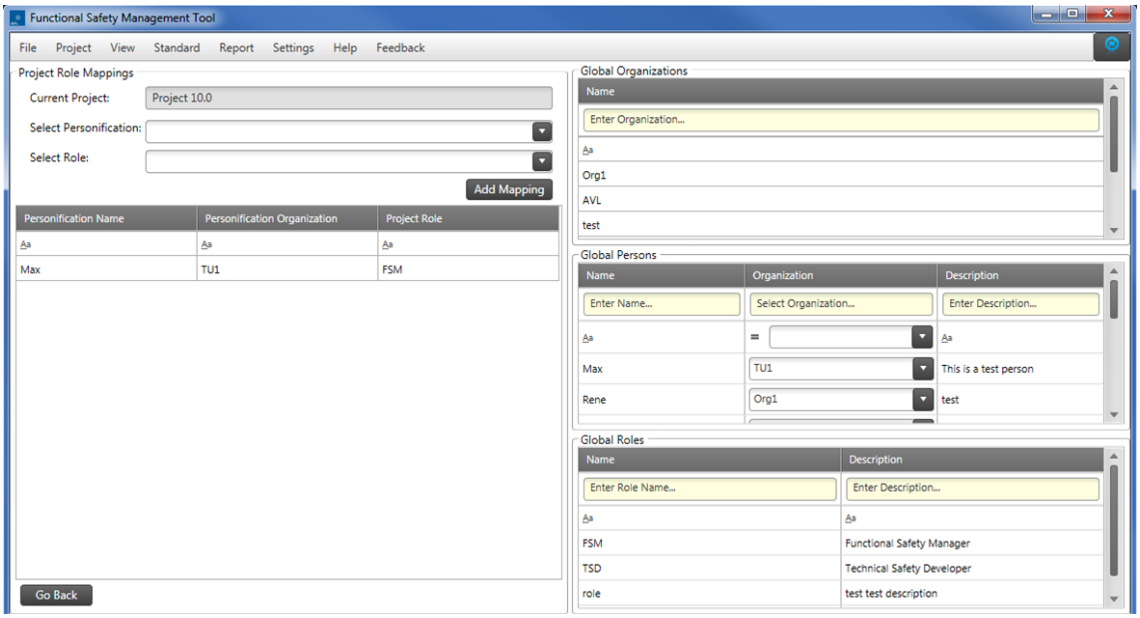

Figure 4.15.: In this module global organizations, persons and project roles can be created. Furthermore, it is possible to create a project-specific mapping of persons and roles.

Tailoring Module

Tailoring the different project-specific tasks is an important step in the safety management process. It allows the selection and deselection of different project-specific tasks for the current project. The according view is illustrated in Figure 4.16. If a task is disabled for the current project, a well-argued rationale shall be inserted. Furthermore, it is possible to also disable entire groups of tasks by disabling the associated activity and to disable whole clusters. The rationale that must be inserted by disabling an element can be reused by all existing rationales. Disabling an element opens a pop-up window that allows entering a rationale which is automatically recommended based on already existing rationales. Furthermore, it is also possible to select existing rationales from the list that is shown in the pop-up window. By disabling an activity or a cluster, the rationale will be linked to the selected activity or cluster and not to the single project-specific tasks. The view is constructed as a tree which contains the structure defined by the safety lifecycle. The first step in this view is now to tailor the specific project tasks before additional properties can be inserted for these tasks. If an item is disabled, a red cross is shown on the left side. Additionally, the foreground of the font will turn gray. If the item is enabled, a green tick appears and the existing properties can be updated.

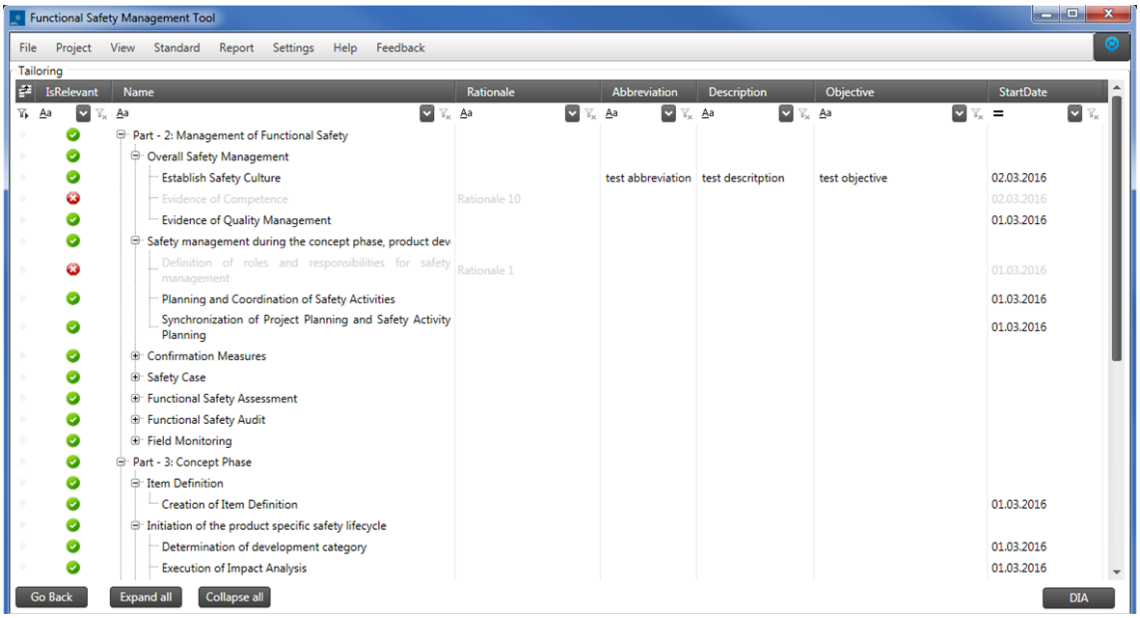

Figure 4.16.: This module allows the tailoring of the existing project-specific tasks and the insertion of additional task specific information.

The view additionally allows the creation of new project-specific tasks. These tasks are all together grouped in a particularly defined activity container which is also grouped within a separately defined cluster container. If a new project-specific task is created, the provided second name field can be used to enter a specific name. The standard name field is read-only and the name is inherited from the generic standard-specific tasks.

