Safety-by-Design in Architecture of Automotive Software Systems

EngD Thesis Report

Safety-by-Design in Architecture of Automotive Software Systems

October/2022

TomTom Automotive - TU Eindhoven (Department of Mathematics & Computer Science)

EngD Software Technology

TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

Safety-by-Design

in Architecture of Automotive Software Systems

D. C. Chirascu

October 2022

Eindhoven University of Technology
Stan Ackermans Institute – Software Technology

EngD Report: 2022/064

Confidentiality Status:
Not confidential

Partners

TomTom Eindhoven University of Technology

Steering Group TU/e: Mark van den Brand, Jerry den Hartog
TomTom: Remco Pasman, Guus Holshuijsen, Marcin Klecha

Date October 2022

Composition of the Thesis Evaluation Committee:

Chair: Mark van den Brand

Members: Jerry den Hartog

Gijs Dubbelman

Arash Saberi

Remco Pasman

Guus Holshuijsen

The design that is described in this report has been carried out in accordance with the rules of the TU/e Code of Scientific Conduct.

Date October, 2022

Contact address
Eindhoven University of Technology
Department of Mathematics and Computer Science
Software Technology
MF 5.080 A
P.O. Box 513
NL-5600 MB
Eindhoven, The Netherlands
+31 402744334

Published by Eindhoven University of Technology

EngD Report 2022/064

Abstract

The increasing vehicle reliance on digital applications raises concerns as to the safe function of automotive software systems. Safety design practices, such as those specified in the ISO26262 standard are yet to be integrated in the software creation and maintenance cycle of automotive software providers. The current work explains the safety design process of ISO26262 and applies it to a proof-of-concept system. Furthermore, it documents the impact of the safety design process on the software creation process, in terms of additional work required and value obtained.

Keywords Automotive Functional Safety, ISO26262, Navigation, Lane-Level Guidance, Lane-Level Navigation, Blind-Spot Highlighting

Preferred reference Safety-by-Design in Architecture of Automotive Systems. Eindhoven University of Technology, EngD Report 2022/064, October 2022.

Partnership This project was supported by Eindhoven University of Technology and **TomTom N.V.**

Foreword

The request TomTom made to the PDEng, now EngD, trainee was very broad "Safety and Security by Design in architecture of automotive software systems". Dan was not daunted by the broad scope and in our first talks expressed an interest in the wonderful world of automotive.

That interest did not wane during his traineeship. We decided that he would focus on safety and how this could fit in with the way of working of TomTom. For this he needed an example project, small enough to fit a one-man team and big enough to be relevant. We ended up with the feature "blind spot detection in lane level navigation".

Dan quickly found his way in the TomTom organization and contacted safety experts, development teams and other people to support him in his efforts. He managed to show how safety could be included in the TomTom way of working and help pave the way for TomTom to better understand the impact of (not doing) safety. I was impressed by the ease with which he contacted relevant people and how he managed to get their support, especially since most of us were working remotely.

Dan's work brought the TomTom organization one step closer to informed decision-making regarding safety. A job well done.

Ir. Guus Holshuijsen
Solution Architect - TomTom

October 2022

Preface

This document is the final report of the "Safety by Design in Architecture of Automotive Systems" project, executed by Dan-Cristian Chirascu, as the graduation project of the Engineering Doctorate (EngD) program in Software Technology. EngD is a two-year doctorate-level program provided by the Eindhoven University of Technology under the banner of 4TU.School for Technological Design, Stan Ackermans Institute.

The project was carried out in collaboration with TomTom to investigate the methods of integrating safety design practices in the creation of automotive software products. The project goal was achieved by exploring and understanding safety design practices followed by applying them on a proof-of-concept system.

The current work is organized as follows:

- Chapter 1 summarizes the automotive industry context, defines the problem statement and lists the research questions.
- Chapter 2 describes the prior work used to answer the research questions, understand automotive safety design practices and apply them to a proof-of-concept system.
- Chapter 3 introduces the reader to the ISO26262 safety design theory concerning software systems.
- Chapter 4 documents the application of the safety design theory on a proof-of-concept system. The static code analysis of existing product code of TomTom is also documented here.
- Chapter 5 provides an analysis of the costs and benefits of safety design on the software creation process.
- Chapter 6 summarizes the results and conclusions and recommends future work.

Dan-Cristian Chirascu

October 2022

Acknowledgements

I would like to thank my company supervisors Remco Pasman, Guus Holshuijsen and Marcin Klecha for their patient feedback and availability. Remco has been essential towards steering the project to its successful end, helping ask the important questions that lead to understanding the company-wide implications of integrating safety-by-design. He has helped me towards understanding the human-side of engineering, and in improving my communication and presentation skills. The feedback received will benefit me greatly in my career as an engineer and architect. Guus' level of involvement in the project is rarely seen among student or trainee supervisors, and was a constant source of insight and support. He not only offered much needed professional guidance on the technical side, but did so patiently and showing great generosity with his attention to detail and general will to help. Marcin helped steer the project technically by providing architectural and risk management insight, which backed-up the analysis and realization of the proof-of-concept system.

I would like to show my gratitude to my TU/e supervisors Mark van den Brand and Jerry den Hartog, who participated actively and patiently through my lengthy and sometimes complex presentations. Mark provided valuable guidance for both myself and the company, relying on extensive automotive and safety engineering knowledge. He helped us understand that functional safety is critical towards entering and thriving in the automotive space, using examples from the automotive industry which brought much needed perspective. Jerry gave important feedback on the safety analysis and helped prioritize the essential quantitative and qualitative results to be analyzed, which helped bring the project to a useful conclusion.

I would like to give special thanks to Arash Saberi and Roland Rosier, who, although not part of the project officially, have helped me quickly learn the essentials of functional safety and its application in design and implementation phases. I have really learned a lot from them regarding safety practices throughout the project months.

Finally, I would like to thank Yanja Dajsuren, the EngD Software Technology program director, whose intensive coordination work helps make this program possible. She has been with my trainee generation from beggining to end, providing much needed advice and encouragement.

Dan-Cristian Chirascu

October 2022

List of Tables

4.1 Functional Requirements 19
4.2 Hazards resulting from malfunctions of the Blind-Spot Highlighting System 20
4.3 Safety Goals and ASIL levels 21
4.4 Proof-of-concept FSRs 21

A.1 HAZOP on quantitative failures 44
A.2 HAZOP on real-time performance failures 45
A.3 HARA 46
A.4 HARA (continued) 47
A.5 HARA (continued) 48
A.6 HARA (continued) 49

B.1 FMEA Severity Scoring 52
B.2 FMEA Occurrence Scoring 53
B.3 FMEA Detection Scoring 54
B.4 System-level FMEA 55
B.5 Component-level FMEA 56
B.6 Component-level FMEA (continued) 57
B.7 Component-level FMEA (continued) 58

C.1 ISO26262:6 Software Implementation Guidelines 59
C.2 ISO26262:6 Software Testing Guidelines 60

List of Figures

1.1 Centralization Trend of Automotive Architecture 1 [14] . . . 2
1.2 Centralization Trend of Automotive Architecture 2 [23] . . . 2
1.3 Concept of future vehicle navigation on TomTom IndiGO platform [1] . . . 3
1.4 Informing driver of nearby vehicle [1] . . . 4
1.5 Example ASIL classifications [5] . . . 5

2.1 Structure of the ISO26262 standard [17] . . . 8

3.1 Concept Phase Sub-Phases . . . 10
3.2 Functional Safety Static Model . . . 11
3.3 Functional Safety Flow Model . . . 12

4.1 Navigation System Use Cases . . . 17
4.2 TomTom Blind-Spot Highlighting Simulation . . . 18
4.3 TomTom Traffic Warning Simulation . . . 18
4.4 Blind-Spot Highlighting: Black-Box View . . . 19
4.5 Initial System Design . . . 22
4.6 Post-FMEA System Design . . . 23
4.7 Post-FMEA System Design . . . 23
4.8 Proof-of-concept simulation - no objects in blind-spots . . . 25
4.9 Proof-of-concept simulation - object in left blind-spot . . . 26
4.10 Proof-of-concept simulation - object in right blind-spot . . . 26
4.11 Proof-of-concept simulation - object in left and right blind-spots . . . 27
4.12 Proof-of-concept simulation - example errors . . . 27
4.13 Proof-of-concept simulation - example errors . . . 28
4.14 Proof-of-concept simulation - example errors . . . 29

5.1 Benefits of safety concept . . . 32
5.2 Benefits of system safety . . . 33

5.3 Cost of TomTom software redesign 34
5.4 Benefits of software safety 35
5.5 Benefits of safety-by-design 36
5.6 Cost of safety - Unoptimized Process 38
5.7 Cost of safety - Optimized Process 38

6.1 Sub-domains of Safety 40

Contents

Foreword i

Preface iii

Acknowledgements v

List of tables vi

List of figures vii

1 Introduction 1

1.1 The Software-Defined Vehicle . . . 1

1.2 TomTom Lane-Level Guidance . . . 1

1.3 Designing with Safety in Mind . . . 3

1.4 Safety Levels . . . 4

1.5 Project Goals . . . 4

2 Functional Safety Standards 7

3 Functional Safety Theory 9

3.1 Concept Phase . . . 9

3.1.1 Introduction . . . 9

3.1.2 Concept Phase Flow . . . 10

3.2 System-level Phase . . . 11

3.2.1 Introduction . . . 11

3.2.2 System-level Phase Flow . . . 12

3.2.3 FMEA Description . . . 13

3.3 Software-level Safety Phase . . . 14

3.3.1 Software Implementation Safety . . . 15

4 Functional Safety Application **17**
4.1 Blind-Spot Highlighting Use Case . . . 17
4.2 Deriving the Functional Safety Concept . . . 18
4.2.1 Item Definition . . . 18
4.2.2 Hazards Formulation . . . 20
4.2.3 HARA . . . 20
4.2.4 Functional Safety Requirements . . . 20
4.3 Deriving Safety Mechanisms and Tests . . . 22
4.3.1 System Design . . . 22
4.3.2 FMEA . . . 22
4.4 Software-level Safety Application . . . 23
4.4.1 Proof-of-Concept Implementation . . . 23
4.4.2 Static Code Analysis . . . 25

5 Benefits and Estimated Cost of Functional Safety **31**
5.1 Benefits of Functional Safety . . . 31
5.1.1 Functional Safety Concept Phase Benefits . . . 31
5.1.2 System-level Safety Benefits . . . 32
5.1.3 Software-level Safety Benefits . . . 33
5.1.4 Benefits of Safety-by-Design . . . 35
5.2 Estimated Cost of Functional Safety . . . 37

6 Conclusion **39**
6.1 Results Summary . . . 39
6.2 Future Work . . . 39

A Hazard Analysis and Risk Assesment **43**
A.1 Hazard and Operability Study . . . 43
A.2 Hazard Analysis and Risk Assesment . . . 43

B Failure Modes and Effects Analysis **51**
B.1 FMEA scoring . . . 51
B.2 System and Component-Level FMEA . . . 51

C Software Safety Phase - Additional Material **59**
C.1 ISO26262 Part 6 Guidelines . . . 59
C.2 Coverity Results . . . 59

D CAN Log Files 61

1 Introduction

1.1 The Software-Defined Vehicle

Demand for services such as in-vehicle infotainment, autonomous driving, and electrification is increasing the software reliance of vehicles [15]. As a consequence, in-vehicle applications are becoming more complex. To support this trend, computationally stronger Domain Control Units (DCUs) are replacing the less powerful Electronic Control Units (ECUs) [14]. Automotive electronics developers expect that, by 2030, vehicle electronic architectures will be more centralized, with DCUs taking over the applications currently supported by multiple ECUs [23], [14]. Figure 1.1 shows the projected evolution of automotive electronic architecture. From bottom to top these are:

- The 3rd generation relies heavily on application specific ECUs. Each function, such as parking assistance, has a dedicated ECU.
- The 4th introduces DCUs supporting all functions of a given domain. A DCU can for instance cover Advanced Driver Assistance Systems (ADAS) functions, such as parking assistance and blind-spot detection. The DCU processing architecture is more powerful and less application specific, which allows for higher software flexibility and complexity [4]. This automotive architecture is also exemplified in Figure 1.2 left side.
- The 5th generation is the most centralized, as ECU functionality is completely taken over by DCUs communicating directly with sensors and actuators. Another variant of highly-centralized architecture is shown in Figure 1.2, where DCUs are merged in one single High-Performance Computing (HPC) platform capable of handling applications from multiple domains.

The increase in computation power allows executing desktop programs or smartphone applications on vehicle hardware, giving birth to the software-defined vehicle. These new processing capabilities enable software companies to launch their products in the automotive space [19]. A category of software that is being integrated in vehicles is that of navigation applications. Access to the in-vehicle network allows navigation systems to communicate with sensors and provide high-accuracy guidance. They can assist the driver with lane-level route information, reporting on the local traffic situation. The automotive and software industry expects future vehicles to provide real-time lane-level guidance including information on vehicles in close proximity.

1.2 TomTom Lane-Level Guidance

TomTom is a geolocation and driving assistance software company that seeks to add lane-level guidance to its range of features. One product that provides high accuracy navigation is the IndiGo digital

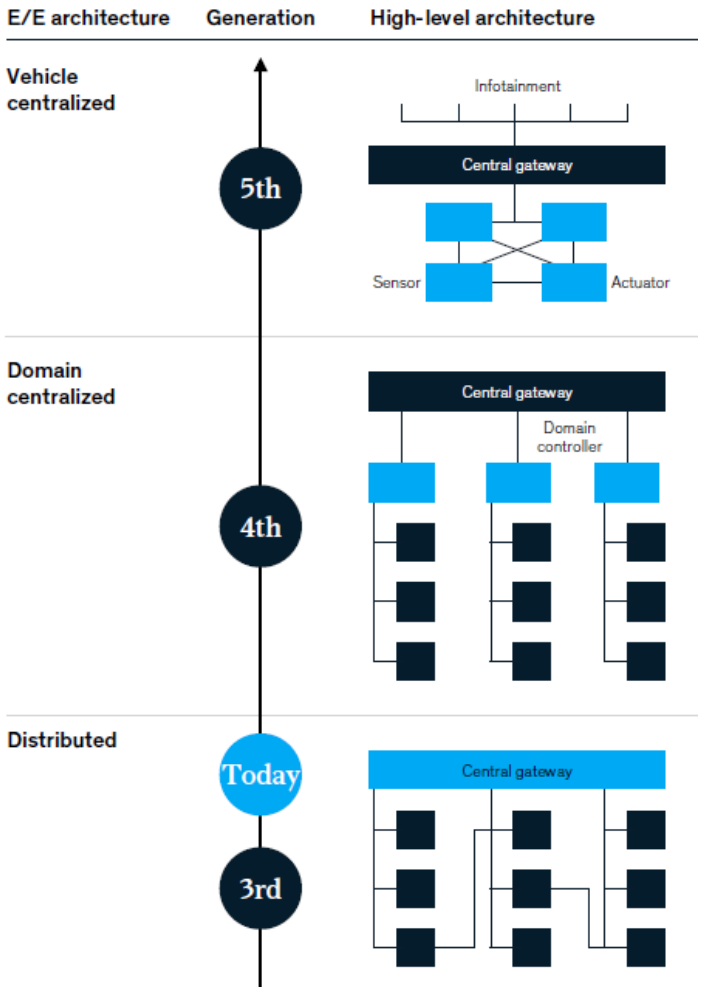

Figure 1.1: Centralization Trend of Automotive Architecture 1 [14]

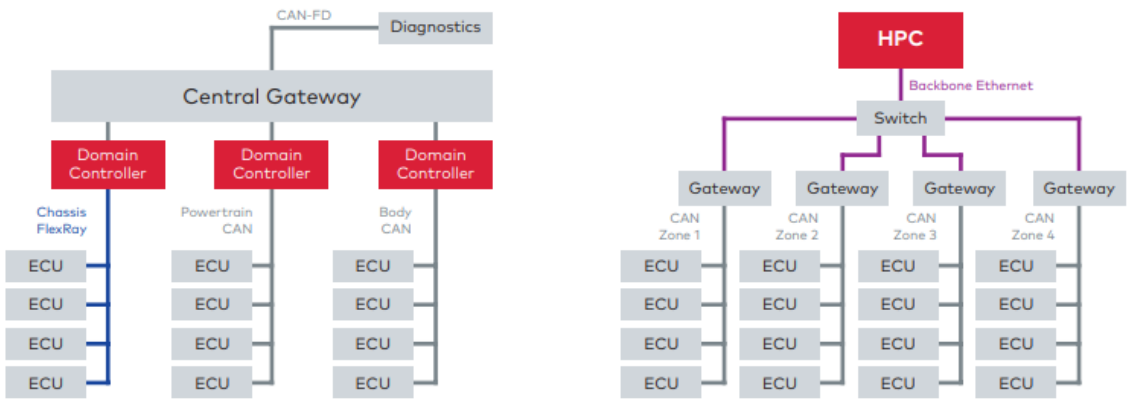

Figure 1.2: Centralization Trend of Automotive Architecture 2 [23]

Figure 1.3: Concept of future vehicle navigation on TomTom IndiGO platform [1]

cockpit [21]. This is a software platform that allows automotive and commercial software companies to deploy and maintain their services on an out-of-the-box In-Vehicle Infotainment (IVI) system. The platform provides a large collection of reusable commercial grade applications and services specifically designed for in-vehicle use. This allows for developing and maintaining software on a smartphone-like platform, but with access to Advanced Driver Assistance Systems (ADAS), sensors, and other vehicle data. The holistic interface provides the driver access to all the features in a safe and non-distracting way.

