VDA QMC

Verband der Automobilindustrie
Qualitäts-Management-Center

Quality Management in the Automotive Industry

Technical risk analysis of software-based functionalities

1st edition, November 2025
Online download document

Quality Management
in the Automotive Industry

Technical risk analysis of software-based functionalities

1st edition, November 2025
Online download document
Verband der Automobilindustrie e. V. (VDA)

ISSN 0943-9412

Non-binding VDA recommendation

The German Association of the Automotive Industry (VDA) recommends that its members apply the following VDA volume when introducing and maintaining QM systems.

Exclusion of liability

VDA publications are recommendations available for general use. Anyone who implements them is responsible for ensuring that they are used correctly in each case.

This VDA publication is based on state-of-the-art technical procedures, current at the time of issue. Implementation of VDA recommendations does not absolve anyone of liability for their actions. In this respect, everyone acts at their own risk.

The VDA and those involved in VDA recommendations shall bear no liability.

If during the use of VDA recommendations, errors or the possibility of misinterpretation are found, it is requested that the VDA be notified immediately so that any possible faults can be corrected.

Copyright

Translations

The German document is the original. In case of questions of interpretation in other language versions, reference should be made to the German version as the original. This publication will also be issued in other languages. The current status must be requested from VDA QMC.

Preface

This volume is the result of a dynamic work process – produced within a short period of time, supported by strong technical focus and precise methodology. The speed with which the requirements, experience and solutions were compiled reflects the urgent need to address software-related risks at this time.

Rather than simply adding to existing methods, we intentionally opted for a new approach: structured, hands-on and compatible with established quality processes. The contents of this volume offer guidance in a complex subject area – not as a rigid framework but as a methodological compass for risk analysis of software-based functionalities in the automotive context.

Table of Contents

1 Introduction, motivation and objectives

With the increasing digitalization and growing capabilities of road vehicles, there has been a significant rise in the use of software-based components. Software-based functionalities are now an integral part of safety-related and quality-critical vehicle systems. Yet risk analysis is still typically performed using methods primarily designed for hardware.

In particular, the latest version of the FMEA, an established standard in quality management, suffers from methodological limitations when it comes to the evaluation of software-induced risks. The specifications of the IATF place additional restrictions on methodological flexibility. Meanwhile, it has been shown in practice that existing risk analysis processes are often too time-consuming, too labor-intensive and not effective enough. For example, risks may not be adequately detected, e.g. due to failure to identify operating conditions or interfaces.

The main purpose of a technical risk analysis is to identify and prevent potential problems early on, before they can produce negative consequences – e.g. errors during integration in a later project phase, project delays or quality defects in operation. So effective risk analysis is crucial to the stability and efficiency of development processes.

This volume addresses the need for a more targeted, more efficient and more context-appropriate approach to the analysis of software risks. The goal is to reduce complexity and offer practical guidance: Which risk analysis methods make sense in which development phase? How can we identify and assess software-related risks early on?

A special emphasis is placed on the phase-specific application of suitable methods in order to promote a preventive approach starting right from the concept phase. Risk analysis should be seen as a helpful tool – not an additional burden but a structured aid for ensuring functional quality.

2 Benefits, scope of application and target group

This volume describes a systematic methodology for identifying, analyzing, evaluating and handling technical risks of software-based functionalities (cf. Chapter 3 Terminology.) The focus is on the complete product life cycle. The methods presented here are intended as a guide for improving the quality, reliability and safety of software-based functionalities in vehicles of the automotive industry and its ecosystem.

2.1 Benefits

2.1.1 Objectives and advantages of risk analyses

The application of the methodology described in this volume is intended to help proactively identify and handle technical risks. This should help prevent negative consequences and get products ready for production. The primary objectives are to prevent the following:

- errors during integration in a later project phase and project delays
- customer complaints, product recalls and costly error corrections
- safety incidents or security breaches
- loss of the customer trust

2.1.2 Technical risks considered

The scope of this analysis covers a wide range of software-related technical risks potentially affecting the vehicle, its driver or its environment. Among other things, this includes:

- **Functional risks:** The software does not perform its intended functionality correctly, resulting in such issues as malfunctions, reduced performance, restricted user experience and damage to image.
- **Safety & security risks:** Software vulnerabilities or failures which could result in an unsafe state (safety) or be exploited by bad actors (security.)
- **Reliability & availability risks:** The software temporarily fails or is not available during operation.