DIA Module

After the tailoring of the project-specific tasks, the development interface agreement can be created. This view is depicted in 4.17 and contains a table that represents the safety lifecycle which consists of all relevant project-specific tasks. Furthermore, the table shows all connected clusters and activities at the beginning of each row. Additionally, connected input and output deliverables are stored in separate columns in the table.

In this view it is possible to select the responsibilities for a project-specific task and project main milestone mapping. For each existing main milestone, a group column which contains two child columns exists. Each of these group columns contain one column for the AVL responsibility and one column for the customer responsibility. The responsibilities are provided as drop-down lists and are based on the AVL RASI matrix (2.1.2).

For each project main milestone a different RASI can be selected. There are also three additional columns in this table. The first one is the "Customer to AVL" column. For all existing main milestones, this column contains an individual row where additional information is provided. The same exists for the "AVL to Customer" column. Finally, a specific column for adding a description exists. All these columns are created dynamically.

The view cannot be updated directly. For updating the information for a specific mapping, an update window can be opened by right-clicking on a specific project task and selecting edit. If the development interface agreement is completed, it is possible to export the entire information with the provided report generator.

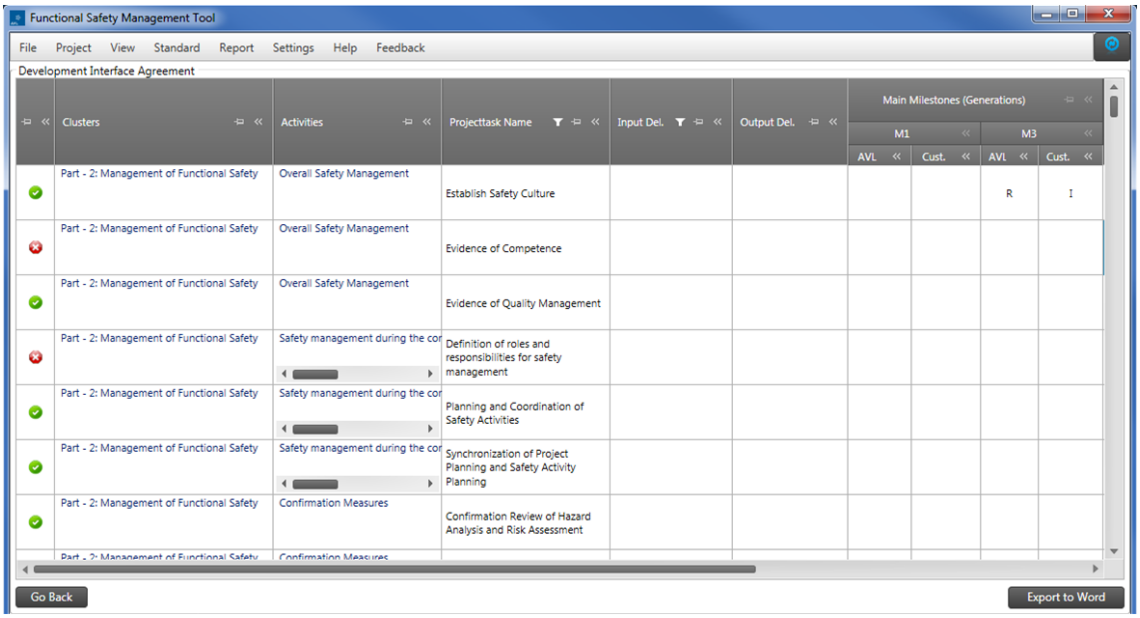

Figure 4.17.: This module allows the generation of a development interface agreement. This DIA shows the project-specific tasks with corresponding information. Furthermore, the linkage to the project main milestones exists and for these mappings responsibilities can be defined.

Review Module

The last provided module is responsible for creating and planning reviews. Reviews can be done on deliverables for all specific milestones. Figure 4.18 illustrated the existing view. It contains a table which lists all existing project deliverables. Furthermore, the table shows all connected project milestones. For each of these connected project milestones, an arbitrary number of reviews can be created.

By right-clicking on a specific milestone, a new review can be created. A new window pops up the information for creating a new review can be fed to this window. First, a specific person must be selected. Furthermore, a description and a date can be added. This date field can for example be used to add a specific date when the review shall be finished. It is very important to know that the review automatically inherits the deadline from the project milestone. Therefore, the date in the review can be used as optional field. It is also possible to enter the date after the review was performed.

If the review is completed, the path to the created review must be inserted into the predefined path field and the completed flag can be selected by a project admin. This information is very important for the safety case and only a project admin or global admin is allowed to select this flag. If all reviews are finished, the report generator can be used for extracting the preliminary process-based safety case.