Figure 1.3 shows a concept of the IndiGO platform, where the cluster display behind the wheel provides lane-level guidance, while the center-stack display on the right shows road-level navigation similar to that of current navigation applications.

1.3 Designing with Safety in Mind

Lane-level navigation applications, while providing the driver with useful real-time traffic information and accurate route guidance, raise safety concerns, as malfunctions can cause the driver to make hazardous maneuvers. Sensor faults and obstructions, as well as data loss or corruption due to software errors, can lead to misguiding the driver and thus to accidents. Figure 1.4 shows an IndiGo platform concept where a nearby vehicle is displayed on the cluster screen. The interface uses a more alarming color, to signal the user that they should be careful when switching lanes, to avoid a collision. Should such a system fail, in the case of a careless driver not checking adjacent lanes before turning, the chance of potentially fatal accidents increases.

To function safely, high-accuracy navigation systems must prevent or mitigate malfunctions and safely handle environment situations that limit functionality. For these reasons, TomTom lane-level guidance features must be designed and implemented with safety in mind. Furthermore, safety should be part of development from its beginning, and not done as an afterthought. Treating safety as an afterthought causes additional delays and cost, not to mention endangering the driver and the environment around the vehicle.

Figure 1.4: Informing driver of nearby vehicle [1]

1.4 Safety Levels

Before defining the project goal and diving into the subject matter, the current work defines the concept of Automotive Safety Integrity Level (ASIL), which is used extensively in the next chapters. ASIL is a safety-criticality classification scheme used in the automotive industry, that helps understand the risk and effect of a vehicle function failing. The ASIL level reflects the risk, with A being the lowest and D the highest. While no safety design is required for QM functions, safety-by-design principles help increase software product quality.

The exact method for determining the ASIL of a function and its supporting system is discussed in Chapter 3. Figure 1.5 shows the ASIL levels of various vehicle functions. The ASIL C-D range is typically assigned to functions that affect vehicle maneuvers and passenger safety, while A-B covers functions that pertain to lighting or infotainment systems.

1.5 Project Goals

To understand the methods and impact of safety design on TomTom products, the current project develops a proof-of-concept system following state-of-the art automotive safety standards. In realizing this goal, the project achieves the following sequence of objectives:

1. Introduction to the safety design process according to state-of-the-art standards, adapted for use in TomTom.
2. Development of a proof-of-concept system, following the safety process introduced in Objective 1.
3. Documentation of the proof-of-concept improvements resulting from the addition of safety in the software development process.

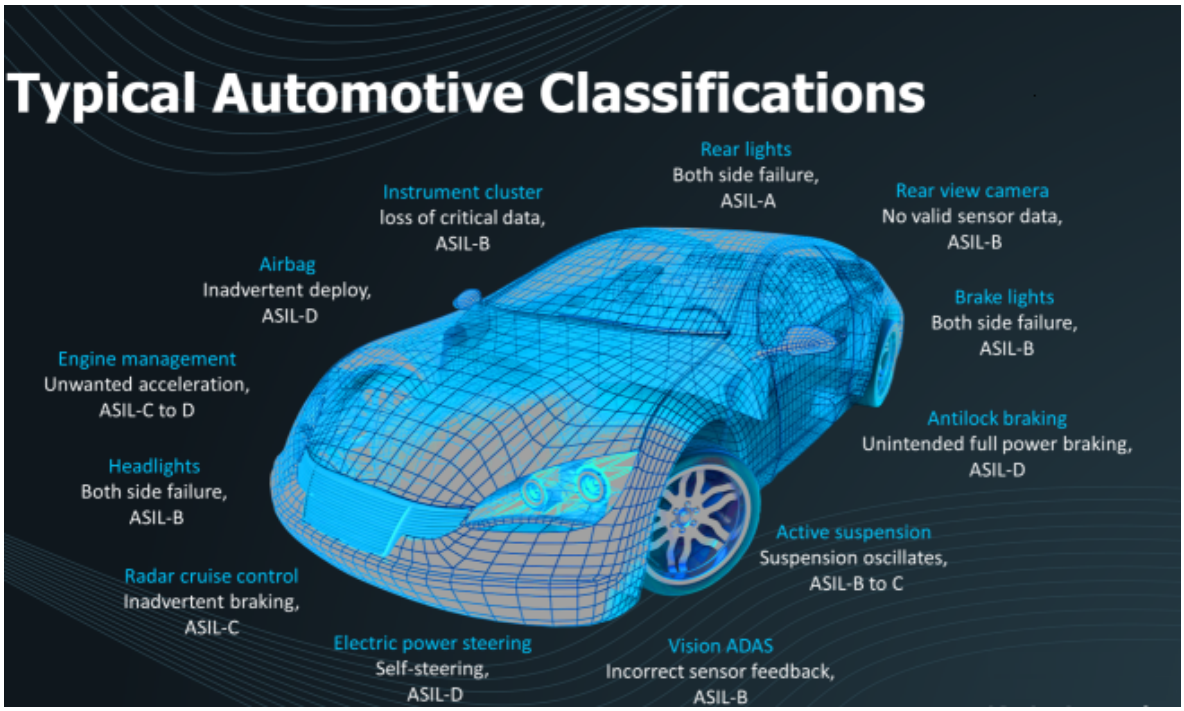

Figure 1.5: Example ASIL classifications [5]

4. Discussion of the value and cost of safety-by-design, in terms of technical and business-level benefits, as well as additional time and resources needed.

The insights gained from pursuing the above goal are listed below:

- Software functional-safety is a straightforward and relatively easy to understand process.
- Safety-by-design for software companies is likely to provide benefits exceeding the costs, in terms of technical and business value added.

2 Functional Safety Standards

The current study relies heavily on the ISO 26262 [13] standard, the state of the art functional safety standard for automotive systems. It is used extensively by the automotive industry and its suppliers [18]. It is based on the IEC 61508 [9] functional safety standard for Electrical/Electronic (E/E) systems. The ISO 26262 standard consists of twelve parts listed below and shown in Figure 2.1:

- Part 1 defines the vocabulary used in the rest of the standard.
- Part 2 addresses the management of functional safety in terms of staff required, planning, and development procedures.
- Part 3 describes the functional safety concept design phase, where the requirements for preventing dangerous failure effects are derived.
- Part 4 describes the system-level safety design phase, where the safety mechanisms necessary to satisfy the requirements of the previous phase are derived.
- Parts 5 and 6 describe the hardware and software-level safety design phases respectively. The focus of these is in specifying implementation and verification techniques that ensure correct functioning of hardware and software components.
- Part 7 covers the release of the system for production.
- Part 8 specifies supporting processes such as configuration, change and documentation management, as well as interfacing with and integration of non- ISO26262-developed applications.
- Part 9 describes guidelines for ASIL decomposition and safety analyses needed to derive the safety-mechanisms introduced in Part 4.
- Part 10 contains additional explanations and diagrams that help better understand the concepts and guidelines of the other parts.
- Part 11 gives very detailed help on the use of semiconductors and microcontrollers in safety-related systems.
- Finally, Part 12 has been added as an adaptation of the standard for motorcycles.

This work focuses mainly on Parts 3, 4 and 6, as they are most relevant for understanding safety design of software systems. Chapter 3 describes the processes specified in these parts in a clear and accessible way, to facilitate the reader's understanding of the standard. Saberi et al. [18] specifies the safety design artifacts and flow of Part 3 using UML, making the standard more understandable and

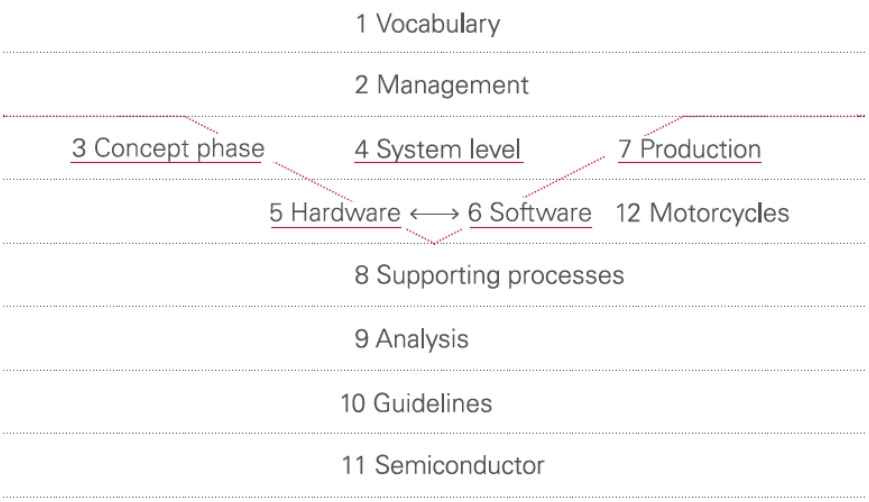

Figure 2.1: Structure of the ISO26262 standard [17]

interoperable with generic system design. The work of [18] is used in describing Part 3. In addition, the current study applies the theory described in the standard on the design and implementation process of a proof-of-concept system. This is done to concretely illustrate the application of the safety design process and the added value it provides, as opposed to ignoring safety in design.

In addition to the ISO 26262 standard, the current work makes use of the set of AUTOSAR guidelines for software implemented in the C++14 language, specified in [2]. AUTOSAR is a global partnership of over 300 automotive companies aiming to establish a standardized software framework for intelligent mobility [3]. The standards provided by the organization help increase the safety of software systems. The AUTOSAR document [2] provides mappings from its C++14 guidelines to the ISO26262 Part 6 rules for safe software implementation. This allows safety engineers to know which set of AUTOSAR C++14 guidelines are relevant for complying with the ISO 26262 standard.

Section 4.4.2 documents the outcome of checking the code used to implement the proof-of-concept against the AUTOSAR guidelines. The chapter reports how many issues occur and which are the most prevalent, along with examples of these.

3 Functional Safety Theory

This chapter introduces the reader to Parts 3, 4 and 6 of the ISO26262 standard. These describe the concept, system-level and software-level phases respectively.

3.1 Concept Phase

3.1.1 Introduction

The concept phase helps the safety engineer know the requirements necessary for the system to be considered safe. Further design decisions must take these requirements into consideration, making the concept phase a necessary first step in the safety design process. To help clarify the exact outcome of the concept phase, the artifacts obtained at the end of the phase are defined below:

1. The requirements necessary for the system to be considered safe, expressed as:
    - *Safety goals*, which specify the top-level safe behavior of the system.
    - *Functional Safety Requirements (FSRs)*, which are derived from the safety goals and specify the system functions necessary for the goals to be realized.
2. *The Automotive Safety Integrity Level (ASIL)* of the system. This is one of four levels (A, B, C and D) used to specify the necessary safety measures for preventing or mitigating a system's failures. Level A requires the least stringent safety measures, while D requires the most stringent.

To help understand the notions introduced above, a theoretical vehicle detection system is considered, with the following safety goal: *The system shall correctly represent a nearby vehicle within n milliseconds of receiving a "vehicle detected" signal*. Example FSRs specifying how the system achieves the safety goal behavior are:

1. The system shall verify that no vehicle detection data is lost until the corresponding vehicle representation is displayed.
2. In case data is found to be missing, the system shall warn the vehicle detection failure.
3. The system shall verify that the vehicle data is processed and rendered within *n* milliseconds.

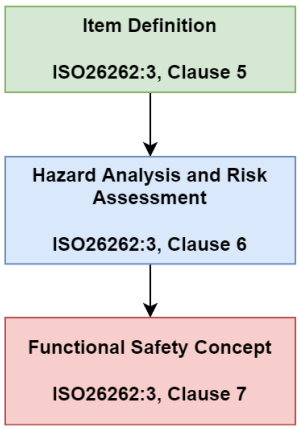

Figure 3.1: Concept Phase Sub-Phases

3.1.2 Concept Phase Flow

In order to obtain the safety goals and FSRs, the safety engineer needs to first know the system-level functions. No detailed system design is required. The concept phase consists of three sub-phases, which are described in Part 3 of the ISO26262 standard. The sub-phases are represented in Figure 3.1 and described below:

1. The first step is known as the *item definition*. The standard uses the term *item* to designate a system or system-of-systems that fulfills a function or part of a function at the vehicle level. The knowledge required at this step is the following:
    - Definition of the system-level functions. This can be obtained via requirements specification or simply drawing the "black-box" view of the system and brainstorming on the inputs, functions and outputs.
    - Definition of the system operating modes. For this, knowledge of the functional states of the system must be known and defined.
    - Definition of the system operational situations. These are the scenarios that can occur during a vehicle's life. They pertain to external factors such as weather, lighting and traffic conditions. These can be of course reduced to conditions considered relevant as the full set of operational situations is virtually infinite.
2. The second step is known as the *Hazard Analysis and Risk Assesment (HARA)*. It helps determine the ASIL level of the system-level functions and the safety goals needed to achieve functional safety. The steps of a HARA are as follows:
    - The HARA starts with listing of the possible malfunctioning behaviors of the system functions and the hazards these can cause.
    - The next step is obtaining the possible hazardous events, by combining the hazards with the possible operational situations in which they can occur and the operating modes of the system. Each combination forms a hazardous event. The hazardous events are scored in terms of severity, exposure, and controllability. The total score obtained for these three metrics determines the ASIL level of the function for which the failure has been analyzed.