- **Performance risks:** This software does not meet the timing, memory or computing power requirements, which impacted the response capability and stability of the system.
- **Integration and communication risks:** Errors in the interfaces between different software components, control units, actuators, sensors and/or with other systems as well as risks due to faulty communication between interconnected systems.

2.2 Scope of application

This publication is intended to supplement industry standards and is not a replacement for them. It provides an application-oriented methodology for selecting and applying suitable risk analysis methods for specific software-based functionalities. It refers to standards such as the AIAG & VDA FMEA Handbook, VDA Maturity Level Assurance, ISO 26262 (Functional Safety,) ISO/SAE 21434 (Cybersecurity) and Automotive SPICE®.

2.2.1 Objectives and advantages of risk analyses

The analysis centers on software-based functionality. It assesses the software's intended functionality (target behavior) and potential deviations from that behavior (malfunctions.) This goes for all the software artifacts and components that go into its implementation, including:

- Apps on the control unit
- Middleware
- Operating systems
- Firmware
- Drivers
- Debug-software boot loader (e.g. flash-BL)
- Configuration data
- Message catalog
- Base application
- AI models

- Remote application
  (backend, phone APP)
- Cloud applications
- Legacy software
- Free and Open-Source Software (FOSS)
- Commercial Off-The-Shelf Software (COTS)
- Modified Off-The-Shelf Software (MOTS)

2.2.2 Special focus: connected-car technology

Automotive vehicles are highly interconnected systems operating in a complex ecosystem (see Figure 2-1).

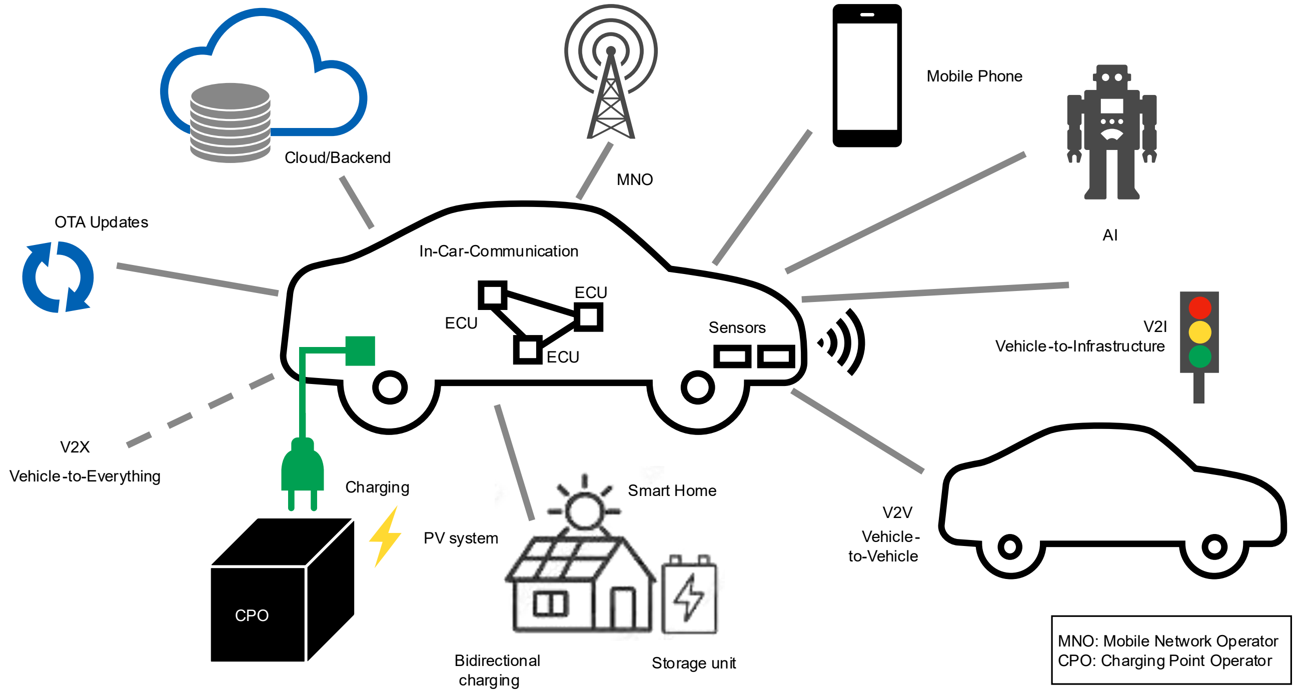


Figure 2-1: Vehicle ecosystem

The resulting risks are a focus of this volume. This refers, in particular, to:

- **In-Car-Communication:** Risks related to communication by individual control units in the vehicle
- **V2X (Vehicle-to-Everything) Communication:** Risks related to data integrity, authenticity and availability of external sources.
- **Cloud and backend interfaces:** Risks involved in interaction with servers, data protection and the reliability of connected services.

An additional level of complexity arises from the globally spread supply chain. Development partners work in parallel on different integration levels of the software. Consistent risk management therefore requires that the risk analysis findings be linked up in a synchronized and traceable manner across system limits and company levels.

Out of scope: This volume focuses on the risks that originate in the software or in the system design of software-based functionalities. The following risks are explicitly excluded from its scope (out of scope):

- Risks that can be attributed exclusively to mechanical and electronic errors.
- Risks involved in production processes.
- Purely commercial project risks (e.g. exceeding budgets, market acceptance.)
- Organizational risks (e.g. project management, resource planning,) to the extent that they do not occur directly in a technical product.

2.3 Target group

This volume is intended for all skilled and managerial staff involved in the specification, development, integration and safeguarding of software in the automotive branch. This includes, in particular:

- software developers and architects
- system and safety engineers
- quality assurance and testing managers
- project and product managers
- technical purchasers and supplier quality engineers

3 Terms and definitions

3.1 Function architecture

Function architecture refers to the structured breakdown of superordinate system and software functionalities into subsystems and components. Both the static behavior (e.g. states and interfaces) and the dynamic behavior (e.g. processes and interactions) are defined and modelled.

3.2 Hardware (HW)

Hardware includes all physical components required for the function of a system – e.g. mechanical, electronic, hydraulic and sensor elements.

3.3 Inductive vs. deductive methods

Inductive methods analyze possible causes and their effects, while deductive methods start with a failure pattern and systematically trace its causes. Both approaches are part of the safety analyses specified in ISO 26262.[1]

3.4 Risk

Risk refers to the combination of the probability of occurrence and the consequences of a certain undesirable future event.[2]

3.5 Software (SW)

Software includes executable programs, process as well as associated documentation and data pertaining to the operation of a computerized system.[3]

[1] ISO 26262-9:2018. *Road vehicles — Functional safety — Part 9: Automotive safety integrity level (ASIL)-oriented and safety-oriented analyses*. International Organization for Standardization, Edition 2, 2018. Section 8

[2] VDA Online Glossary, based on: ISO/IEC/IEEE 24765:2017, *Systems and software engineering – Vocabulary*, 2nd Edition, September 2017.

[3] VDA Online Glossary, based on: ISO/IEC/IEEE 24765:2017, *Systems and software engineering – Vocabulary*, 2nd Edition, September 2017.

3.6 Software-based functionality

A software-based functionality refers to a specific capability or behavior of a system that is performed or assisted by software. This function can occur either “in-vehicle” or in combination with the interlinked systems (see Figure 2-1).
The job of the software-based functionality is to provide the designated functions and processes by means of the software in order to meet defined requirements and objectives within its system context.
The software-based functionality is described through the definition of input, processing and output in compliance with non-functional requirements within its system context.

3.7 V2X (Vehicle to Everything)

V2X describes the interlinking of a vehicle with external partners. This includes specific types such as

- V2I (*Vehicle to Infrastructure*) for communication with transport infrastructure and
- V2V (*Vehicle to Vehicle*) for direct interlinking between vehicles.

4 Basic principles of risk analysis

The basic principles of risk analysis are central guidelines that ensure that risks in technical systems, particularly in functional safety (e.g. according to ISO 26262, ISO 31000, IEC 61508) and cybersecurity (e.g. according to ISO/SAE 21434, ISO/IEC 27001), are systematically and comprehensibly identified, evaluated and appropriately handled. The procedure and prerequisites for performing risk analyses on software/software-based functionality are the same as for risk analyses of hardware. It is not possible to identify all risks, but the most critical issues must be covered.
When it comes to the systematic identification of risks, there are various prerequisites to consider before performing a risk analysis, as described in the following section. The section after that deals with the systematic approach to risk identification.