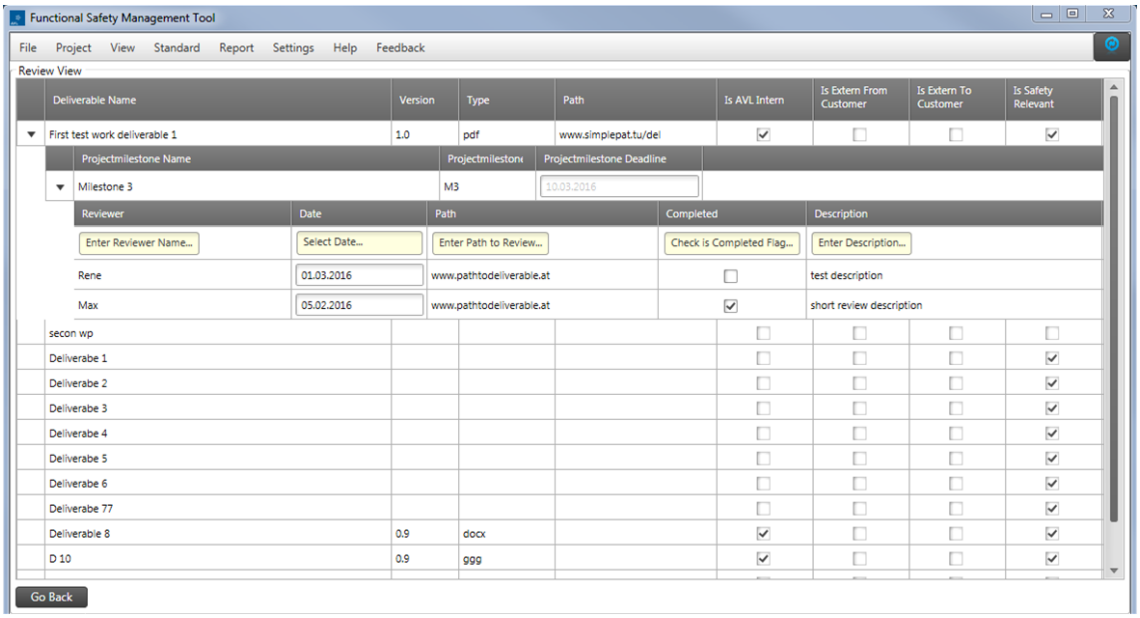

Figure 4.18.: The review module allows the creation of reviews for each deliverable within a connected milestone. A review consists of a reviewer, the path to the review, a specific date, a description and a is completed flag.

4.2.3. Report Generator

For getting the important information out of the tool, a report generator is developed and integrated. The report-generated framework is based on the "Open XML[2]" library which allows the manipulation of office-based documents like Microsoft Office Word files.

Existing Reports:

- Development Interface Agreement Report
- Preliminary Process-Based Safety Case Report

The developed report generator provides a framework for taking existing templates and adding dynamic information to these files. Template documents are stored on the server and can within the tool directly be brought to the client. Therefore, a window is created which lists all existing templates of the server. By selecting a specific template, a report can be generated. It is also possible to create separate templates by downloading an existing template from the server and updating this template with project-specific content. Thereafter, a global admin or project admin can upload the new template to the server. The round-trip can be conducted with the existing tool environment and needs no manual copying of files between the client and the server.

For exporting a report, two different possibilities exist. The first one is to open the Report generator by clicking "Report|Generate DIA" or "Report|Generate Safety Case" in the menu bar. A new window appears where a specific project template can be selected and thereafter the specific report is generated.

Another possibility is to open the project-specific view directly by double-clicking on a specific project in the main screen (Figure 4.2). In this project overview, two buttons for creating the development interface agreement and the process-based safety case exist. Both buttons are listed in the left bottom corner of the view.

The reports will automatically be opened after the successful creation and the report is saved in an arbitrary location. The created report is a Microsoft Office Word file and contains one the one side the predefined static content and on the other side the project-specific information that is needed for the specific report.

The report generator is build up with a powerful API that allows the simple creation of new reports.

[2] `https://msdn.microsoft.com/de-de/library/office/bb448854.aspx`

4.3. Unit Testing Framework

Testing is an important step during the development of a new tool. This section describes the created testing framework which is based on unit tests and contains unit tests for the different database manipulation and database fetching scenarios. To illustrate the created test framework, the tests of the activity entity are depicted in more detail.

The Visual Studio tool already provides a testing framework[3] named "Test Explorer" (Figure 4.19) for directly creating and running unit tests in the Visual Studio tool. Therefore, as a first step, an individual unit test project that is responsible for unit tests is created. This project needs all other created tool assemblies as inputs for testing them.

Furthermore, for testing database scenarios, a specific empty database for executing the test cases is created. This database contains the same structure as the database that is used within the tool environment. A particular database is needed because otherwise it is very difficult to test the expected results. If the database is always empty before running a test, the expected results can be defined more easily. Because the test framework contains an individual database, it also contains an individual database configuration for the object relational mapping that is responsible for the data exchange whit the database. This configuration contains further properties for evaluating and inspecting the created database statements during the data exchange. Therefore, it is really simple to see which database statements are created within the ORM. Such a property cannot be enabled in a productive environment because by activating this property, the tool needs much more time for displaying this database statement. Therefore, this feature is very time-consuming and slows down the system.

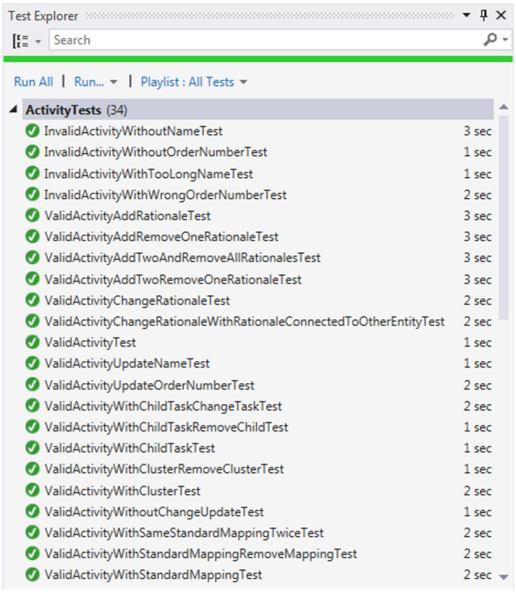

Figure 4.19.: Cutout of existing unit tests for the activity entity

[3] https://www.visualstudio.com/en-us/get-started/code/create-and-run-unit-tests-vs

For each existing database entity, the provided testing architecture contains an individual test class that is derived from a test factory base class. A cutout of the class diagram for the unit testing architecture is illustrated in Figure 4.20. The base class contains two main methods. One method is called each time before entering a single test case and the other one is called after the completion of each single test case. The base class also contains some helper classes that allow a simple creation of all existing entities. This is very helpful because in most cases, each test contains more than three entities.

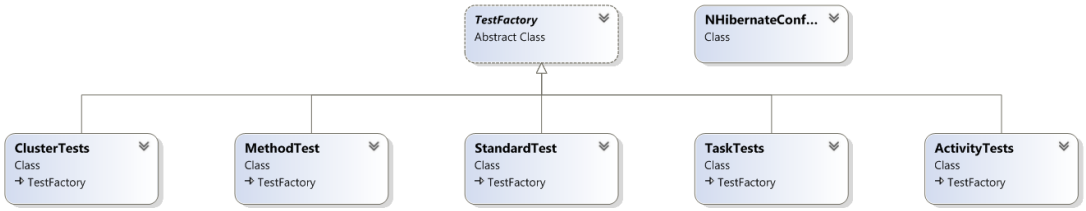

Figure 4.20.: Cutout of the existing testing framework class diagram

The first provided base class method creates the data factory which is responsible for creating the sessions that are needed for accessing the data manipulation methods provided by the ORM. Additionally, it creates a valid user with a correspondent role in order to access the defined WCF service methods.