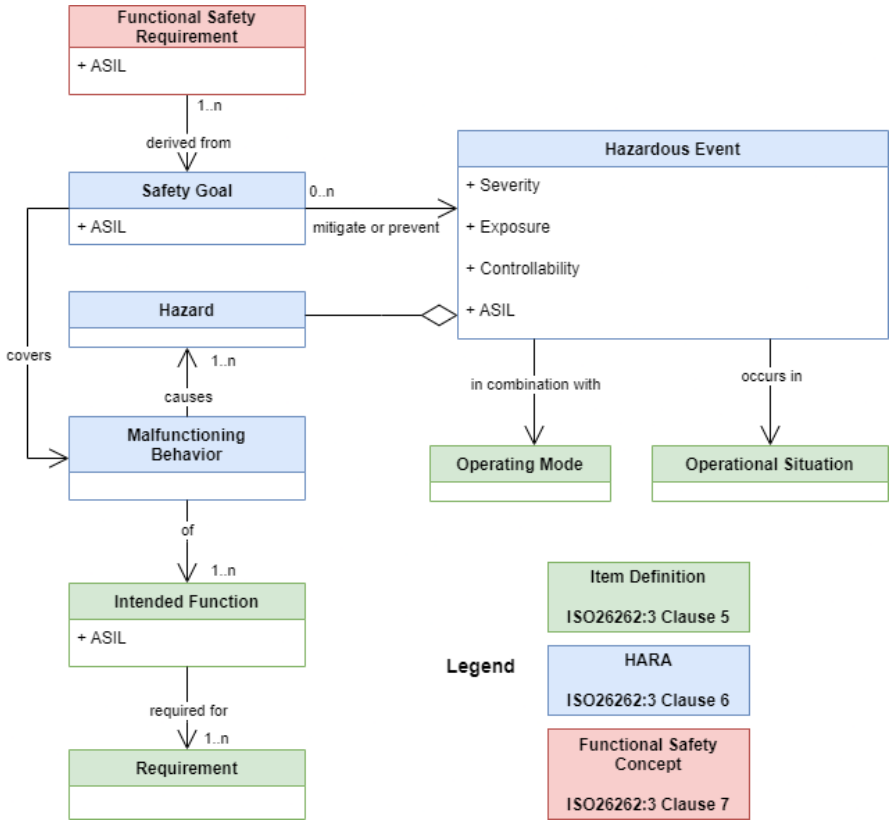

Figure 3.2: Functional Safety Static Model

- The last step of the HARA is formulating the safety goals necessary to prevent the hazardous effects of malfunctions. These inherit the ASIL level of their associated functions.

3. Once the safety goals are known, the last step is formulating the functional safety requirements. These form what the standard defines as the *functional safety concept*. The functional safety requirements can then be merged with the functional requirements, in order to be used as input in the system-level safety design phase.

Figure 3.2 illustrates in detail the artifacts obtained while performing Steps 1 through 3, and maps them to their respective steps using the same colors as in Figure 3.1. Figure 3.3 models the flow of the steps using the same color mapping. Following this process, the functional safety concept for the proof-of-concept system is obtained in Section 4.2.

3.2 System-level Phase

3.2.1 Introduction

The system-level phase helps the safety engineer in making design decisions that satisfy the concept phase FSRs. To help understand the exact outcome of this phase, the concrete artifacts obtained at its end are defined below:

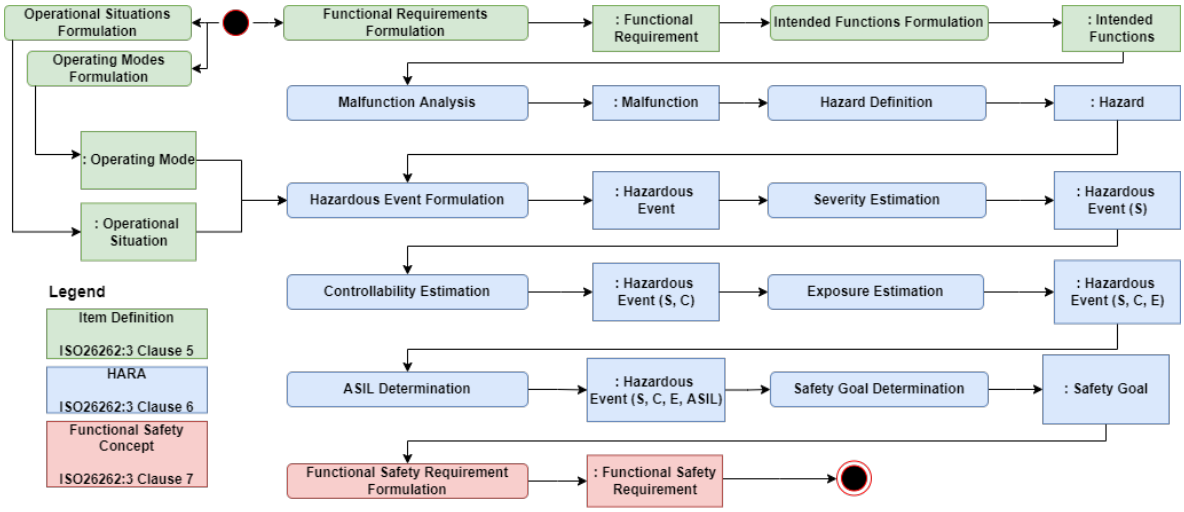

Figure 3.3: Functional Safety Flow Model

1. *Safety mechanisms* that realize the failure prevention or mitigation specified in the FSRs. The safety mechanisms are determined via a process called safety analysis.
2. *Technical Safety Requirements (TSRs)* that specify the safety mechanisms.
3. *Specification of the tests* required to ensure that the failure mechanism works correctly.

Continuing with the vehicle detection system example, the following theoretical safety mechanisms are given as examples:

- Data and timestamp checking mechanism that checks whether data is lost, or is rendered too late. In case of either type of failure, the user is warned of the issue and reminded to use necessary driving precautions such as checking rear-view mirrors.
- One or more redundant data channels with a switch that forwards the data arriving fastest. The switch also serves to check if there is data available on at least one channel within the scheduled time slot. In case of data unavailability, the same warning is given as for the data and timestamp checking mechanism.

3.2.2 System-level Phase Flow

The system-level phase is addressed in Part 4 of the ISO26262 standard. While it does not enforce a particular safety analysis method, the standard requires that the employed method identifies and removes internal and external sources of failure. Where this is not possible, the failure effects are to be prevented or mitigated.

Two of the most common safety analysis methods are the *Fault Tree Analysis (FTA)* and the *Failure Mode and Effect Analysis (FMEA)*:

- The FTA is a safety design method that helps list and visually represent possible component-level faults that can contribute to system-level malfunctions. A tree-like graph is derived based

on logically connecting system level failure events with component level faults using logical gates. For a more detailed description of the FTA, see [7].

- The FMEA, in addition to listing system failures and their component-level causes, helps specify and prioritize sources of failure and the actions needed to mitigate or prevent failure effects. While not using an easily readable tree-like graph, the FMEA better documents the effects of failures on the user, the occurrence probability of the listed failures, and the chance of them being detected during testing. This input is then used to list the design changes or tests needed to satisfy the functional safety requirements.

3.2.3 FMEA Description

The current work makes use of the FMEA method, as it results in better documented failure causes and actions necessary to improve safety. For this reason, this subsection describes this safety analysis method as performed in this work. An application of FMEA on the proof-of-concept system is described in Section 4.3 and Appendix B.

FMEA is done in multiple cycles of analysis, targeting failure modes at various abstraction levels. The first cycle is done at system level, followed by another cycle at component-level. Further cycles can follow as needed, analyzing failure modes of sub-components. Each cycle has 9 steps, as follows:

1. *Listing of functions*: This step consists of listing the system or component-level functions, depending on the cycle. The system-level listing of functions is identical to the intended function formulation step of Section 3.1.2, which can be re-used. Component-level functions may be more technical as they describe the roles of the software components in realizing higher-level functions.

2. *Listing of potential failure modes*: This step consists of brainstorming on the potential failure modes of the intended functions. This step is similar to the malfunction listing of the HARA in Section 3.1.2. The difference is that FMEA requires a precise technical description of the failure, so the necessary safety mechanisms and tests can be determined to ensure the risk of failure is within acceptable bounds.

3. *Specifying failure mode effects*: In this step, the FMEA requires specifying the effects of the failure modes listed in the previous step. This is similar to the specification of hazards caused by malfunctioning behavior of Section 3.1.2. Once again, a more detailed description is necessary to help in determining the right safety mechanisms and tests.

4. *Scoring failure mode effect severity*: The FMEA then requires scoring of the failure mode effects *severity*. This project uses the criteria of Appendix B, Table B.1.

5. *Specifying potential cause(s) of failure modes*: This step consists of describing the precise technical causes of the failure modes. These can be attributed to sub-component malfunctions and addressed in the next cycle(s).

6. *Scoring failure mode occurrence*: The FMEA then requires scoring of the failure mode *occurrence*, understood as the combination of factors affecting the chance of a failure event. These are:

   (a) Clear understanding of the requirements and technology used.

(b) Experience with the given requirements, technology, design, and customer.

(c) Degree to which failure prevention methods are present and documented.

The *occurrence* level is the summed scoring of the above factors, plus one. The scoring is done according to the criteria of Appendix B, Table B.2.

7. *Scoring failure mode detection*: The final scoring is for failure mode cause *detection*, understood as the chance of the failure mode being detected through testing. This project uses the detection scoring criteria of Appendix B, Table B.3.

8. *Calculation of Risk Priority Number (RPN)*: The FMEA calculates the RPN as the product of the severity, occurrence and detection scores. The purpose of the RPN is to prioritize failure modes, so its exact value matters less than the resulting prioritization of issues to be solved by safety mechanisms and tests.

9. *Listing of actions to be taken to reduce the RPN below acceptable limit*: The final step of an FMEA cycle is brainstorming on actions that reduce the RPN below a given threshold. The actions can be either design of a safety mechanism, or testing practices that help reduce the chance of failure below an acceptable level. Testing is at least required to ensure good functioning of the safety mechanism, as there may not be a second mechanism that mitigates or prevents malfunction.

Once the system cycle is completed, the component and sub-component cycles will make the system-level failure mode causes more specific, as these are traced back to their relevant system parts. The FMEA is complete once the engineering team is satisfied with both of the following:

- Sufficient failure modes are brainstormed, depending on system scope and target safety level.
- The failure modes are mitigated or prevented by the safety mechanisms specified in Step 9. Where this is not possible, testing is planned to ensure the function or safety mechanism behaves as specified.

Cycles can of course be repeated as needed, such as when there are design changes, added functions, or new failure modes are determined for existing functions.

3.3 Software-level Safety Phase

The purpose of the software-level safety phase is to ensure that appropriate software design, implementation, and verification principles are followed, given the system ASIL level. ISO26262 Part 6 addresses software-level safety and specifies guidelines on the following Topics:

1. Software safety requirements.
2. Software architectural design.
3. Software implementation.
4. Software unit and integration testing.

The specific guidelines for each topic are listed in Appendix C Section C.1. No example or flow description is given for software-level safety, as the standard only specifies rules on the topics above and no process for software safety is documented. The rules, alongside general software implementation practices, suffice in generating safe software.

3.3.1 Software Implementation Safety

The current work covers only application of rules for Topic 3: *software implementation*. Verifying that these rules are met is done via static code analysis, which is the practice of identifying safety issues in the code during its development, and before its deployment and execution. It ensures that the code statements are unlikely to lead to any unexpected behavior when the executable is built with multiple compilers or run on multiple execution environments. One code statement may execute differently when any of these are changed. To avoid such errors, static code analysis tools scan the code and checks for compliance with software implementation guidelines.

In addition to static code analysis, testing of the code is also covered in ISO26262:6. Also known as dynamic code analysis, testing ensures the code behaves as expected during its execution. The current work does not cover testing as most of the ASIL A-B required testing is already done within TomTom. The standard requires 100% branch coverage and the company already practices testing with 80% branch coverage.

Section 4.4 covers an application of software safety on the proof-of-concept and provides examples of static code analysis execution and output.

4 Functional Safety Application

4.1 Blind-Spot Highlighting Use Case

To design the proof-of-concept system, the definition of a use-case is needed for understanding which function the system realizes. Figure 4.1 shows the use cases considered, which are described below:

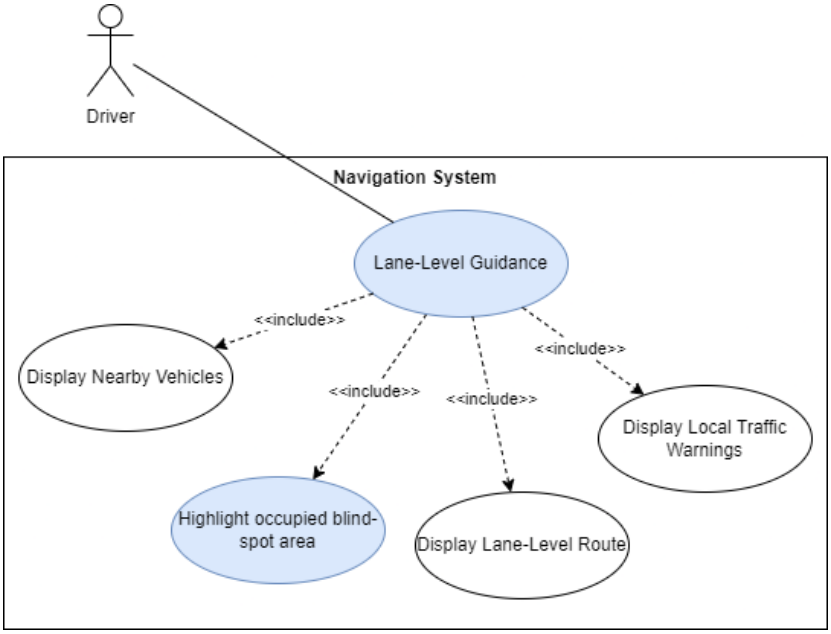

Figure 4.1: Navigation System Use Cases

- *Display Nearby Vehicles*: The system displays nearby vehicles as the one shown in the IndiGo example of Figure 1.4.
- *Highlight occupied blind-spot area*: The system displays a red-colored highlighting when a vehicle is detected in the left or right blind-spot. A possible realization of the use-case is shown in Figure 4.2, where a TomTom simulation displays red highlighting of a vehicle in the blind-spot. The IndiGo example of Figure 1.4 displays red-colored highlighting when there are vehicles both in the blind-spot and on the side of the driver's car. This use-case however considers the blind-spot angle only.
- *Display Lane-Level Route*: The system displays the lane-level route as a curved carpet in front of the car. The carpet serves to guide with lane-switching maneuvers. Figures 1.3 and 1.4

Figure 4.2: TomTom Blind-Spot Highlighting Simulation

illustrate the carpet in white in front of the driver's car.

- *Display Local Traffic Warnings*: The system displays warnings related to local traffic, such as traffic jams. The TomTom simulation of Figure 4.3 shows a traffic warning on the planned route. The IndiGo example in Figure 1.4 also shows a traffic warning in the upper-right corner.

Figure 4.3: TomTom Traffic Warning Simulation

The current work addresses blind-spot highlighting, as it is considered the simplest use case involving safe driving assistance. Life-threatening accidents can occur in the event of a system malfunction and a careless driver at the wheel, which makes the need for safety design easy to understand. Once the use case is determined, the safety design of the system on a functional level can begin. This is documented in the next section.

4.2 Deriving the Functional Safety Concept

4.2.1 Item Definition

To help define the system functions, the blind-spot highlighting system is represented using the "black-box" view, shown in Figure 4.4. The system receives sensor data and uses it to create render data, which is passed on to a display, where the blind-spot areas are highlighted accordingly.

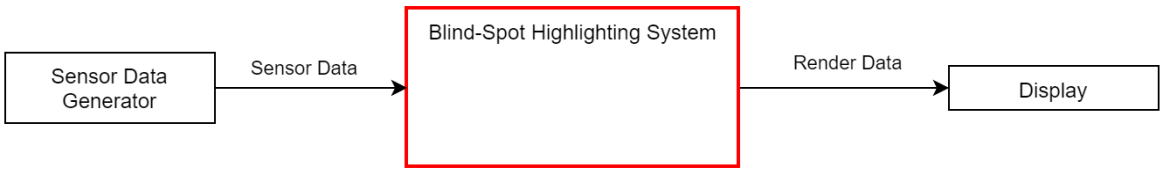

Figure 4.4: Blind-Spot Highlighting: Black-Box View

Intended Functions

Following the concept phase flow, the system intended functions are listed. The system however has only one intended function, specified below:

Intended Function 1: Highlight a blind-spot area whenever the input data reports the area as occupied.

The function is sufficient for describing the functionality realized by the blind-spot highlighting system. The function is specified in the functional requirements listed in Table 4.1.