4.1 Prerequisites for performing a risk analysis

The prerequisites of a risk analysis include:

- the description of the software functionalities resulting, for example, from a system analysis.
- the description of non-functional requirements, e.g. safety relevance of the software functionality, performance, reliability, tolerances, signal quality.
- the allocation of software-functionalities to elements of a system/software architecture.

4.2 Systematic approach to the identification of risks

The systematic approach comprises the following four steps:

1. Systematic procedure
    - No ad-hoc analyses – instead, a comprehensible process
    - Independent of the architecture level

2. Considering the system context
    - Define contents

- Consider the system context (risk analysis factors in system boundaries, environmental conditions, user behavior and interfaces.)
- Consider relevant risks from upstream risk analyses (e.g. HARA, TARA, D-FMEA system)

3. Identifying and assessing potential risks
    - Select the appropriate method of risk analysis (see Chapter 5)

4. Defining and implementing appropriate measures
    - Examples of appropriate measures (see ISO 21434:2021 RQ-15-17) include:
        - Risk avoidance
          (e.g. by revoking functionalities)
        - Risk reduction
          (e.g. implementation of technical measures for risk reduction)
        - Transferring the risk
          (e.g. with insurance providers)
        - Accepting the risk

If a method fails to meet one or more of the requirements from the aforementioned four steps, a defined supplementary system of compensation must be used.

4.3 System division and interaction at the interfaces

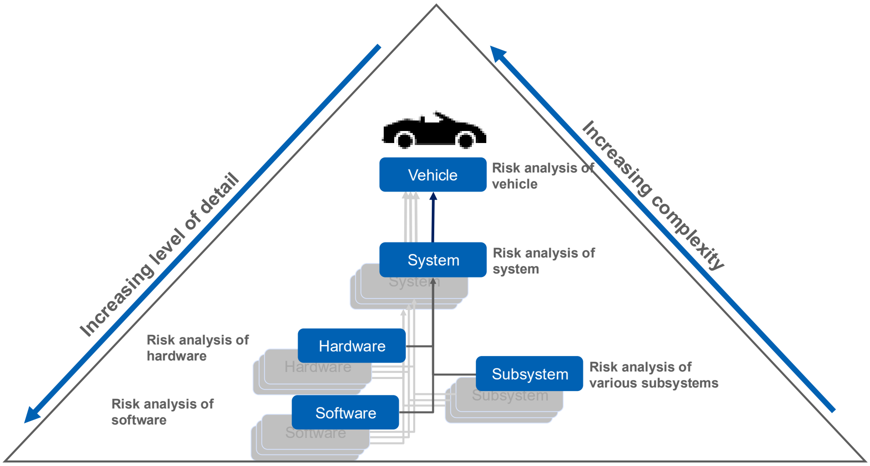

Increasing level of detail
Increasing complexity
Vehicle
Risk analysis of vehicle
System
Risk analysis of system
Hardware
Risk analysis of hardware
Subsystem
Risk analysis of various subsystems
Software
Risk analysis of software

Figure 4-1: Schematic diagram of system and sub-system with interfaces within a vehicle in the automotive industry

The figure shows how an automotive vehicle is split up into its systems and sub-systems. Between the systems and sub-systems, there are interfaces defined for mutual interaction. Risk analyses are performed on all levels.

The software is a component of a (sub-)system which has an interface with the hardware, the risk analysis of which is discussed in this publication.

5 Guide for method selection

5.1 Instructions for selecting risk analysis methods

This chapter provides a structured guide for selecting appropriate risk analysis methods for software functions in software-based vehicle systems. The targeted selection of suitable methods if essential in order to for the analysis to be efficient, relevant and comprehensible. It allows for a targeted identification, evaluation and handling of risks and helps us find efficient analysis paths. A methodologically sound approach ensures that the risk analysis process is systematic and application-oriented in accordance with the specific requirements of software-based vehicle functions.