The second method that is called after each single test case is completed, closes the before created session and deletes all data from the test database. It is very important to clear the entire database after each unit test in order to guarantee a valid testing framework that can be checked with assertions.

Listing 4.2 shows a simple existing unit test where a new activity is created for a specific standard. After the creation of the activity and standard they are mapped with the 'standardactivitymapping' entity. At the end, the necessary checks ("Asserts") are introduced in order to guarantee that the unit test passes. A second unit test is depicted in Listing 4.1. This test case illustrates a negative test case and checks whether an exception is thrown by creating an activity without defining a specific activity name.

```
[Test]
public void InvalidActivityWithoutNameTest()
{
  var activity = new Activity()
  {
    OrderNumber = 1
  };

  // we expect an exception
  Assert.Throws<FaultException<ProcessFault>>(() => _fsmService.
      SaveItem(activity));
}
```

Listing 4.1: Negative unit test for creating a new activity

```
[Test]
public void ValidActivityWithStandardMappingTest()
{
  var activity = GetTestActivity(); // method is located in base class
  var activityId = _fsmService.SaveItem(activity);

  var standard = GetTestStandard();
  var standardId = _fsmService.SaveItem(standard);

  var standardActivityMapping = new Standardactivitymapping()
  {
    Activity = activity,
    Standard = standard
  };

  var standardActivityMappingId = 0;
  if (activity.AddStandardActivityMapping(standardActivityMapping))
    standardActivityMappingId = _fsmService.SaveItem(standardActivityMapping);

  Assert.AreEqual(1, _fsmService.GetAllItems(Entities.Activity).Count);
  Assert.AreEqual(1, _fsmService.GetAllItems(Entities.Standard).Count);
  Assert.AreEqual(1, _fsmService.GetAllItems(Entities.Standardactivitymapping).Count);

  Assert.AreEqual(activity, _fsmService.GetItemById(Entities.Activity, activityId));
  Assert.AreEqual(standard, _fsmService.GetItemById(Entities.Standard, standardId));
  Assert.AreEqual(standardActivityMapping, _fsmService.GetItemById(Entities.
      Standardactivitymapping, standardActivityMappingId));

  Assert.AreEqual(standardActivityMapping, ((Activity)_fsmService.GetItemByName(Entities.Activity
      , "Activity")).Standardactivitymappings[0]);
  Assert.AreEqual(activity, ((Activity)_fsmService.GetItemByName(Entities.Activity, "Activity")).
      Activityclustermappings[0].Activity);
}
```

Listing 4.2: Creating a new activity with a standard mapping test

Unit testing is a very time-consuming task, but a well-defined testing framework is very helpful and essential for developing a tool environment with a huge number of modules. Only creating valid unit test cases is insufficient and therefore also negative unit test cases should be created. Once a stable testing framework is established, it is very helpful for integrating new features or for updating already existing modules. In summary, it can be said that a system without a well-defined and working testing framework cannot and shall not be used within a productive environment.

The created testing framework in this thesis already contains 34 unit tests only for the activity entity. Because 51 different entities exist within the tool environment, the number of existing test cases is very high. In the following, some example activity test cases are listed.

- Valid activity test
- Valid activity with cluster mapping test
- Valid activity with task mapping test
- Valid activity with rationale mapping test
- Invalid activity with too-long name test
- Invalid activity without name test
- Invalid activity without order number test
- Valid activity with connected clusters remove cluster test
- ...

The conclusion is that testing is a mature task during the development of every application or system. With a well-defined testing framework, it is possible to reduce the errors and to avoid problems. Furthermore, if a new developer joins the team and starts working on a well-tested project, it will help him or her to get an overview of the tool more easily. In addition, the period of vocational adjustment will significantly be shorter.

5. Conclusion and Future Work

5.1. Conclusion

An important challenge for the implementation of functional safety in an organizational structure is to ensure consistency and completeness of the safety lifecycle and the associated workproducts and tasks. Consistency is an essential factor for implementing an efficient and effective safety management within an organization and projects.

This thesis described the design and implementation of a novel collaborative safety management tool that was developed in cooperation with the AVL List GmbH. The provided tool allows the creation of standard-specific lifecycles and the creation of derived projects. Such projects inherit a safety lifecycle defined by the relevant standard. Furthermore, the projects automatically receive all necessary links to the standard-specific artifacts. Additionally, a report generator allows users to automatically generate reports such as the development interface agreement or the process-based safety case generation.

The provided tool environment consists of a client-server based architecture which communicates via defined WCF services. Furthermore, the server contains a MySQL database in the back-end and an object relational mapping library that is responsible for the entire data exchange with the database. The tool environment also allows multiple users to work concurrently on the same projects. With an integrated broadcasting mechanism, the users can automatically be notified if some changes within projects occur. Therefore, the tool environment always shows consistent and current data.

In addition, the tool environment automatically tracks data changes by an integrated auditing mechanism. This mechanism further allows the access of historical data and all changes which may occur during a project lifecycle. Furthermore, an integrated user management allows the definition of different user permissions and therefore the precise definition of the tool user's responsibilities.

The main benefits of the provided tool architecture are its responsibility for the traceability between the different artifacts and supporting consistency and completeness of the provided artifacts. Additionally, by providing a well-defined database architecture with adequate regulations, the tool avoids redundant information and provides a robust collaborative working framework. The integrated report generator allows the quick creation of large documents and, hence, the tool environment helps reduce time and costs by avoiding of manual report creations.

5.2. Future Work

The provided tool implementation in this thesis serves as a proof of concept for a novel collaborative safety management tool. The tool already allows the generation of a generic standard-specific and a project-specific part. Furthermore, it contains a framework for saving data changes throughout the entire project lifecycle. A first improvement of the existing tool implementation could be the enhancement of the already existing architecture with the ability to directly extract historical data from the tool and to further compare these data with the other versions. In addition, the existing report generator could be used to export these historical data and include them in specific provided reports.