Table 4.1: Functional Requirements

Id	Short Description	Description	Rationale
LLG-BLSP-FR01	Read blind-spot sensor data.	Whenever the vehicle provides the system with blind-spot data, the system shall convert it to data that can be rendered.	To highlight occupied blind-spot areas, the system must be able to read the blind-spot data.
LLG-BLSP-FR02	Highlight occupied blind-spot area.	When the system reads that a blind-spot area is occupied, the system shall render it as occupied.	The cluster navigation system must inform the driver of occupied lanes.
LLG-BLSP-FR03	Stop highlighting when blind-spot area no longer occupied.	When the system reads that a blind-spot area is not occupied and if the area is highlighted, the system shall render it as not occupied.	The cluster navigation system must inform the driver of occupied lanes.

Operating Modes and Operational Situations

The only operating mode considered is the system being on and performing *intended function 1*. The operational situations considered abstract from environment conditions such as lighting or weather, which influence visibility. They are reduced to the actions that the driver can make, as these are ultimately the main causes of hazardous situations. Thus the operation situations are summarized to a careful driver operating the vehicle versus a clumsy driver doing so.

4.2.2 Hazards Formulation

Once the system is defined at a functional level, the next step is documenting possible hazards caused by malfunctions. The technique used to identify hazards is the *Hazard and Operability Analysis (HAZOP)* [10], which uses keywords that help spot deviations from intended design [11]. The resulting hazards are summarized in Table 4.2. The full HAZOP process is described in Appendix A Section A.1.

Table 4.2: Hazards resulting from malfunctions of the Blind-Spot Highlighting System

Function	Hazard ID	Hazard Description
Highlight a blind-spot area whenever the input blind-spot data reports it as occupied	LLG-BLSP-H01	System shows vehicle in blind-spot when there is none.
	LLG-BLSP-H02	System shows no vehicle in blind-spot area when there is one.
	LLG-BLSP-H03	System shows wrong blind-spot area as occupied.
	LLG-BLSP-H04	System shows vehicle entered blind-spot too late.
	LLG-BLSP-H05	System shows vehicle left blind-spot too late.

4.2.3 HARA

Once the hazards have been identified, the HARA can be performed to obtain the ASIL levels of system-level functions and safety goals that increase safety in case of hazardous events. For brevity, this section contains a summarized version of the HARA done for the blind-spot highlighting system, shown in Figure 4.3 . The full table can be found in Appendix A Section A.2. In this case, the safety goals can be combined in one safety goal, which inherits the highest ASIL level among the former. The combined safety goal is shown in Table 4.3, using the following abbreviations:

- SG ID - Safety Goal ID.
- CSG - Combined Safety Goal, which the safety goal of the respective row is combined into.

4.2.4 Functional Safety Requirements

Finally, the FSRs that satisfy the safety goals are formulated. These inherit the highest ASIL of the safety goals they address. The FSRs are listed in Table 4.4, using the following abbreviations:

- FSR ID - Functional Safety Requirement ID.
- SG ID - ID of the Safety Goal supported by the respective FSR.

Table 4.3: Safety Goals and ASIL levels

SG ID	ASIL	Safety Goal Description	Target Hazard(s)	CSG	Rationale
LLG-BLSP-SG04	B	**Ensure blind-spot area is highlighted with acceptable maximum latency after a vehicle is detected in that area.**	LLG-BLSP-H04	LLG-BLSP-CSG01	Summarized to safety goal about output being correct and on-time. Exceeding maximum latency is a subset of not arriving on time.
LLG-BLSP-SG03	B	**Ensure the correct blind-spot area is highlighted when the vehicle enters that area.**	LLG-BLSP-H03	LLG-BLSP-CSG01	Summarized to safety goal about output being correct and on-time. Wrong blind-spot area is a subset of incorrect output.
LLG-BLSP-SG02	B	**Ensure blind-spot area is always highlighted when there is a vehicle in that area.**	LLG-BLSP-H02	LLG-BLSP-CSG01	Summarized to safety goal about output being correct and on-time. False negative is a subset of incorrect output.
LLG-BLSP-SG01	QM	**Ensure blind-spot area is never highlighted if there is no vehicle in that area.**	LLG-BLSP-H01	LLG-BLSP-CSG01	Summarized to safety goal about output being correct and on-time. False positive is a subset of incorrect output.

CSG ID	ASIL	Combined Safety Goal Description	Note
LLG-BLSP-CSG01	B	**Ensure blind-spot areas are highlighted correctly and within a given FTTI.**	FTTI is an input parameter to the safety mechanism.

Table 4.4: Proof-of-concept FSRs

FSR ID	FSR Description	SG ID	ASIL
LLG-BLSP-FSR01	The system shall verify that no blind-spot data is lost until displayed.	LLG-BLSP-CSG01	B
LLG-BLSP-FSR02	The system shall verify that the blind-spot data values are within given boundaries.	LLG-BLSP-CSG01	B
LLG-BLSP-FSR03	The system shall verify that the blind-spot data is read, processed and rendered within a given FTTI.	LLG-BLSP-CSG01	B

4.3 Deriving Safety Mechanisms and Tests

After determining the safety goals and FSRs, the current work introduces the design of the blind-spot highlighting system. This design is subjected to an FMEA safety analysis. The result is a new design with the necessary safety mechanisms is place.

4.3.1 System Design

The initial design of the system is shown in Figure 4.5. The reading, processing, and rendering of the blind-spot sensor data is allocated to three software components respectively: data input, data processing, and renderer. The data source and display are input and output systems outside of the design scope. The segmented shapes preceding the data processing and renderer components represent data buffers, in case they receive data at a faster rate they can process. The initial design does not detail the communication protocols or data format, which are detailed in the software safety phase.

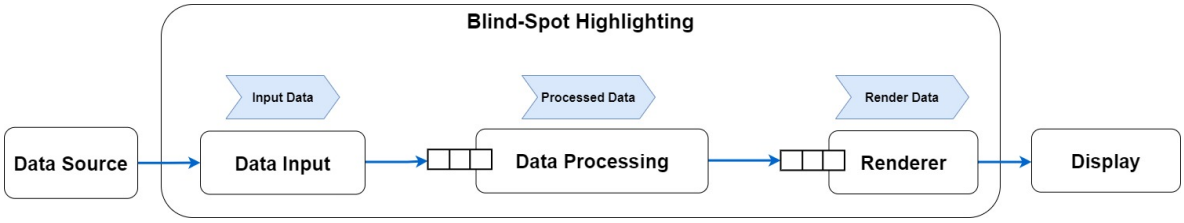

Figure 4.5: Initial System Design

4.3.2 FMEA

Once the initial design is obtained, the safety analysis for deriving the safety mechanisms can be performed. For brevity, this section summarizes both the system and component-level FMEAs to the list of identified failures and safety mechanisms that mitigate them. The full FMEA can be found in appendix B, Section B.2.

Figure 4.6 shows the post-FMEA design, featuring the added safety mechanism. The changes to the initial design are summarized below:

- The data input and data processing components check if the input data cannot be read or if the data received is erroneous. In either case, the renderer is signaled to provide the display with error warning render data. The display then warns the user that the blind-spot highlighting system cannot be relied upon and the driver should be cautious.
- Communication between components is done through reliable mechanisms that prevent data loss.
- All components generate timestamp upon start of their respective data handling. Each timestamp is associated with the data for which it is generated. Components check if time difference between generated timestamps and previous component timestamps exceeds FTTI.

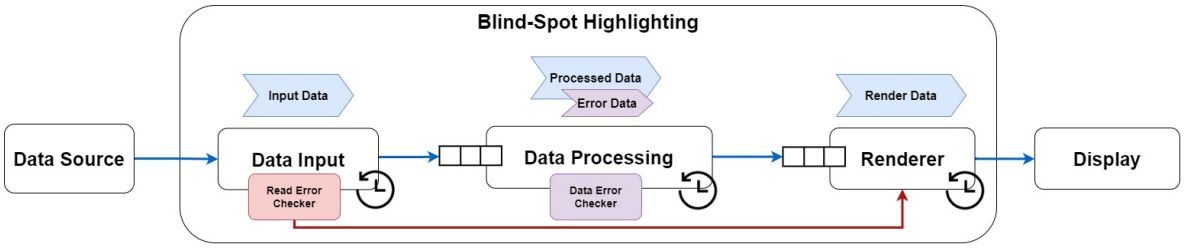

Figure 4.6: Post-FMEA System Design

4.4 Software-level Safety Application

This section describes the proof-of-concept system implementation and static code analysis.

4.4.1 Proof-of-Concept Implementation

The aim of the proof-of-concept is to simulate an actual blind-spot highlighting system that reads vehicle sensor data and displays the corresponding traffic situation on the cluster display. The current implementation simulates this functionality by reading files containing manually generated sensor data, processing it, and displaying the corresponding image in a browser.

Proof-of-Concept Components

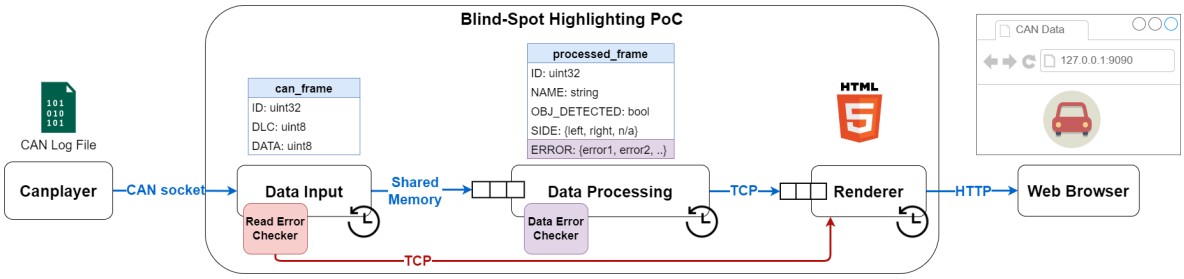

Figure 4.7: Post-FMEA System Design

Figure 4.7 illustrates the implementation of the proof-of-concept system. The data source and display components of Figure 4.6 are replaced with the *Canplayer* and *Web Browser* components respectively. The list below explains the implementation of each component:

1. *Canplayer*: Reads the manually generated sensor data from CAN log files and makes it available to a specified CAN socket. This function, also known as replaying CAN data, is realized by the already existing canplayer software tool, which is part of the can-utils toolset [8] The tool replays the contents of CAN log files to simulate vehicles entering and leaving the left and right blind-spot areas.

2. *Data Input*: Uses the Linux *socketCAN* library to read the data provided by *Canplayer* and passes it on to the *Data Processing* component through shared memory. During the *Data Input*

phase, the CAN frame ID, DLC and DATA fields are read and stored in a *can_frame* object. Extracting the meaningful information from the *can_frame* is done in the *Data Processing* phase.

3. *Data Processing*: Reads the data provided by the *Data Input* component and extracts the meaningful information from the *can* ID, DLC and DATA fields, using it to determine blind-spot status and errors. The component then forwards the blind-spot and error data to the *Renderer* via TCP loopback.

4. *Renderer*: Uses the data provided by the *Data Processing* component to render HTML code visualizing the blind-spot detection situation and system error status. The data is transformed to HTML using the Wt [6] library.

5. *Web Browser*: The cluster display is simulated by using a web-browser to visualize the html data provided by the *Renderer*.

Proof-of-Concept Simulated Display

To visually observe the blind-spot highlighting function, the proof-of-concept displays the sensor data both in text and as images simulating a cluster display. Figures 4.8 to 4.12 show the browser display of the blind-spot highlighting simulation. On the left side, the display shows a text output that reports the status of the blindspot sensors. *STATUS_LBSS* and *STATUS_RBSS* are names assigned to the CAN frames reporting the blind-spot sensor detection status, for the left and right sensor respectively. The right side shows a representation of the driver's vehicle and vehicles present in the blind-spots when these are detected.

The log files read by the canplayer follow the CAN specification of a TomTom Jeep Renegade and can be found in Appendix D. One log file contains data simulating an error-free scenario and another contains erroneous CAN frames. The simulated scenarios are the following:

- Error-free scenario: no data errors are detected. The scenario simulates the driver vehicle being overtaken from different sides as follows:

    1. The scenario starts with an initial state with no vehicles overtaking. This is shown by the *Nothing in Blind-Spot* message under *STATUS_LBSS* and *STATUS_RBSS* simultaneously. This state is captured in Figure 4.8.
    2. The next state simulates a vehicle overtaking on the left side. This is shown by the *Object in Blind-Spot LEFT* message under *STATUS_LBSS*. This state is captured in Figure 4.9.
    3. The left-side vehicle then finishes overtaking and thus no objects are detected in any blind-spot.
    4. A vehicle then overtakes on the right side, shown by the *Object in Blind-Spot RIGHT* message under *STATUS_RBSS*. This state is captured in Figure 4.10.
    5. The right-side vehicle then finishes overtaking and thus no objects are detected in any blind-spot.
    6. Two vehicle then overtake on both the left and right sides, shown by the corresponding messages. This state is captured in Figure 4.11.

- Error-prone scenario: simulates the same sequence of events as the error-free scenario, with errors inserted between Steps 1 and 2. The errors displayed are the following:

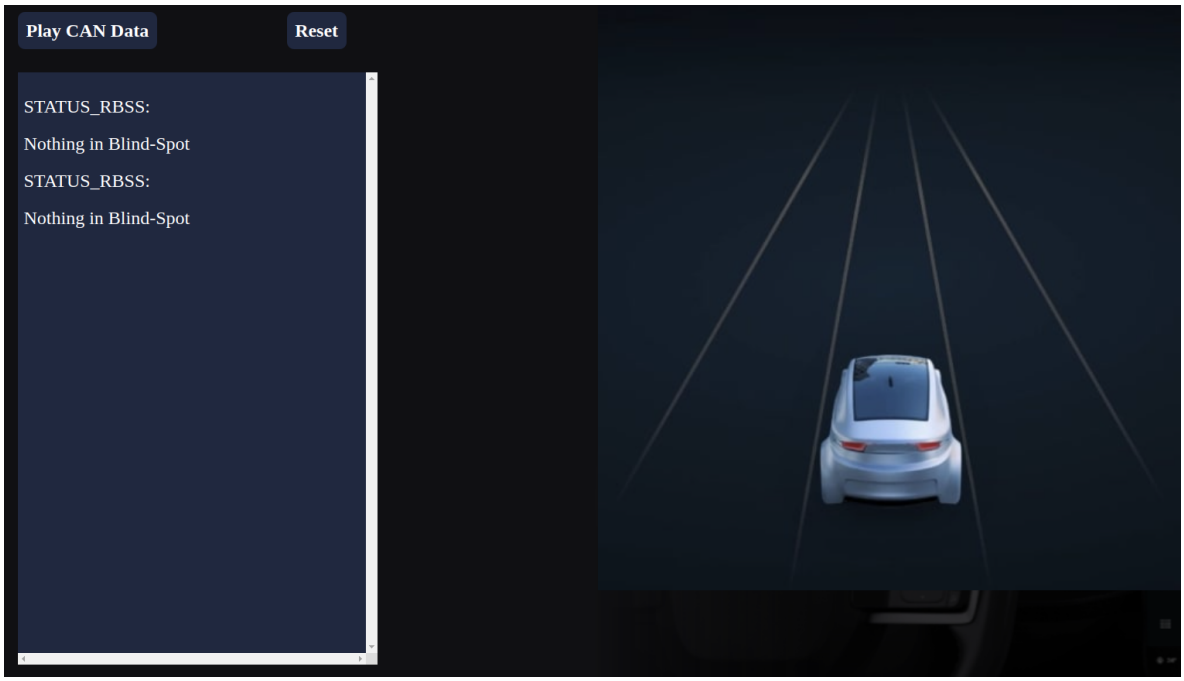

Figure 4.8: Proof-of-concept simulation - no objects in blind-spots

1. *ERROR_ID_UNKNOWN*, indicating that the CAN frame ID is not recognized.
2. *ERROR_DATA_EXCEEDS_LENGTH* indicating that a frame has been read with a known ID however, the data size is larger than the DLC value.
3. *ERROR_DATA_MISSING* indicating that a frame has been read with a known ID, however the data size is smaller than the DLC value.

Figure 4.12 shows the state reached after one frame with unknown ID.