5.1.1 Phase definition and process over time

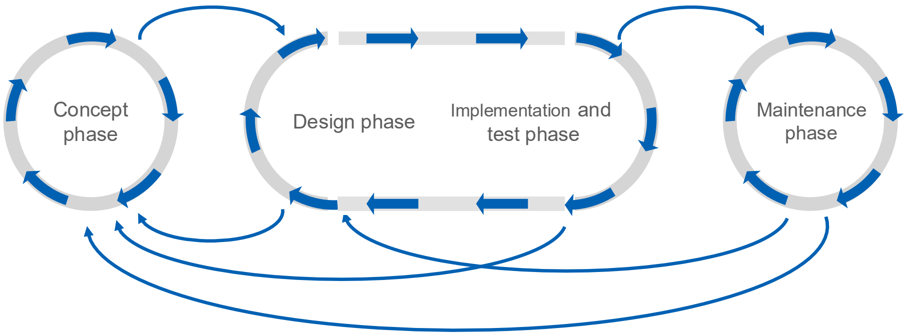


Figure 5-1: Phase model

The figure shows four main phases: the concept, design, implementation and test phases as well as the maintenance phase. In each of these phases, there are specific risk identification activities to perform.

In the concept phase, an initial risk identification step is performed based on basic function concepts. This is followed by the design phase, in which a detailed analysis is performed based on the system architecture.

The implementation phase comprises the analysis and evaluation of risks from the specific implementation. We will also be looking at risks involved in the integration of multiple subsystems.

In the maintenance phase, the focus of the risk analysis is on ensuring that the integrity of the systems in operation is not compromised.

A central element within this process is iteration: Findings from later phases continuously flow back into earlier development stages, allowing the risk analysis to be continuously improved and adapted. This iterative procedure supports robust and adaptive product development throughout the entire life cycle.

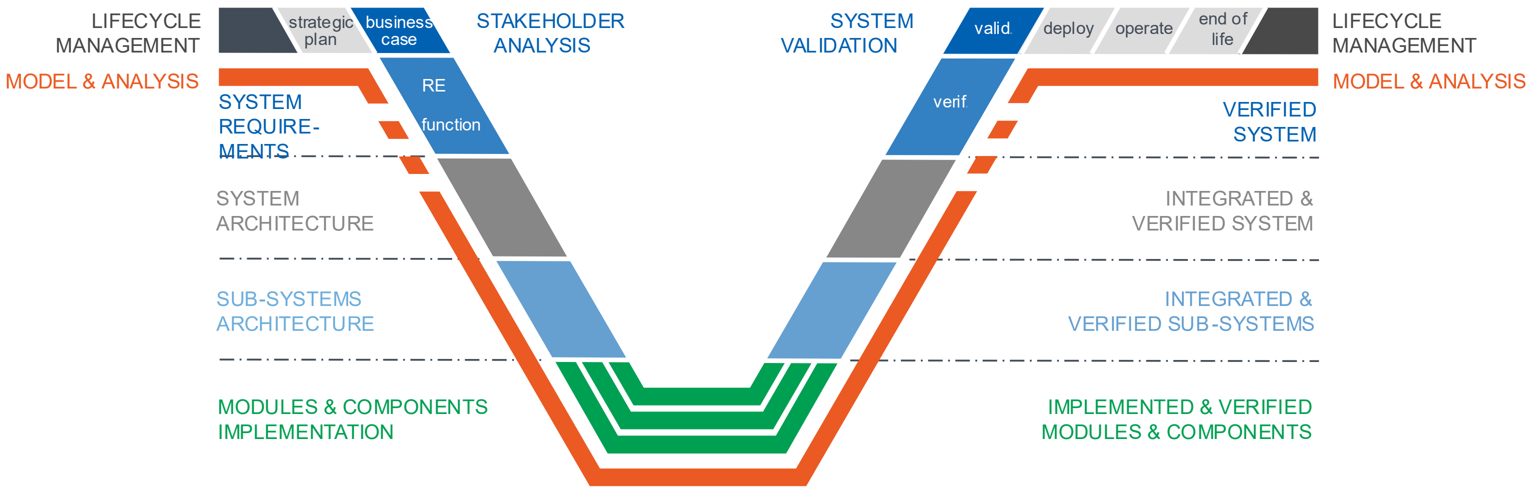


Figure 5-2: Overview of system integration chain

5.1.2 Classification and acceptance criteria for risks

The selection and application of risk analysis methods is largely determined by a classification based on the analysis the project scope and both the functional and technical requirements. It promotes a targeted concentration on safety-critical and functionally critical system areas and optimizes resource utilization in the analysis process.