Another enhancement could be the extension of the already existing wizards which allow the creation of new projects and standards based on existing information contained in the database. These wizards could be further improved by integrating a mechanism where it is possible to step by step go through an existing project or standard and select all artifacts which should be taken over into the new project or standard. Hence, in the future, it could be possible to derive a new project from an already existing one with only selecting the artifacts which are needed for the new project. The same mechanism could also be integrated into the standard part.

The developed safety management tool already contains a report generator which allows the creation of two different project-specific reports. These reports are the development interface agreement report and the process-based safety case report. The report generator contains a powerful API which allows the simple implementation of new report generators. Therefore, another improvement can be the direct generation of new reports from the provided tool. A new report could be for example the safety plan, the functional safety status report, the review report or a deliverable overview. It would also be possible to extract standard-specific artifacts for particular reports.

The term safety case was discussed in this thesis and an according report for a process-based safety case can be directly generated from the tool. The safety case generation could be further improved by integrating the possibility to directly create a graphical goal structure within the provided tool using GSN. Therefore, a graphical user interface could be developed which allows the generation of graphical safety cases by providing graphical elements and their relations.

As AVL List GmbH wants to start using the developed tool a tool qualification is needed. The goal of this tool qualification is to show that the tool does not produce any errors which can directly spread out to the system or product. Therefore, the confidentiality of the entire tool architecture shall be given by providing a tool qualification throughout the entire process. Currently, this is done by dumping the database state and checking the exported artifacts against conditioned historical artifacts to ensure consistency. In the future, a mechanism for showing tool qualification throughout the entire process of using the tool directly shall be introduced. By providing a use case in the tool, this tool qualification shall be completed by directly generating a report of the tool qualification from the tool. This report could show the required tool confidence level [12].

As can be seen, there are some helpful enhancements and improvements of the provided safety management tool. Therefore, this work could be the baseline of new topics for theses which can contribute to future work on an even better safety management tool.

A. AVL defined User Requirements

The following document shows the AVL defined user requirements for the novel Safety Management Tool that was developed during this thesis.

Maintenance of Standard Safety Lifecycles

It shall be possible to define and maintain safety lifecycles of standards in terms of

- name
- version
- clusters
- activities
- tasks
- methods
- workproducts (prescribed by the safety standard)
- normative requirements
- safety levels (e.g. ASIL, SIL, AgPL)

For each safety lifecycle of a standard it shall be possible to map

- activities to clusters
- tasks to activities
- methods to tasks
- tasks to (predecessor) tasks
- (input or output) workproducts to tasks
- normative requirements to tasks

A task shall inherit all the output workproducts of its predecessor tasks as input workproducts.

It shall be possible to define the sequence of

- clusters
- activities
- tasks

Maintenance of Project-Specific Safety Lifecycles

It shall be possible to define and maintain projects

If shall be possible to define for projects

- sub projects
- project milestones
- project-specific roles

Project milestones shall be denotable by

- name
- alias name
- date
- main milestone flag (see also DIA)

It shall be possible to define the project-specific safety lifecycle in terms of

- project-specific tasks
- deliverables

Deliverables shall be assignable to

- (sub) projects
- project milestones
- workproducts

It shall be possible to exclusively declare deliverables as

- AVL internal
- from customer
- to customer

It shall be possible to declare deliverables as

- not safety related

If a deliverable is not assigned to a workproduct then it shall be assignable to a project-specific task

Project-specific tasks shall be definable in terms of

- task
- (sub) project
- methods
- deliverables

It shall be possible to define organizations in terms of

- organization name
- description

It shall be possible to define roles in terms of

- role name
- description

It shall be possible to define persons in terms of

- name
- organization
- description

It shall be possible to create project-specific roles in terms of

- role
- person

It shall be possible to define reviews in terms of

- project-specific roles (reviewers)
- deliverable
- project milestone
- link to protocol
- completed flag

It shall be mandatory to map textual rationales on the deselection (tailoring) of

- clusters
- activities
- project-specific tasks

Potentially selectable textual rationales shall be automatically recommended based on previous rationales.

It shall be possible to define the development interface between customer and AVL in terms of

- main milestones
- project-specific tasks
- responsibility (R/A/S/I,-)

It shall be possible to define the deliverable responsibilities of AVL and customer in terms of

- project milestones
- responsibility (R/A/S/I,-)
- link to protocol
- completed flag

Baselining and Branching

It shall be possible to baseline standard safety lifecycles

It shall be possible to baseline projects

Systematic Reuse

Wizards shall be available which support the creation of a new projects

Wizards shall be available which support the creation of a new standards

Document Export

It shall be possible to export a DIA document for the sub projects listing

- project-specific tasks
- responsibility (R/A/S/I,-)
- deliverables
- main milestones
- review status (completed/not completed)

It shall be possible to export a Safety Case document for the sub projects listing

- project
- project-specific roles
- project-specific tasks
- methods
- deliverables
- deliverable status (completed/not completed)
- reviews
- review status (completed/not completed)

The export of documents shall be based on templates which contain a dedicated space for project-specific information (chapters, text, figures, tables etc.).

User Management

Users shall not be able to view or change any contents unless login credentials are provided.

It shall be possible to manage the users of the tool in terms of

- user Ids
- user name
- user rights (global admin, project admin, regular user)
- project assignments
- access rights on project level (view, modify)

The global admin shall be able to

- create/modify projects
- create/modify users
- assign user rights (project admin, regular user) to projects
- create/modify safety lifecycles
- define subprojects and milestones
- assign access privileges to regular users
- change all other data on project level

The project admin shall be able to

- define subprojects and milestones
- assign access privileges to regular users
- change all other data on project level

The regular user shall be able to

- change all other data on project level (depending on access privileges)

Multi User Management

It shall be possible for users with ‘modify’ access rights to lock/unlock projects.

If a project is locked and the project is modified then the user interfaces all active tool users shall be updated.