4.4.2 Static Code Analysis

To analyze the effects of coding guidelines enforcement, the code is initially implemented in the C++14 language without considering any safe coding practices.

The current work uses Coverity [20] to scan the code and determine its compliance with ISO26262:6 software implementation guidelines, which can be found in Appendix C, Table C.1. The guidelines are not language specific and can manifest in different code statements depending on the programming language used. The AUTOSAR C++14 ruleset [2] provides the necessary input for the Coverity scanner to recognize when the proof-of-concept code violates any implementation rules of Appendix C, Table C.1.

The proof-of-concept contains 3925 lines of C++ code, excepting the third party libraries. The Coverity scanner provided the following results:

- 1863 total issues. These are the sum total of issues raised by Coverity for the 3925 lines of code. They include both ISO26262:6 relevant and non-relevant issues.

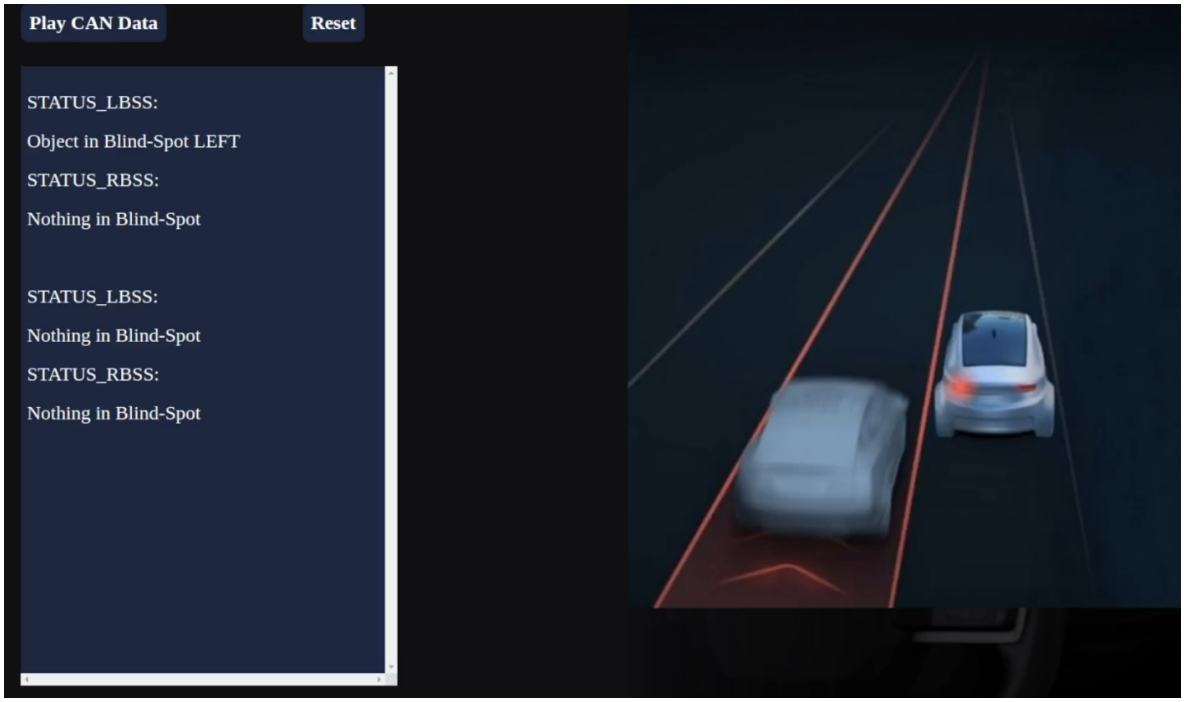

Figure 4.9: Proof-of-concept simulation - object in left blind-spot

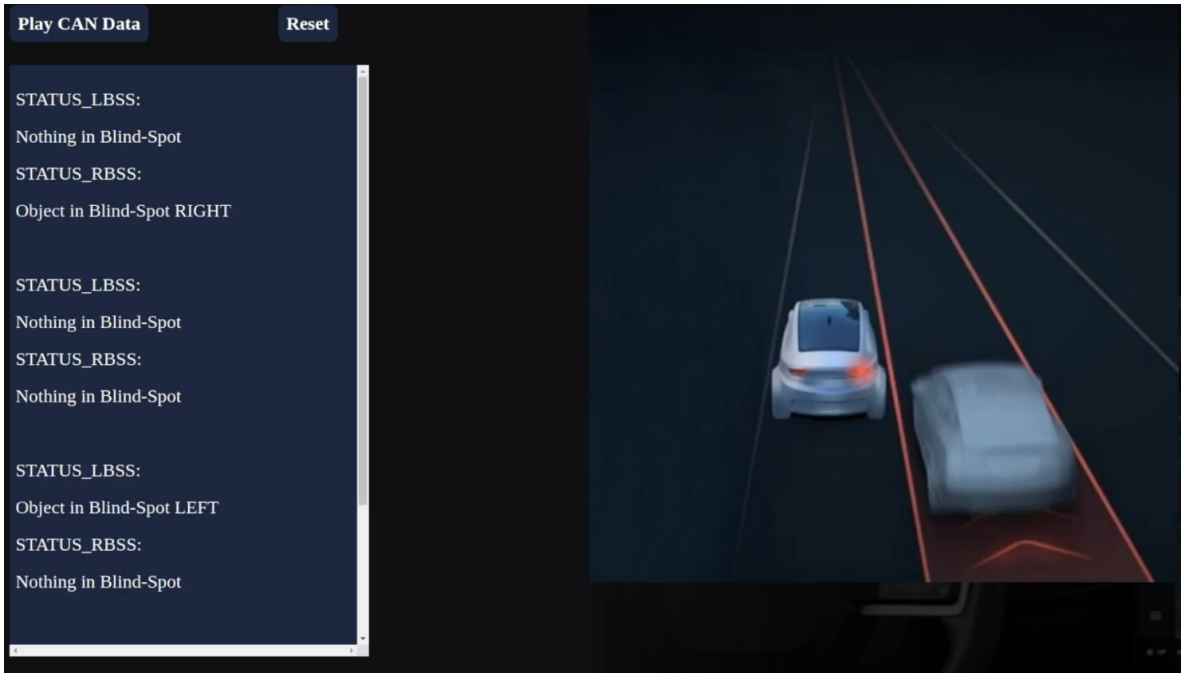

Figure 4.10: Proof-of-concept simulation - object in right blind-spot

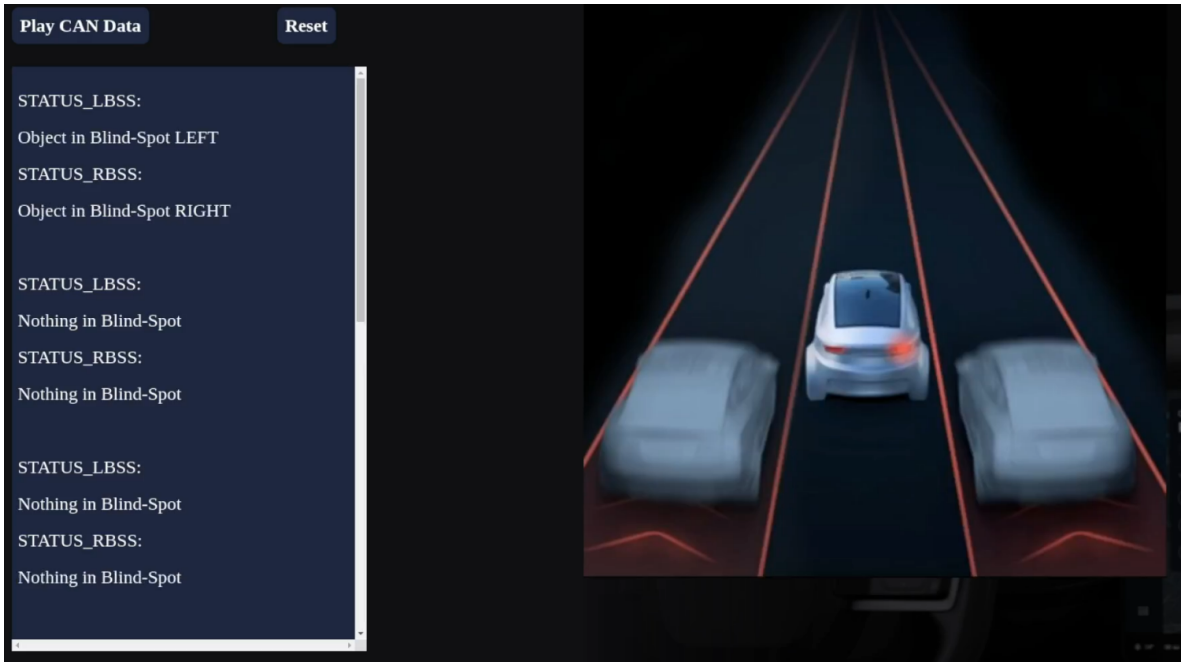

Figure 4.11: Proof-of-concept simulation - object in left and right blind-spots

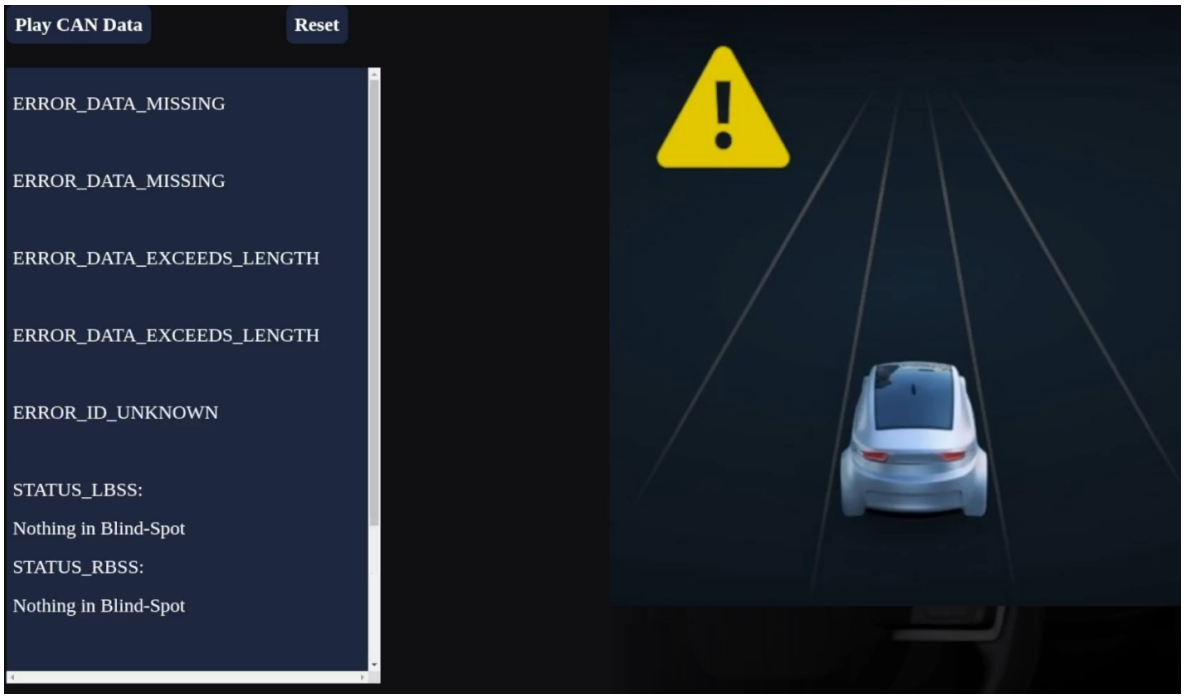

Figure 4.12: Proof-of-concept simulation - example errors

- 490 ISO26262:6 relevant issues, that the AUTOSAR C++14 document maps to the functional safety standard guidelines.

Figures 4.13 and 4.14 show the Coverity reports for total and ISO26262:6-relevant issues respectively:

- Figure 4.13 shows a screenshot of the Coverity interface, listing the 1863 total issues. An example of a non-relevant issue is the one selected in the Figure. The selected issue pertains to the AUTOSAR C++14 M7-3-1 rule, which has no mapping to the requirements of ISO26262:6 listed in Table C.1. Rule M7-3-1 is listed in the AUTOSAR standard as: *The global namespace shall only contain main, namespace declaration and extern "C" declarations*. In the particular case of the proof-of-concept code, function `unpack_left_shift_u8` is found to be in the global namespace, which triggers reporting of the issue.
- Figure 4.14 shows a screenshot of Coverity listing the 490 relevant issues. An example of a relevant issue is the one selected in the Figure. It pertains to the AUTOSAR C++14 A4-7-1 rule, which maps to the ISO26262:6 requirement for use of defensive implementation techniques. The standard assigns the requirement to ASIL C, however the mapped AUTOSAR rule is deemed important due to potential change of an integer value. Rule A4-7-1 is defined as *An integer expression shall not lead to data loss*. The assignment of the returned value of the `read` function, which is a 64-bit signed long, to a 32-bit signed int variable, may cause data loss and thus `nbytes` triggers reporting of the issue.

The static code analysis concludes the application of the ISO26262 parts 3, 4 and 6 standard on the design and implementation of the blind-spot highlighting proof-of-concept system. Chapter 5 goes on to analyze the impact of the functional safety process on TomTom automotive software development.

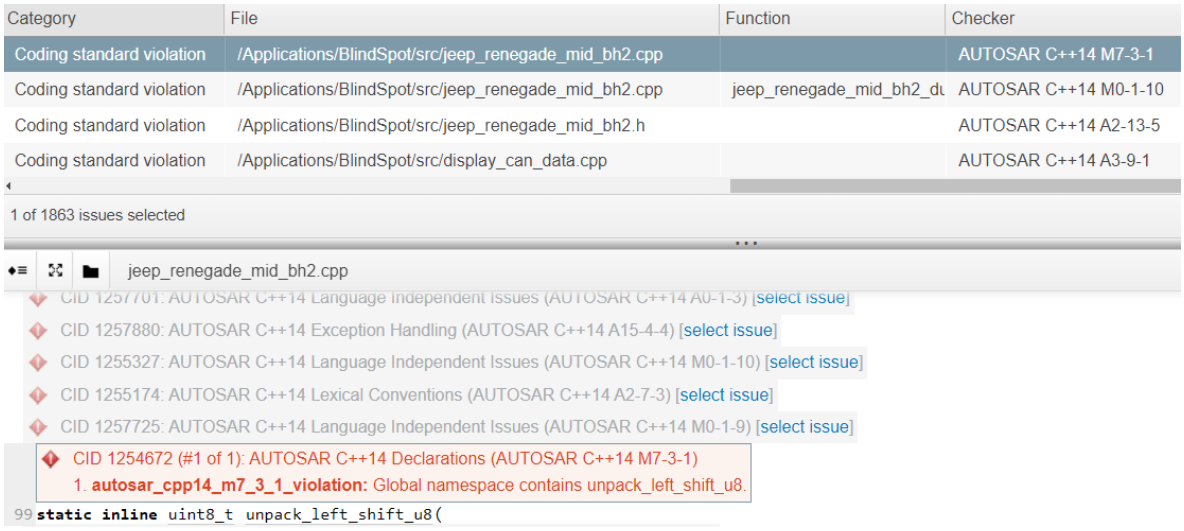

Figure 4.13: Proof-of-concept simulation - example errors

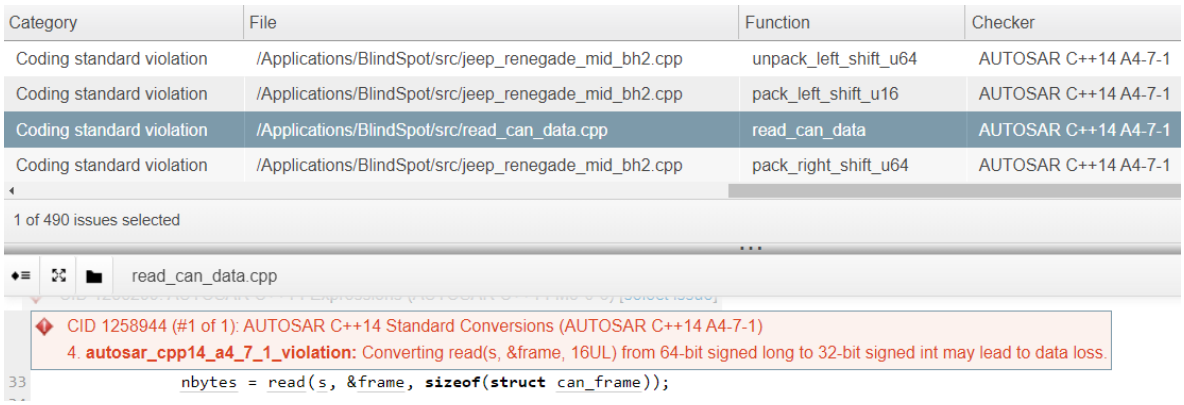

Figure 4.14: Proof-of-concept simulation - example errors

5 Benefits and Estimated Cost of Functional Safety

5.1 Benefits of Functional Safety

This section discusses the advantages the functional safety process for automotive software. The benefits of the concept, system and software phases are discussed individually, followed by the value added by the entire safety-by-design process. The current work argues that the benefits of each phase and the overall process generally outweigh the costs, therefore safety-by-design must be seriously considered when software companies enter the automotive space.