User Interface

There shall be the following separate administration UIs (user interfaces)

- workproducts UI
  - (for defining workproducts)
- methods UI
  - (for defining methods)
- requirements UI
  - (for defining requirements and assigning requirements to tasks)
- standard safety lifecycle UI
  - (for defining the clusters, activities, tasks and assigning workproducts)
- project role UI
  - (for defining organizations, persons, roles and project-specific roles)
- project UI

    - (for project, project milestones, sub projects)
- tailoring UI
    - (for defining and tailoring project-specific tasks)
- deliverable UI
    - (for defining deliverables and assigning deliverables to workproducts)
- DIA UI
    - (for defining responsibilities (R/A/S/I/-) for project-specific tasks)
- deliverable responsibilities UI
    - (for assigning project milestones to deliverables and defining responsibilities (R/A/S/I/-) for deliverables)
- task inputs/outputs UI
    - (for defining input tasks and assigning workproducts to tasks)
- project-specific task UI
    - (for project-specific input tasks and assigning deliverables)
- review UI
    - (for defining and assigning reviews)
- user management UI
    - (to manage users, user rights and access privileges)

There shall be the following separate read-only UIs (user interfaces)

- standard safety lifecycle read-only UI
- project-specific safety lifecycle read-only UI

The UI shall support information consistency by avoiding redundant inputs of identical information.

B. WPF – XAML Example

Figure B.1 shows a simple graphical user interface created with the WPF library and furthermore Figure B.2 depicts the appropriate XAML file.

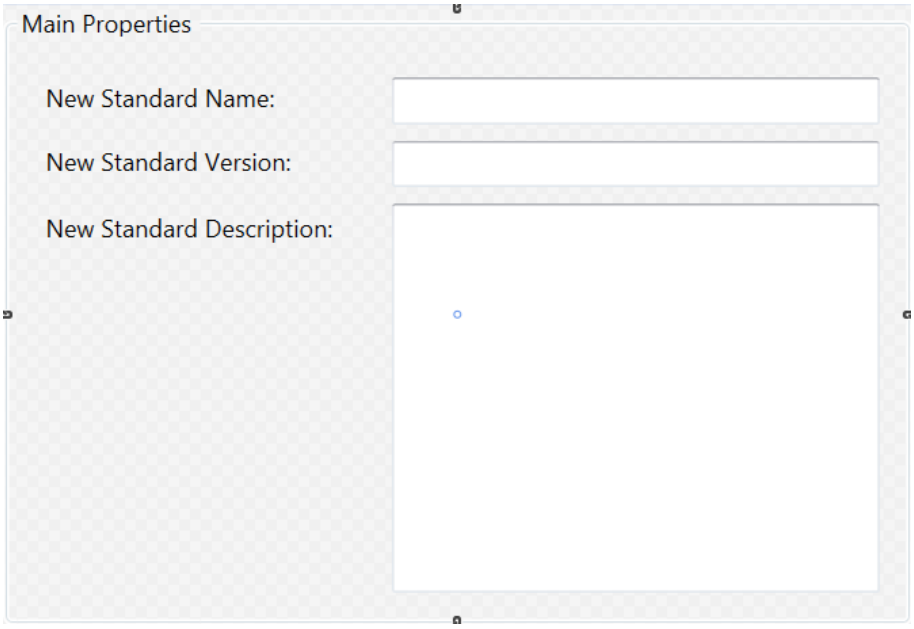

Figure B.1.: An example graphical user interface created with the WPF library

```
<UserControl x:Class="FSM.Views.Wizards.StandardWizardViews.NewStandardWelcomePageView"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             mc:Ignorable="d"
             d:DesignHeight="305" d:DesignWidth="447">
    <GroupBox Header="Main Properties">
        <Grid>
            <Label Content="New Standard Name:" HorizontalAlignment="Left" Margin="10,17,0,0" VerticalAlignment="Top" Width="170"/>
            <Label Content="New Standard Version:" HorizontalAlignment="Left" Margin="10,48,0,0" VerticalAlignment="Top" Width="170"/>
            <Label Content="New Standard Description:" HorizontalAlignment="Left" Margin="10,81,0,0" VerticalAlignment="Top" Width="170"/>
            <TextBox Height="23" Margin="185,20,10,0" TextWrapping="Wrap" Text="{Binding Name, Mode=TwoWay}" VerticalAlignment="Top"/>
            <TextBox Height="23" Margin="185,51,10,0" TextWrapping="Wrap" Text="{Binding Version, Mode=TwoWay}" VerticalAlignment="Top"/>
            <RichTextBox Margin="185,82,10,10">
                <FlowDocument>
                    <Paragraph>
                        <Run Text="{Binding Description}"/>
                    </Paragraph>
                </FlowDocument>
            </RichTextBox>
        </Grid>
    </GroupBox>
</UserControl>
```

Figure B.2.: Associated XAML file of the above illustrated graphical user interface

C. EER Diagram

The EER diagram is an advanced database modeling tool which allows the graphical creation and adaption of database related information. This extension is already integrated into the MySQL workbench tool. Within this tool, tables can be simple created by dragging and dropping the elements from the toolbox onto the specific database diagram. Furthermore, attributes can be added to these tables in order to specify the detailed table properties. If the properties are defined, the tables can be linked together with different kinds of relationships as defined in the following.

- **1:1 Relationship** - In a one-to-one relationship one entity is connected to exactly one other entity in the database (C.1).

- **1:n Relationship** - In a one-to-many or many-to-one relationship one entity is connected to more than one other entities (C.2). This relation is additional named collection mapping or child parent mapping because the first entity holds a list of second entities.

- **m:n Relationship** - In a many-to-many relationship two entity tables are connected within an own mapping table. In such a case each entity can hold a list of other entities (C.3).

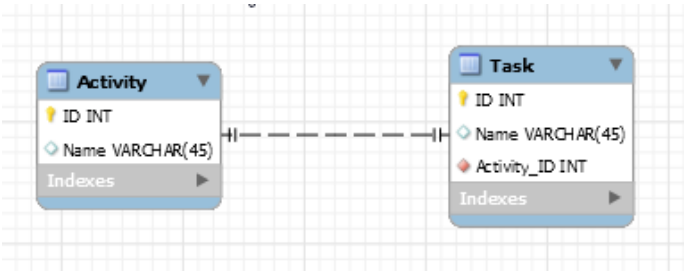

Figure C.1.: An example one-to-one mapping inside an EER Diagram

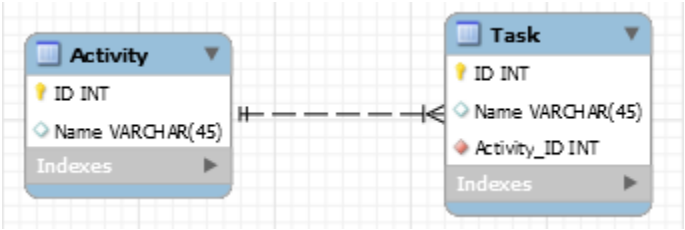

Figure C.2.: An example one-to-many mapping inside an EER Diagram

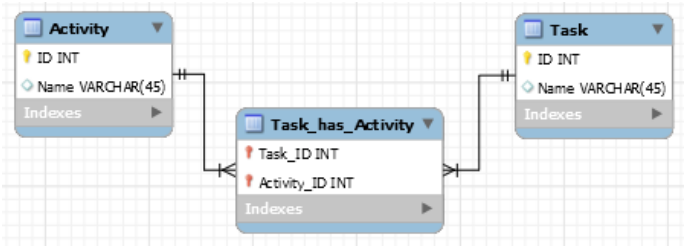

Figure C.3.: An example many-to-many mapping inside an EER Diagram

D. Example NHibernate Entity Configuration

Listing D.1: NHibernate task entity configuration

```
<?xml version="1.0" encoding="utf-8"?>
<hibernate-mapping namespace="NHibernateHelper" assembly="
    NHibernateHelper" xmlns="urn:nhibernate-mapping-2.2">
  <class name="Task" table="`task`" >
    <id name="Id" access="property" column="`ID`">
      <generator class="native" />
    </id>
    <property name="Name" type="String" column="`Name`" length="255
        " not-null="true"/>
    <property name="OrderNumber" type="Int32" column="`OrderNumber`
        " />

    <bag name="Taskactivitymappings" inverse="true" cascade="all-
        delete-orphan" lazy="true">
      <key column="`TaskID`" />
      <one-to-many class="Taskactivitymapping" />
    </bag>

  <bag name="Projecttasks" cascade="none" inverse="true" lazy="true
      ">
    <key column="`TaskID`" />
    <one-to-many class="Projecttask" />
  </bag>

    <bag name="Standardtaskmappings" inverse="true" cascade="all-
        delete-orphan" lazy="true">
      <key column="`TaskID`" />
      <one-to-many class="Standardtaskmapping" />
    </bag>
  </class>
</hibernate-mapping>
```

Listing D.2: NHibernate task class that is used for the associated database table

```
1  using System;
2  using System.Collections.Generic;
3  using System.Runtime.Serialization;
4
5  namespace NHibernateHelper
6  {
7      [Serializable]
8      [DataContract(IsReference = true)]
9      public partial class Task : Mappings.Contract.Item
10     {
11         public Task()
12         {
13             Taskactivitymappings = new List<Taskactivitymapping>();
14             Projecttasks = new List<Projecttask>();
15             Standardtaskmappings = new List<Standardtaskmapping>();
16         }
17
18         [DataMember]
19         public virtual string Name { get; set; }
20
21         [DataMember]
22         public virtual int? OrderNumber { get; set; }
23
24         [DataMember]
25         public virtual IList<Taskactivitymapping>
               Taskactivitymappings { get; set; }
26
27         [DataMember]
28         public virtual IList<Projecttask> Projecttasks { get; set;
               }
29
30
31         [DataMember]
32         public virtual IList<Standardtaskmapping>
               Standardtaskmappings { get; set; }
33
34       // orm helper methods
35         public virtual bool AddTaskactivitymapping(
               Taskactivitymapping taskactivitymapping)
36         {
37             if (!this.Taskactivitymappings.Contains(
                   taskactivitymapping))
38             {
39                 taskactivitymapping.Task = this;
40                 this.Taskactivitymappings.Add(taskactivitymapping);
41                 return true;
42             }
43             return false;
44         }
45
```

```
46          public virtual bool RemoveTaskactivitymapping(
                Taskactivitymapping taskactivitymapping)
47          {
48              if (this.Taskactivitymappings.Contains(
                    taskactivitymapping))
49              {
50                  this.Taskactivitymappings.Remove(
                        taskactivitymapping);
51                  return true;
52              }
53              return false;
54          }
55
56          public virtual bool AddStandardtaskmappings(
                Standardtaskmapping standardtaskmapping)
57          {
58              if (!this.Standardtaskmappings.Contains(
                    standardtaskmapping))
59              {
60                  standardtaskmapping.Task = this;
61                  this.Standardtaskmappings.Add(standardtaskmapping);
62                  return true;
63              }
64              return false;
65          }
66
67        // equality and hash code overridden methods
68          private int? _hashCode;
69
70          public override int GetHashCode()
71          {
72              if (_hashCode.HasValue)
73                  return _hashCode.Value;
74
75              var transientEntity = Id == 0;
76              if (transientEntity)
77              {
78                  _hashCode = base.GetHashCode();
79                  return _hashCode.Value;
80              }
81              return Id.GetHashCode();
82          }
83
84          public override bool Equals(object obj)
85          {
86              var other = obj as Task;
87              if (other == null)
88                  return false;
89
90              var thisIsTransient = Id == 0;
91              var otherIsTransient = other.Id == 0;
92
```

```
93              if (thisIsTransient && otherIsTransient)
94                  if (ReferenceEquals(this, other))
95                      return true;
96
97              if(Id == 0 || other.Id == 0)
98                  return Name == other.Name && OrderNumber == other.
                        OrderNumber;
99
100             return Id == other.Id && Name == other.Name &&
                    OrderNumber == other.OrderNumber;
101         }
102
103         public static bool operator ==(Task lhs, Task rhs)
104         {
105             return Equals(lhs, rhs);
106         }
107
108         public static bool operator !=(Task lhs, Task rhs)
109         {
110             return !Equals(lhs, rhs);
111         }
112     }
113 }
```

E. Infragistics Controls and Components

- XamBusyIndicator
- Drag and Drop Framework
- Excel Engine
- Math Library
- Persistence Framework
- Reporting
- Resource Washer
- Syntax Parsing Engine
- Theme Manager
- Undo Redo Framework
- Word Engine
- XamBarcode
- XamBarcodeReader
- XamBulletGraph
- XamCalculationManager
- XamCalender
- XamCarousel
- XamCarouselListBox
- XamCarouselPanel
- XamColorPicker
- XamComboEditor
- XamContextMenu
- XamDataCards
- XamDataChart
- XamDataGrid
- XamDataPresenter
- XamDataTree
- XamDiagram
- XamDialogWindow
- XamDockManager
- XamDoughnutChart
- XamFormulaEditor
- XamFunnelChart
- XamGantt
- XamGeographicMap
- XamGrid
- XamInputs
- XamLinearGauge
- XamMap
- XamMenu
- XamMonthCalender
- XamMultiColumnComboEditor
- XamNetworkNode
- XamOrgChart
- XamOutlookBar
- XamPieChart
- XamPivotGrid
- XamPropertyGrid