5.1.1 Functional Safety Concept Phase Benefits

The functional safety concept determines if and why a system's functions are safety related. Deriving the ASIL of a function first of all helps software companies decide whether to develop the function in the first place. A company can opt for the non-safety-compliant version of a function or system, with the risk of the product not being as easy to market as one designed for its respective ASIL. Should the company choose to design a safety-compliant product, the functional safety concept provides the goals and requirements to be achieved for compliance.

While deriving the safety goals and FSRs can take longer, understanding whether a function is safety-related or not, and even obtaining its ASIL can be done relatively quickly. The safety engineer needs to think about a function on a high-level, along with a possible worst-case combination of malfunctions, operating modes, and operational situations, to determine if the system is in the ASIL A-B range. Once that is determined the company can decide on pursuing development of the function or system. The current work argues that the benefits of safety-by-design generally outweigh the cost. This is discussed in detail in Section 5.1.4.

Figure 5.1 summarizes the benefits of the functional safety concept, which are detailed in the list below:

1. *Benefit 1*: Can be performed early, quickly and with low-cost. The HARA can answer whether a system is in the ASIL A-B category once a worst-case scenario malfunction is determined. It does not require technical system knowledge or human resources apart from the safety engineer and a system designer.
2. *Benefit 2*: As soon as the the ASIL is determined, the company can decide whether to proceed with safe system design or not.
3. *Benefit 3*: Provides context for further design. Should the company opt for a safe system design, the ASIL level will determine if the software is to be run in ASIL-compliant operating system

Figure 5.1: Benefits of safety concept

and hardware. Furthermore, as the safety mechanisms for ASIL A-B require failure-state data from sensors, the software company can already assume that data is provided by the vehicle.

5.1.2 System-level Safety Benefits

The system-level safety phase determines the system, component and sub-component vulnerabilities of the system that may cause malfunctioning behavior. In addition from defining the safety mechanism, the engineering team must also think of the following system aspects:

- How errors in the input data can lead to malfunctioning behavior.
- How the execution environment affects functionality and real-time behavior.
- How the communication protocols and operating systems affect data availability, integrity and real-time behavior.
- How software components can malfunction.
- How safety-mechanisms can malfunction.

The list is derived from the safety analysis performed on the proof-of-concept system. Analysis on more complex systems will likely lead to additional aspects. Figure 5.2 shows the benefits of the system-level safety phase, which are listed below:

1. *Benefit 1*: Safety analysis helps eliminate technical debt. Given the extensive scope of the system analysis, technical debt is likely to be significantly reduced or at least documented. For example, an FMEA will recursively analyse failure modes and effects, starting from the system-level, going down to component and sub-component levels. This generates discussion of the technical details of the system, and reporting what improvements can be done in terms of development, knowledge of requirements and technology, experience and testing.

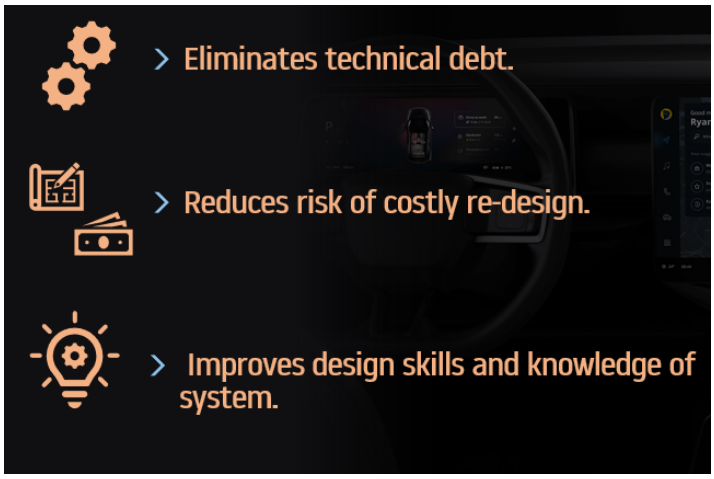

Figure 5.2: Benefits of system safety

2. *Benefit 2*: Improves design skills and knowledge of the system. Due to the extensive scope of the analysis, the engineering team brainstorms about all the aspects listed above, and likely more. Thus engineers will learn from each other on multiple aspects of the system.

3. *Benefit 3*: Drastically reduces the risk of a costly re-design. Re-design can lead to costs exceeding that of safety-by-design, which would have provided a safe and higher-quality faster. Figure 5.3 illustrates the additional cost incurred after an ASIL B redesign of the TomTom Autostream system [22], which provides high-definition map data to the vehicle for autonomous driving. The redesign has been estimated to add an additional cost of 31.5% more than what the cost would have been if safety-by-design was opted for.

5.1.3 Software-level Safety Benefits

The software safety phase ensures that the code used to implement the system behaves as expected, regardless of the development, deployment and conditions under which it is run.

Configuration work is required to enable static code analysis in combination with the various build environments used by software teams. Tools such as Coverity can be configured to provide real-time analysis via integrated development environment (IDE) extensions. While the process of configuration can be quite lucrative, once done, the cost of software safety reaches its minimum. As a result, the company also gains understanding of its configuration practices and build environments. An example outcome of configuration optimization are ready-made and documented project configurations. These can be used depending on the required build environment, having static code analysis tools available to support the software safety process.

Figure 5.4 shows the benefits of the software safety process, which are detailed in the list below:

1. *Benefit 1*: Errors are avoided with minimal cost. Static code analysis signals the errors that may occur before the code is deployed and executed. These can thus be resolved in the earliest software implementation phase.

2. *Benefit 2*: Resilience against development and deployment environment change. The code is

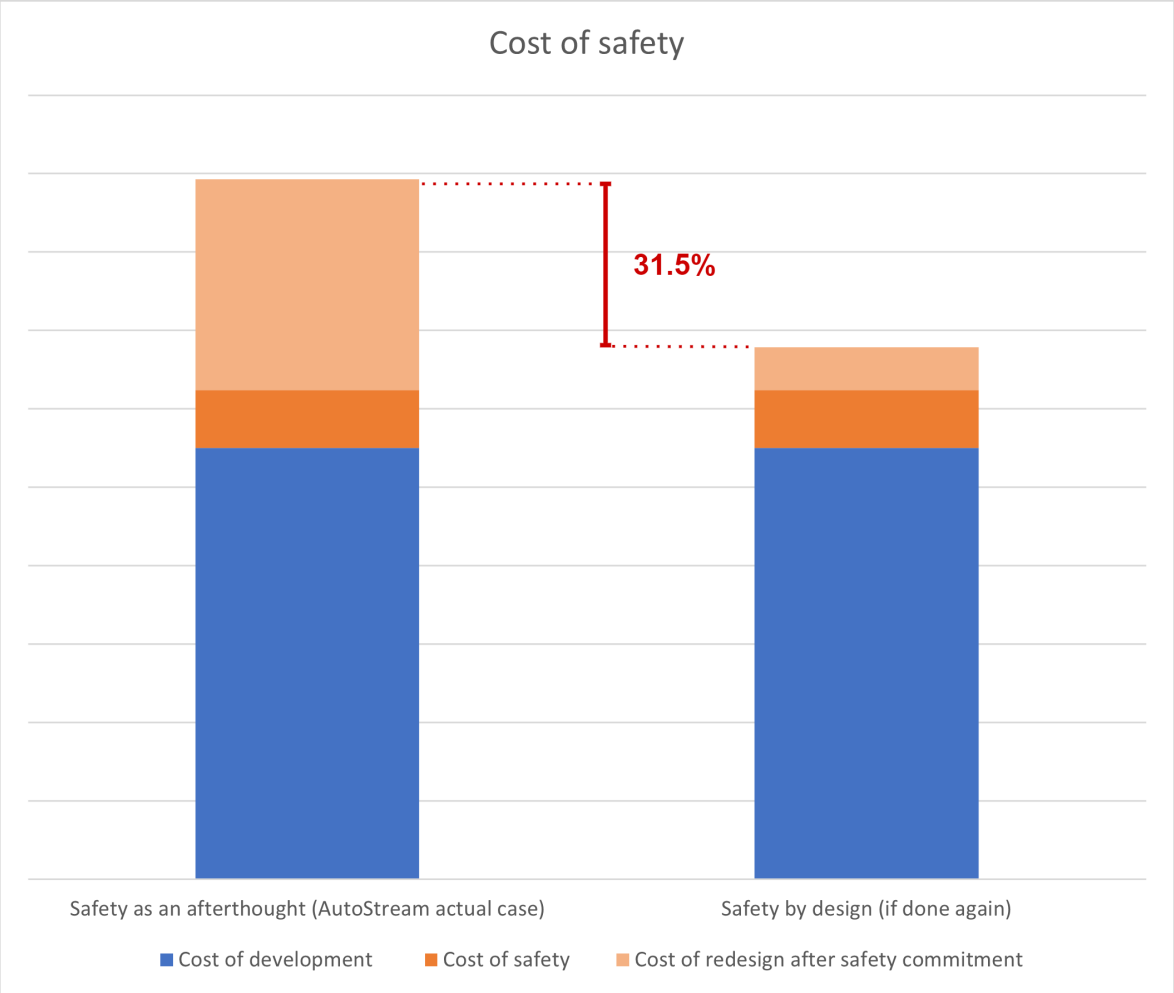

Figure 5.3: Cost of TomTom software redesign

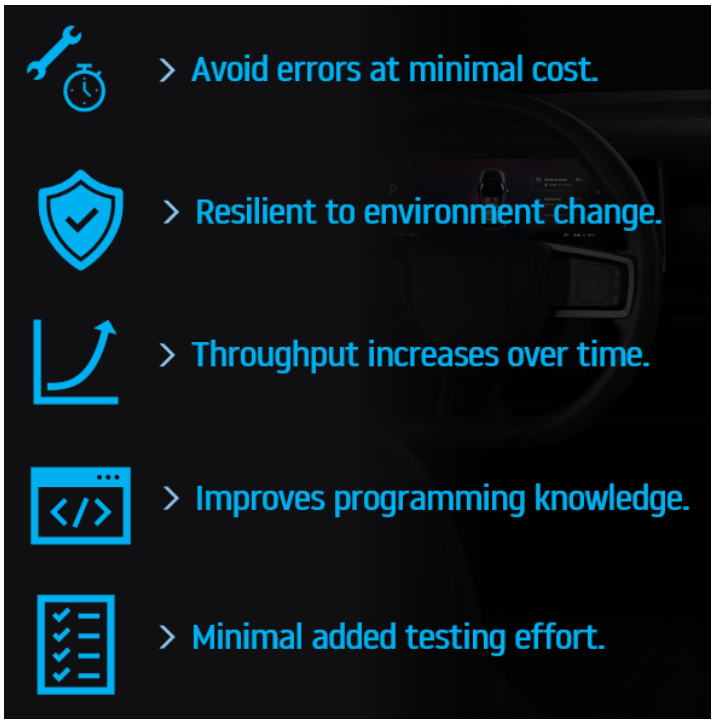

Figure 5.4: Benefits of software safety

more likely to behave as expected when built with different compilers or deployed on different operating systems or hardware architectures.

3. *Benefit 3*: Throughput increases with time. As the company optimizes configuration of software tools and engineers learn safety programming practices, the time spent altering the code to satisfy software-safety guidelines is reduced.

4. *Benefit 4*: Improves programming knowledge. As exemplified in Figures 4.13 and 4.14, the issues reported help understand the inner-working and non-obvious effects of the lines of code used. Extensive side-effects and less known behavior is covered by the code analysis rules, thus the lessons that can be learned are also extensive.

5. *Benefit 5*: Testing effort will likely not be significantly larger than that of existing practices. In the case of TomTom, testing branch-coverage must increase from 80% to 100%, which does not affect the existing configuration and testing process, but only the coverage.

5.1.4 Benefits of Safety-by-Design

In addition to the technical quality and safety added by the three phases of safety-by-design, the overall process provides advantages on the business-level. The current work asserts that companies must seriously and rationally consider safety-by-design in its software product development. The access to large sectors of the automotive market depend on a software company's willingness to be ASIL compliant. Furthermore, the current work argues that the business and technical benefits of safety outweigh the costs.

Figure 5.5: Benefits of safety-by-design

Figure 5.5 shows the business-level benefits of the safety-by-design process, which are detailed in the list below:

1. *Benefit 1*: Higher-quality and safer product. This is a technical as well as a business benefit. Higher-quality products lead to higher customer satisfaction, thus improves customer's confidence that the company will provide good services.
2. *Benefit 2*: Out-of-the box ASIL product. Car manufacturers or other customers may require that a product be developed at a certain ASIL. Employing safety-by-design from the start will prevent redesign, the customer having to wait or possibly walking away.
3. *Benefit 3*: Greater access to automotive customers. Having ASIL-ready products enable software companies to more easily enter the automotive market, where safety and technical quality requirements are more demanding. Should companies not consider safety-by-design, companies risk falling behind, if not completely being excluded from the automotive space.
4. *Benefit 4*: Possibility of diversifying in the automotive market. Along with access to the automotive market comes the possibility of diversifying a company's software products. As safety processes are integrated in software design, the processes and expertise can be re-used to design more products in the automotive space.
5. *Benefit 5*: Proof of safe design and implementation. Should there be a malfunction or accident, the software company will have proof that the appropriate safety processes are present in the development cycle. Vehicle callbacks or tragic accidents can severely impact a company's

reputation and finances. In April 2022, Mercedez-Benz USA recalled 126,443 vehicles [16] due to a bad failure mechanism of the rearview camera software. The issue has been traced back to a deviation in the development process of the software supplier.

6. *Benefit 6*: Low safety requirements. Commercial software companies entering the automotive space will likely not develop ASIL C-D functions. Thus company processes can be optimized for the lower cost ASIL A-B range.

5.2 Estimated Cost of Functional Safety

Finally, this section documents the costs of the safety-by-design process applied to the proof-of-concept system. The functional safety concept is not considered in the cost analysis, as the current work considers it the responsibility of the safety engineer and not of the development team. The concept can be derived in parallel to the design phase of the system and so does not add up to the cost of the product development. Additionally, testing has also not been factored in the cost estimation. In the case of TomTom the additional testing cost estimated is of 25% of the existing effort, which is the increase from 80% to 100% branch coverage.

The cost of the safety-by-design process is shown in Table 5.6 . The cost estimation is that the safety processes highlighted in blue take up 32.69 % of the total cost, and 48.56% of the non-safety related work. This is an estimation of an initial cost, which is expected to decrease to the cost estimated in Table 5.7, where the cost of static code analysis is not considered. This optimized cost is obtainable once real-time code analysis configured into the development environment. This is already available and can be implemented as soon as the company is able. The optimized safety cost is of 16.66% of the non-safety effort and thus 20% of the total work. The current work asserts that the optimized cost is a more realistic expectation, as it can be reached through appropriate configuration and software engineering practices, which is not difficult for sotware companies.

Should testing be factored in, the total cost for safety would not exceed 25% of the entire development effort, as the extra testing work is increased by 25% at most. This cost is considered reasonable for the many benefits of safety-by-design. The current work thus asserts that software companies must seriously consider safety-by-design, as a way to increase product safety and quality, as well as gain access and diversify into the automotive market.

Phase	Duration	% of Total
System Design	1 week	13.46
FMEA + Safety Mechanism Design	1 week	13.46
Implementation	4 weeks	53.85
Coverity configuration and scan	1 week	13.46
Resolving code issues (estimation)	3 days (12.48% of 4 weeks, rounded up)	5.77

Phase	Duration (days)	% of Total
Safety	17	32.69
Non-Safety	35	67.31

Figure 5.6: Cost of safety - Unoptimized Process

Phase	Duration	% of Total
System Design	1 week	16.67
FMEA + Safety Mechanism Design	1 week	16.66
Implementation	4 weeks	66.67

Phase	Duration (days)	% of Total
Safety	5	16.66
Non-Safety	25	83.34

Figure 5.7: Cost of safety - Optimized Process

6 Conclusion

6.1 Results Summary

The current work applies the functional safety practices of ISO26262 to the blind-spot highlighting proof-of-concept system. The results obtained show that the advantages outweigh the costs when ASIL A-B safety is integrated in the development of commercial software products. Not only does safety allow a company to enter the automotive market, but provides numerous benefits in terms of customer satisfaction, risk, accountability, and technical skill of employees.

The current work shows that, if a commercial software company plans to enter the automotive space, it must seriously consider integrating safety in its software development processes. While immediate integration is not expected, the company can experiment with safety-by-design. Trial runs can help determine whether the advantages materialize, beyond the increase in software quality, which is an immediate and certain effect.

The safety analysis on the proof-of-concept system shows that automotive functions that report blind-spot presence are assigned ASIL-B. Should a HARA be performed on similar functions that provide the driver with local traffic awareness, the same ASIL will be reached. The study proves that, if a software company wishes to develop software for driver situational awareness, then it should adopt ASIL-B safety practices to be functional safety compliant. Market surveys and automotive customer inquiries can help the company find out how it should adapt for the future, using concrete feedback from stakeholders in the company's business space. Finally, the current work concludes that companies need safety-by-design for not only entering, but also thriving in the automotive space.

6.2 Future Work

The current work focuses on the technical aspects of ISO26262 functional safety and its impact on software development. Thorough analysis of the business effects of safety, and of design practices that fall outside of ISO26262 functional safety are in large part left out of scope. They represent a natural continuation to the current work and are left for further study. The list below details on recommended future work:

- *Study on how safety impacts software company business processes*: The current work focuses on the impact of safety on product or project-level development. The thorough study of the impact on company-wide processes and business is left as future work.
- *Study of optimal configuration management and software engineering practices for software-level safety*: While the recurrent cost of safety is low, the initial changes may not be trivial,

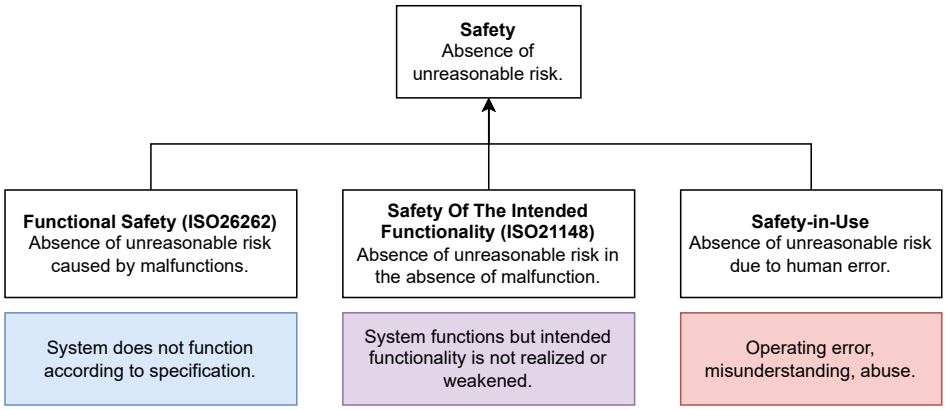

Figure 6.1: Sub-domains of Safety

depending on the skill and resolve of company engineers. A study on the role and best-practice for the safety engineer, software engineer and configuration manager can help companies understand the concrete changes necessary for aligning towards safety-by-design.

- *Study of safety-by-design practices falling outside of ISO26262*: Functional safety focuses on mitigation and prevention of system malfunction effects. Situations may occur that do not cause malfunctions, but prevent the system to realize its intended functionality. For example, occlusions on vehicle cameras caused by the environment can prevent a system from providing situational awareness, even though the system using the cameras works just fine. The system may still work as per specification internally, however external factors or insufficient performance can render the function impossible or reduce its effect. Such situations are addressed by the ISO21448 standard [12], which focuses on Safety Of The Intended Functionality (SOTIF) of road vehicles.

  Additionally, system misuse, due to driver misunderstanding the system User Interface (UI), can also lead to unsafe situations. More generally, safety-in-use and safe human-to-machine interaction also represent an important safety domain, becoming more important with the increase in vehicle intelligence and automation.

  Figure 6.1 represents the three safety sub-domains functional safety, SOTIF and safety-in-use. The technical and business-related aspects of SOTIF and Safety-in-Use are recommended as future study, as they will take center stage in automotive engineering, alongside functional safety.

Bibliography

[1] Introducing tomtom indigo - open digital cockpit software platform. URL:https://www.tomtom.com/press-room/general/143509/introducing-tomtom-indigo-the-worlds-first-open-digital-cockpit-software-p

[2] Autosar. Guidelines for the use of the C++14 language in critical and safety-related systems, October 2018. URL: https://www.autosar.org/fileadmin/user_upload/standards/adaptive/18-10/AUTOSAR_RS_CPP14Guidelines.pdf.

[3] Autosar. Part 1 - The AUTOSAR Partnership and Standardization, July 2021. URL: https://www.autosar.org/fileadmin/user_upload/AUTOSAR_EXP_Introduction_Part1.pdf.

[4] Ondrej Burkacky, Johannes Deichmann, and Jan Paul Stein. How will changes in the automotive-component market affect semiconductor companies?, Jun 2019. URL: https://www.mckinsey.com/industries/semiconductors/our-insights/how-will-changes-in-the-automotive-component-market-affect-semiconductor-c

[5] Embitel. Understanding How ISO 26262 ASIL is Determined for Automotive Applications, April 2018. URL: https://www.embitel.com/blog/embedded-blog/understanding-how-iso-26262-asil-is-determined-for-automotive-applications.

[6] Emweb. Webtoolkit library. https://www.webtoolkit.eu/wt/, 2022.

[7] George A, Taylor W., Nelson J. Writing Good Technical Safety Requirements. SAE Technical Paper, January 2016. doi:10.4271/2016-01-0127.

[8] Oliver Hartkopp et. al. Can-utils. https://github.com/linux-can/can-utils, 2022.

[9] Functional safety of electrical/electronic/programmable electronic safety-related systemse. Standard, International Electrotechnical Commission, March 2010.

[10] Hazard and operability studies (HAZOP studies) - Application guide. Standard, International Electrotechnical Commission, March 2016.

[11] Product Quality Research Institute.

[12] Road vehicels - Safety of the intended functionality. Standard, International Organization for Standardization, Geneva, CH, January 2019.

[13] Road vehicles - Functional Safety. Standard, International Organization for Standardization, Geneva, CH, January 2019.

[14] McKinsey & Company. Automotive software and electronics 2030, July 2019. URL: https://www.mckinsey.com/~/media/mckinsey/industries/automotive%20and%20assembly/our%20insights/mapping%20the%20automotive%20software%20and%20electronics%20landscape%20through%202030/automotive-software-and-electronics-2030-final.pdf.

[15] McKinsey & Company. Rewiring car electronics and software architecture for the 'Roaring 2020s', January 2020. URL: https://www.mckinsey.com/industries/automotive-and-assembly/our-insights/rewiring-car-electronics-and-software-architecture-for-the-roaring-2020s.

[16] National Highway Traffic Safety Administration (NHTSA). NHTSA Part 573 Safety Recall Report 22V-232, April 2022. URL:https://static.nhtsa.gov/odi/rcl/2022/RCLRPT-22V232-3452.PDF.

[17] Petry, Erwin. Overview os IS 26262:2018, an Introduction into Automotive Functional Safety, November 2020.

[18] A. Saberi et al. *Integrated optimal design for hybrid electric vehicles*. PhD thesis, Eindhoven University of Technology, 2020.

[19] Siemens. The criticality of the automotive E/E architecture, April 2020. URL: https://www.plm.automation.siemens.com/media/global/en/Siemens-SW-criticality-of-the-automotive-EE-architecture-White%20Paper_tcm27-77625.pdf.

[20] Synopsys. Coverity scan - static analysis. https://scan.coverity.com/, 2022.

[21] TomTom. TomTom IndiGO - Digital Cockpit Platform. URL:https://www.tomtom.com/products/tomtom-indigo/.

[22] TomTom. Autostream Introduction, August 2022. URL:https://developer.tomtom.com/autostream-sdk/documentation/product-information/introduction.

[23] Vector. High-performance computing platforms in the automobile, February 2020. URL: https://cdn.vector.com/cms/content/products/vconnect/docs/2020-02_Automobil-Elektronik_Mastering-Automotive-OTA.pdf.

A Hazard Analysis and Risk Assesment

A.1 Hazard and Operability Study

Tables A.1 and A.2 list the hazards that may result from quantitative and real-time performance failures respectively.

A.2 Hazard Analysis and Risk Assesment

Tables A.3 to A.6 contain the tables for the HARA performed on the blind-spot highlighting system function 1.

Table A.1: HAZOP on quantitative failures

Failure Mode	**Possible Failure**	**Possible Hazard**
Unexpected	False positive: blind-spot detection state read as positive, when blind-spot detection state is negative/none.	System shows vehicle in blind-spot when there is none.
More (Magnitude)	If blind-spot area is boolean or numeric then increase of actual value can result in wrong blind-spot area. If blinds-pot detection state is boolean, then increase of actual value can result in it being stuck at 1.	System shows vehicle in blind-spot when there is none. System shows wrong blind-spot area as occupied.
Less (Magnitude)	If blind-spot area is boolean or numeric then decrease of actual value can result in wrong blind-spot area. If blinds-pot detection state is boolean, then decrease of actual value can result in it being stuck at 0.	System shows no vehicle in blind-spot area when there is one. System shows wrong blind-spot area as occupied.
Incorrect	False negative: read as negative/none, when blind-spot detection state is positive. Wrong blind-spot area.	System shows no vehicle in blind-spot area when there is one. System shows wrong blind-spot area as occupied.
Reversed	Reading reversed data values. This can result in: If blind-spot area is boolean, then wrong blind-spot area. If blind-spot detection state is reversed, then state becomes incorrect.	System shows vehicle in blind-spot when there is none. System shows no vehicle in blind-spot area when there is one. System shows wrong blind-spot area as occupied.
Stuck at value	The blind-spot detection state is always read as the same value, even if correct value changes.	System shows vehicle in blind-spot when there is none. System shows no vehicle in blind-spot area when there is one. System shows wrong blind-spot area as occupied.
Incomplete	Reading does not capture all necessary data. Results in blind-spot data being old or null.	System shows vehicle in blind-spot when there is none. System shows no vehicle in blind-spot area when there is one. System shows wrong blind-spot area as occupied.
No	No data is read.	System shows no vehicle in blind-spot area when there is one.

Table A.2: HAZOP on real-time performance failures

Failure Mode	**Possible Failure**	**Possible Hazard**
Redundant	The same blind-spot data values are read multiple times.	System shows vehicle in blind-spot when there is none. System shows no vehicle in blind-spot area when there is one. System shows wrong blind-spot area as occupied.
More (Rate)	The same blind-spot data values are read multiple times.	No hazard? Values still get updated on time.
Less (Rate)	Blind-spot data values are read too infrequently.	System shows vehicle in blind-spot when there is none, due to old value (very briefly, if at all). System shows no vehicle in blind-spot area when there is one, due to old value (very briefly, if at all).
Before	Blind-spot data is read before the corresponding update. Negative time offset.	System shows vehicle in blind-spot when there is none. System shows no vehicle in blind-spot area when there is one. System shows vehicle entered blind-spot too late. System shows vehicle left blind-spot too late.
After	Blind-spot data is read after the corresponding update. Positive time offset.	System shows vehicle in blind-spot when there is none. System shows no vehicle in blind-spot area when there is one. System shows vehicle entered blind-spot too late. System shows vehicle left blind-spot too late.
Early	Blind-spot data read too early (before update).	System shows vehicle in blind-spot when there is none, due to old value (very briefly, if at all). System shows no vehicle in blind-spot area when there is one, due to old value (very briefly, if at all). System shows vehicle entered blind-spot too late. System shows vehicle left blind-spot too late.
Late	Blind-spot data read too late (after update).	System shows vehicle entered blind-spot too late. System shows vehicle left blind-spot too late.
Erratic	Blind-spot data read at erratic intervals. Reading frequency not in sync with blind-spot data update rate.	System shows vehicle in blind-spot when there is none. System shows no vehicle in blind-spot area when there is one. System shows vehicle entered blind-spot too late. System shows vehicle left blind-spot too late.
Intermittent	Blind-spot data is not read some times. Results in potentially incorrect values.	System shows vehicle in blind-spot when there is none. System shows no vehicle in blind-spot area when there is one. System shows vehicle entered blind-spot too late. System shows vehicle left blind-spot too late.
Never	Data is never read.	System shows no vehicle in blind-spot area when there is one.

Table A.3: HARA

Situation analysis & Hazard identification				Risk assessment					Determination of Safety Goal	
Hazard	**Driver Type**	**Maneuver**	**Effect**	**S**	**Justification**	**E**	**Justification**	**C**	**Justification**	**ASIL**	**Safety Goal**
System shows vehicle in blind-spot when there is none.	Trustworthy driver: Cautious in case of bad visibility or bad road conditions. Paying attention to the traffic situation.	Driving and no lane change or turn	just confusion, no injuries	S0	Only driver annoyance.	E4	Exposure taken from frequency of driving situation. Steering is considered E4 in ISO26262:3, Table B3, Driving forward is more frequent	C1	Driver keeps on driving forward.	QM	None
		Changing lane at high speed						C1	The driver can break or reverts to the current lane.		
	Inexperienced driver: Careless in case of bad visibility or bad road conditions. Not paying attention to the traffic situation.	Driving and no lane change or turn	accident with potential injuries	S1	Potentially dangerous depending on driver tendency to break			C1	Driver keeps on driving forward.		
		Changing lane at high speed	accident with potential loss of life	S1	Potentially dangerous depending on driver tendency to break or undo turn			C1	The driver can break or reverts to the current lane.	QM	**Ensure blind-spot area is never highlighted if there is no vehicle in that area.**
System shows no vehicle in blind-	Trustworthy driver.	Driving and no lane change or turn	just confusion, no injuries	S0	Only driver annoyance.	E4	Steering is considered E4 in ISO26262:3,	C1	Driver keeps on driving forward.	QM	None

Table A.4: HARA (continued)

Situation analysis & Hazard identification			Risk assessment														Determination of Safety Goal
							Table B3, Driving forward is more frequent	C2	The driver can break or reverts to the current lane.								
								C1	Driver keeps on driving forward.								
spot area when there is one.		Changing lane at high speed	accident with minor injuries	S1	Potentially dangerous depending on driver tendency to break											B	**Ensure blind-spot area is always highlighted when there is a vehicle in that area.**
	Inexperienced driver.	Driving and no lane change or turn						C1	The driver can break or reverts to the current lane.								
		Changing lane at high speed	accident with potential loss of life	S3													
System shows wrong blind-spot area as occupied.	Trustworthy driver.	Driving and no lane change or turn	just confusion, no injuries	S0	Only driver annoyance.	E4	Steering is considered E4 in ISO26262:3, Table B3, Driving forward is more frequent	C1	Driver keeps on driving forward.							QM	None
		Changing lane at high speed						C2	The driver can break or reverts to the current lane.								
	Inexperienced driver.	Driving and no lane change or turn	accident with minor injuries	S1	Potentially dangerous depending on driver			C1	Driver keeps on driving forward.								