- XamRadialGauge
- XamRadialMenu
- XamRibbon
- XamRichTextEditor
- XamSchedule
- XamSegmentedDisplay
- XamSlider
- XamSparkline
- XamSpellChecker
- XamSpreadsheet
- XamSyntaxEditor
- XamTabControl
- XamTagCloud
- XamTileManager
- XamTimeline
- XamTreeGrid
- XamTreemap
- XamZoombar

Bibliography

[1] ISO - International Organization for Standardization. ISO 26262 Road vehicles Functional Safety Part 1 - Vocabulary, 2011.

[2] ISO - International Organization for Standardization. ISO 26262 Road vehicles Functional Safety Part 2 - Management of functional safety, 2011.

[3] TP Kelly. Arguing safety-a systematic approach to safety case management. 1998. *Department of computer science, University of York*.

[4] Habli Ibrahim and Kelly Tim. Process and Product Certification Arguments - Getting the Balance Right. ACM, 2006.

[5] Tim Kelly. A Systematic Approach to Safety Case Management. 2004.

[6] ISO - International Organization for Standardization. ISO 26262 Road vehicles Functional Safety Part 3 - Concept phase, 2011.

[7] ISO - International Organization for Standardization. ISO 26262 Road vehicles Functional Safety Part 9 - ASIL-oriented and safety-oriented analyses, 2011.

[8] Thomas Claudius Huber. *Windows Presentation Foundation 4.5*, volume 3.0. Galileo Press, 2013.

[9] ISO International Organization for Standardization. Iso 26262 road vehicles functional safety part 1-10, 2011.

[10] ISO International Organization for Standardization. Iec 61508 functional safety of electrical/ electronic / programmable electronic safety-related systems.

[11] Peter KAFKA. The Automotive Standard ISO 26262, the innovative driver for enhanced safety assessment & technology for motor cars. Elsevier Ltd, 2012.

[12] ISO - International Organization for Standardization. ISO 26262 Road vehicles Functional Safety Part 8 - Supporting processes, 2011.

[13] Philip Stirgwolt. Effective management of functional safety for iso 26262 standard. IEEE, 2013.

[14] Krammer Martin, Armengaud Eric, and Bourrouilh Quentin. Method Library Framework for Safety Standard Compliant Process Tailoring. 37th EUROMICRO Conference on Software Engineering and Advanced Applications, IEEE, 2011.

[15] Ulf Wilhelm, Susanne Ebel, and Alexander Weitzel. *Handbuch Fahrerassistenzsysteme: Grundlagen, Komponenten und Systeme für aktive Sicherheit und Komfort*, chapter Funktionale Sicherheit und ISO 26262, pages 85–103. Springer Fachmedien Wiesbaden, Wiesbaden, 2015.

[16] Löw Peter, Pabst Roland, and Petry Erwin. *Funktionale Sicherheit in der Praxis*, volume 1.0. dpunkt.verlag, 2010.

[17] ISO - International Organization for Standardization. ISO 26262 Road vehicles Functional Safety Part 10 - Guideline on ISO26262, 2011.

[18] Timothy Patrick Kelly. *Arguing Safety - a Systematic Approach to Managing Safety Cases*. PhD thesis, University of York, 1998.

[19] Gsn community standard version 1, 2011.

[20] Matsuno Yutaka and Yamamoto Shuichiro. An implementation of gsn community standard. San Francisco, CA, USA, 2013. ASSURE 2013, IEEE.

[21] ISO - International Organization for Standardization. ISO 26262 Road vehicles Functional Safety Part 4 - Product development at the system level, 2011.

[22] ISO - International Organization for Standardization. ISO 26262 Road vehicles Functional Safety Part 5 - Product development at the hardware level, 2011.

[23] ISO - International Organization for Standardization. ISO 26262 Road vehicles Functional Safety Part 6 - Product development at the system software level, 2011.

[24] ISO - International Organization for Standardization. ISO 26262 Road vehicles Functional Safety Part 7 - Production and operation, 2011.

[25] Armengaud Eric, Bourrouilh Quentin, Griessing Gerhard, Helmut Martin, and Reichenpfader Peter. Using the CESAR Safety Framework for Functional Safety Management in the Context of ISO 26262. EMBEDDED REAL TIME SOFTWARE AND SYSTEMS, 2012.

[26] Thomas Wahl Ajitha Rajan, editor. *CESAR - Cost-efficient Methods and Processes for Safety-relevant Embedded Systems*, volume 1. Springer Vienna, 2013.

[27] I.J. Popplewell N. Crompton. Integrated Functional Safety Management for Software to achieve Functional Safety throughout the Lifecyce Phases. System Safety. IEEE, 2010.

[28] IEC61511 FUNCTIONAL SAFETY - SAFETY INSTRUMENTED SYSTEMS FOR THE PROCESS INDUSTRY SECTOR, 2004.

[29] Software and Systems Process Engineering Meta-Model, 2008.

[30] Barbara Gallina. A Model-driven Safety Certification Method for Process Compliance. International Symposium on Software Reliability Engineering Workshops, IEEE, 2014.

[31] Palin Rob, Ward David, Habli Ibrahim, and Rivett Roger. Iso 26262 safety cases: Compliance and assurance. IEEE, 2011.

[32] Origin Consulting York Limited. GSN Community Standard Version 1. online, 2011.

[33] Rene Obendrauf. Software architecture of a collaborative functional safety management tool environment, February 2016.

[34] Pröll Stefan, Zangerle Eva, and Gassler Wolfgang. *MYSQL - Das umfassende Handbuch*, volume 3. Rheinwerk Verlag, 2015.

[35] Suhas Chatekar. *Learning NHibernate 4*. https://bitbucket.org/RogerKratz/nhibernate.envers, 2015.