Table A.5: HARA (continued)

Situation analysis & Hazard identification				Risk assessment								Determination of Safety Goal
					tendency to break							
		Changing lane at high speed	accident with potential loss of life	S3	Potentially dangerous depending on driver tendency to break, undo turn, or simply drive into blind-spot vehicle			C1	The driver can break or reverts to the current lane.	B	**Ensure the correct blind-spot area is highlighted when the vehicle enters that area.**
System shows vehicle entered blind-spot too late.	Trustworthy driver.	Driving and no lane change or turn	just confusion, no injuries	S0	Only driver annoyance.	E4	Steering is considered E4 in ISO26262:3, Table B3, Driving forward is more frequent	C1	Driver keeps on driving forward.	QM	None
		Changing lane at high speed						C2	The driver can break or reverts to the current lane.		
	Inexperienced driver.	Driving and no lane change or turn	accident with minor injuries	S1	Potentially dangerous depending on driver tendency to break			C1	Driver keeps on driving forward.		
		Changing lane at high speed	accident with potential	S3	Potentially dangerous depending on driver			C1	The driver can break or reverts to the current lane.	B	**Ensure blind-spot area is highlighted with**

Table A.6: HARA (continued)

Situation analysis & Hazard identification				**Risk assessment**							**Determination of Safety Goal**
			loss of life		tendency to break, undo turn, or simply drive into blind-spot vehicle						**acceptable maximum latency after a vehicle is detected by ADAS in that area.**
System shows vehicle left blind-spot too late.	Trustworthy driver.	Driving and no lane change or turn	just confusion, no injuries	S0	Only driver annoyance.	E4	Steering is considered E4 in ISO26262:3, Table B3, Driving forward is more frequent	C1	Driver keeps on driving forward.	QM	None
		Changing lane at high speed						C2	The driver can break or reverts back to the current lane.		
	Inexperienced driver.	Driving and no lane change or turn	accident with minor injuries	S0	Only driver annoyance.			C1	Driver keeps on driving forward.		
		Changing lane at high speed	accident with potential loss of life	S0	Only driver annoyance.			C2	The driver can break or reverts to the current lane.		

B Failure Modes and Effects Analysis

This appendix documents the FMEA scoring used, as well as the system and component-level FMEA performed for the proof-of-concept system.

B.1 FMEA scoring

Tables B.1, B.2, and B.3 document the severity, occurrence and detection scoring used for the FMEA, respectively.

B.2 System and Component-Level FMEA

Tables B.4 to B.7 document the system and component level FMEA respectively. They use the following abbreviations:

- FM - Failure Mode
- S - Severity Score
- OC - Occurrence Total Score
- RTC - Requirements / Technology Clear
- Exp - Experience
- PC - Prevention Control
- DT - Detection Score
- RPN - Risk Priority Number

Table B.4 does not have a *Function* column, as there is the only one considered is *Intended Function 1: Render a blind-spot area highlighting whenever the input data reports the area as occupied*, determined in section 4.2.1.

Table B.1: FMEA Severity Scoring

Failure mode	Severity of Effect	Typical TomTom Examples	Rank
Failure to meet safety and/or regulatory requirements	Very high severity ranking when a potential failure mode affects safe vehicle operation and/or involves noncompliance with government regulation without warning	- Injury to customer due to extensive radiation - Injury to customer due to fire - Injury to customer due to touching hot display surface	10
	Very high severity ranking when a potential failure mode affects safe vehicle operation and/or involves noncompliance with government regulation with warning	Despite visual warning, injury to customer due to touching hot display surface	9
Loss or Degradation of Primary Function[1]	Vehicle/item inoperable (loss of primary function)	Empty vehicle battery	8
	Vehicle/item operable but at a reduced level of performance. Customer very dissatisfied.	Driver not provided with a timely rear view camera image when leaving the parking lot	7
Loss or Degradation of Secondary Function[2]	Vehicle/item operable but comfort/convenience item(s) inoperable. Customer dissatisfied	- TomTom device fails to switch on and start operating, due to e.g. high ambient temperature - TomTom device fails to stop operating and switch off - TomTom device not responding to external events	6
	Vehicle/item operable but comfort/convenience item(s) operable at a reduced level of performance. Customer somewhat dissatisfied.	- Driver not provided with vehicle guidance temporarily due to undetermined vehicle position when leaving a parking garage or ferry - Driver not provided with traffic aware vehicle guidance due to not receiving traffic information	5
Annoyance	Fit & finish squeak & rattle item does not conform. Defect noticed by most customers (greater than 75%)	Rattling noise due TomTom device SD card movements	4
	Fit & finish squeak & rattle item does not conform. Defect noticed by 50% of customers.	SD card insertion/removal requires relatively high force	3
	Fit & finish squeak & rattle item does not conform. Defect noticed by discriminating customers (less than 25%)	Screen animation speed not fully deterministic due to varying TomTom device work load.	2
Failure without effect	No discernible effect.	-	1

Table B.2: FMEA Occurrence Scoring

Ranking	0	1	2	3
Requirements / Technology	Fully clear	Some minor aspects unclear	Some major aspects unclear	Undefined / unclear
Experience	Successful experience on all aspects (requirements, technology, design, customer)	Successful experience on the key aspects (technology, design)	Successful experience on 1 key aspect (technology or design)	No successful experience on the key aspects (technology, design)
Prevention control	Reviewed with specific check in root cause	Reviewed	No review	No review, no document (design)
Total	**1 + SUM (Req./Technology + Experience + Prevention control)**			

Table B.3: FMEA Detection Scoring

Detection	**Likelihood of detection by design control**	**TomTom design controls**	**Rank**
Absolute uncertainty	No design control	No control in place	10
Low	Remote chance it will be detected	Code reviews with no specific check on root cause	8
Medium	Low chance it will be detected	-High level reviews on design and interface with no specific check on root cause - Manual tests like system test and functional test with no specific testcase on root cause	6
High	Moderately high chance it will be detected	- Automated tests like smoke test, software functional test and system test with no specific testcase on root cause - Manual tests like functional test and system test, with specific testcase on root cause	4
Very high	Very high chance it will be detected	- Automated tests like smoke test, functional test, and system test with specific testcase on root cause - High level reviews on design and interface with specific check on root cause	2
Almost certain	Almost certain chance it will be detected	Detection mechanism in design	1

Table B.4: System-level FMEA

FM	Potential Effect(s) of Failure	S	Potential Cause(s) Of Failure	Current design / process controls					RPN	Actions
				OC	RTC	Exp	PC	DT		
CAN data is lost by system.	Blind-spot highlighting incorrect.	9	System is unable to read the provided CAN data. Data is lost while communicated between software components. System software components lose the data. Buffer overflow.	4	1	1	2	10	360	This requires **design change**, as a checker has to be implemented for single-fault tolerance: Check if values are within expected boundary (i.e boolean either 0 or 1). Assume expected boundaries are provided by OEM CAN Spec, or something similar. Check if frame is received within the expected time-slot. Check if frame is of the expected size.
CAN data is altered erroneously.	Blind-spot highlighting incorrect.	9	Software components alter data erroneously.	4	1	1	2	10	360	
Render data is provided too late.	Blind-spot highlighting shown too late.	9	Software components execute too slowly. Inter-component communication is too slow.	4	1	1	2	10	360	

Table B.5: Component-level FMEA

Function	Potential Failure Mode	Potential Effect(s) of Failure	S	Potential Cause(s) Of Failure	Current design / process controls					RPN	Actions
					OC	RTC	Exp	PC	DT		
Data Input receives data from *Canplayer*: - Signals indicating presence of object in blind-spot. - Signals indicating blind-spot sensor fault state. Assumption: At this stage, there is no checking that involves data completeness or values. The task of the *Data Input* component is to simply read the data from the CAN socket, populate can_frame structs with the read data and provide them to *Data Processing.*	Capturing data into can_frame struct fails, due to read error.	Blind-spot highlighting not performed.	9	Read error caused by CAN socket failure.	4	1	1	2	10	360	The read error cannot be fixed as it pertains to bad input from external system. It needs to be signaled and system switched to safe state in case of repeated read errors. Tests shall be performed that the functions and checks generate the correct output and within FTTI.
	Check if there is an error when reading from CAN socket fails.		9	Internal software error of the checker.	4	1	1	2	10	360	
Data Processing receives data from *Data Input.* Assumption: *Data Processing* interprets the data field of the provided can_struct and performs checks for completeness and correctness.	can_frame ID is unknown.	Blind-spot highlighting may not be performed, or may be incorrect, if critical frame IDs are not recognized and the frames are ignored.	9	Internal software errors or external factors causing alteration of ID value.	4	1	1	2	10	360	*Data Processing* shall check if - ID is recognized. - Data size exceeds or is below DLC. - Data values are within expected bounds. - Data is received and processed within FTTI. If any of these checks fail than the errors will be logged, and the frame
	can_frame data field size does not match DLC.	Blind-spot highlighting may not be performed, or may be incorrect, if data is missing or exceeds length.	9	Internal software errors or external factors causing alteration of DLC value, as well as data value or size.	4	1	1	2	10	360	

Table B.6: Component-level FMEA (continued)

	Data arrives too late.	Blind-spot highlighting shown too late.	9	Data delayed by (unreliable) communication method. *Data Input* takes too long.	4	1	1	2	10	360	will not be used by the *Renderer*. If more than n (parameter) checks fail consecutively, the system shall switch to a safe state, until the errors no longer occur. Unreliable networks can be fixed by - Using the same execution environment for communication components, thus allowing for shared memory communication. - Using a more reliable communication protocol (ex. switch from UDP to TCP for proof-of-concept) Tests shall be performed that the functions and checks generate the correct output and within FTTI.
	Data does not arrive.	Blind-spot highlighting shown too late.	9	Data lost by (unreliable) communication method.	4	1	1	2	10	360	
	Buffer overflows.	Blind-spot highlighting incorrect.	9	*Data Input* is faster than *Data Processing* and buffer overflows or exceeds allocated memory.	4	1	1	2	10	360	
	Checking if ID is recognized fails	Blind-spot highlighting is not performed or is incorrect. User is not warned that there is an error.	9	Internal software error of the checker.	4	1	1	2	10	360	
	Checking if data is less than or exceeds DLC fails.	Blind-spot highlighting is not performed or is incorrect. User is not warned that there is an error.	9	Internal software error of the checker.	4	1	1	2	10	360	
	Checking if data is within bounds fails.	Blind-spot highlighting is not performed or is incorrect. User is not warned that there is an error.	9	Internal software error of the checker.	4	1	1	2	10	360	

Table B.7: Component-level FMEA (continued)

	Checking if data arrives within FTTI fails.	Blind-spot highlighting is not performed or is incorrect. User is not warned that there is an error.	9	Internal software error of the checker.	4	1	1	2	10	360	
Renderer receives data from *Data Processing.* Assumption: At this point data is complete and correct as per the checks done in *Data Processing*.	Data arrives too late.	Blind-spot highlighting shown too late.	9	Data delayed by (unreliable) communication method. *Data Processing* is too slow.	4	1	1	2	10	360	Unreliable networks can be fixed by - Using the same execution environment for communication components, thus allowing for shared memory communication. - Using a more reliable communication protocol (ex. switch from UDP to TCP for proof-of-concept) Tests shall be performed that the functions generate the correct output and within FTTI.
	Data does not arrive.	Blind-spot highlighting not performed.	9	Data is lost by (unreliable) communication method.	4	1	1	2	10	360	
	Rendering is done too late.	Blind-spot highlighting shown too late.	9	*Renderer* or another preceding data processing phase is too slow.	4	1	1	2	10	360	
	Buffer overflows (processing too slow)	Blind-spot highlighting incorrect.	9	*Renderer* is slower than *Data Processing* and buffer overflows or exceeds available memory.	4	1	1	2	10	360	

C Software Safety Phase - Additional Material

C.1 ISO26262 Part 6 Guidelines

The specified software implementation guidelines, according to ASIL level, are the following:

Table C.1: ISO26262:6 Software Implementation Guidelines

ASIL	Guidelines
A	- Enforcement of low complexity - Use of language subsets - Enforcement of strong typing - Use of naming conventions - One entry and one exit point in subprograms and functions - Initialization of variables - No multiple use of variable names - No unconditional jumps
B	Same methods as for ASIL A, plus: - Use of unambiguous graphical representation - Use of style guides - No dynamic objects or variables, or else online test during their creation - Restricted use of pointers - No implicit type conversions - No hidden data flow or control flow
C+D	Same guideline as for ASIL B, plus: - Use of defensive implementation techniques - Use of well-trusted design principles - Avoid global variables or else justify their usage - No recursions

The software unit and integration testing guidelines are the following:

C.2 Coverity Results

Table C.2: ISO26262:6 Software Testing Guidelines

ASIL	**Test Case Derivation**	**Testing**
A	Analysis of requirements.	Walk-through of the code, static code analysis, requirements-based testing, interface testing, resource usage evaluation.
B	Same methods as for ASIL A, plus equivalence class and boundary class analysis.	Same methods as for ASIL A, where walk-through is replaced by inspection.
C+D	Same methods as for ASIL B.	Same methods as for ASIL B, with the addition of fault injection testing, back-to-back testing, data flow and control flow verification.

D CAN Log Files

TODO explain

```
(1636965235.086000) slcan0 44a#00000000
(1636965235.087000) slcan0 44c#00000000

(1636965235.587000) slcan0 44a#00000100
(1636965235.588000) slcan0 44c#00000000

(1636965236.086000) slcan0 44a#00000000
(1636965236.087000) slcan0 44c#00000000

(1636965237.086000) slcan0 44a#00000000
(1636965237.087000) slcan0 44c#00000100

(1636965237.586000) slcan0 44a#00000000
(1636965237.587000) slcan0 44c#00000000

(1636965238.086000) slcan0 44a#00000000
(1636965238.087000) slcan0 44c#00000000

(1636965238.587000) slcan0 44a#00000100
(1636965238.588000) slcan0 44c#00000100

(1636965239.086000) slcan0 44a#00000000
(1636965239.087000) slcan0 44c#00000000
```

Listing D.1: CAN Log file - Error Free Scenario

```
(1636965234.589000) slcan0 44a#00000000
(1636965234.590000) slcan0 44c#00000000

(1636965235.086000) slcan0 44a#00000000
(1636965235.087000) slcan0 44c#00000000

(1636965237.588000) slcan0 305#0000

(1636965235.186000) slcan0 44a#1111111111111111
(1636965235.187000) slcan0 44c#1111111111111111

(1636965235.186000) slcan0 44a#00
(1636965235.187000) slcan0 44c#00

(1636965235.587000) slcan0 44a#00000100
(1636965235.588000) slcan0 44c#00000000

(1636965236.086000) slcan0 44a#00000000
```

```
(1636965236.087000) slcan0 44c#00000000

(1636965236.587000) slcan0 44a#00000000
(1636965236.588000) slcan0 44c#00000000

(1636965237.086000) slcan0 44a#00000000
(1636965237.087000) slcan0 44c#00000100

(1636965237.586000) slcan0 44a#00000000
(1636965237.587000) slcan0 44c#00000000

(1636965238.086000) slcan0 44a#00000000
(1636965238.087000) slcan0 44c#00000000

(1636965238.587000) slcan0 44a#00000100
(1636965238.588000) slcan0 44c#00000100

(1636965239.086000) slcan0 44a#00000000
(1636965239.087000) slcan0 44c#00000000
```

Listing D.2: CAN Log file - Scenario with errors

PO Box 513
5600 MB Eindhoven
The Netherlands
tue.nl

EngD SOFTWARE TECHNOLOGY