Project-specific acceptance criteria complement defined thresholds by establishing what risk level (e.g. severity and probability of occurrence) is considered acceptable in the given application context. Risks that exceed these thresholds require corresponding risk management actions. The criteria are defined in close collaboration with the interested parties and form the foundation for consistent and comprehensive risk decisions.

5.1.3 Criteria for the definition, delineation and prioritization of the scope of analysis in the product life cycle

The selection of the appropriate risk analysis methods largely depends on the current stage of product development as well as von the specific characteristics of the software under analysis. Accordingly, there are different criteria to consider along the product life cycle in order to define, delineate and prioritize the scope of analysis.

5.1.3.1 Concept phase

In the concept phase, the focus is on system-wide observation as well as the identification of potential risk areas.

- **System architecture overview:** Initial rough model of the system architecture, including main components and how they cooperate (HW, SW, communication) in order to identify potential weak points or critical dependencies.
    - System context (boundary analysis): Delineation of the system to be analyzed, including its interfaces to software, hardware, environment and users
- **Analysis of requirements:** Detection and analysis of functional and non-functional requirements, including input/output specifications of the software functionality, e.g.:
    - regulatory requirements and standards
    - customer requirements

    - interface requirements
    - integration in the supply chain
    - safety
    - (cyber)security
    - data protection requirements
    - patent rights and licenses

- **Innovation assessment:** Assessment to determine whether something is a new development, a significant change or an established design.

5.1.3.2 Design phase

This phase focuses on the robust design of architectures, interfaces and dependencies as well as safety and cyber security aspects, including in systems with machine learning.

- **Detailed architecture and module assessment:** In-depth analysis of the individual software components and their functionality, including communication, state management and error handling.

- **Interface analysis:** Analyzing software-end dependencies and their interactions with other system components (SW ↔ SW / SW ↔ HW).

- **Dependencies on third-party systems and third-party software:** Analyzing the risks stemming from integrated software libraries, middleware or external services that are subject safety and stability requirements.

- **Safety mechanisms and safeguarding strategies:** Evaluation of implemented safety functions such as watchdogs, redundancies, failure detection and management as well as their effectiveness in the risk context.

- **Security**: Evaluation and handling of cybersecurity risks.

5.1.3.3 Implementation and test phase

In this phase, the focus is on the implementation and testing of requirements.

- **Implementation, integration, verification and validation of software and hardware requirement:** Ensure the coherent implementation of requirements both on the software level and on the hardware level, e.g. safeguarding control units and bus communication.

- **Validation of risk prevention measures:** Perform tests and reviews to check if planned and implemented actions adequately hedge the risks.

5.1.3.4 Maintenance phase

In this phase, the focus is on the monitoring and management of risks in actual operation.

- Consider the actual environmental conditions under which the software function is operated
    - Software over the air update (OTA)
    - V2X, V2I, V2V
- Prevention of problems that can occur due to the product being modified during ongoing operation
- Review of the technical risk analysis in conjunction with problems in the field

- Lessons learned from field observation as input for new risk analyses

5.1.4 Overview of method selection

The following tables show you see which risk analysis methods are particularly relevant for each phase of the product life cycle. They can be used to help with decision-making, and they emphasize that a combination of methods is often the best approach.

Table 5-1: Risk analysis methods according to phases

Risk analysis methods	Concept phase	Design phase	Implementation and test phase	Maintenance phase	Chapter
ATAM – Architecture Tradeoff Analysis Method	++++	+ ++			5.2.1
CCA – Common Cause Analysis		+	+		5.2.2
CPA – Criticality and Priority Assessment	++++	+++	++	+	5.2.3
DRBFM – Design Review Based on Failure Mode		++++	++	+++	5.2.4
ETA – Event Tree Analysis		++		+++	5.2.5
FMEA – Failure Mode and Effects Analysis	+++	+++	++	+	5.2.6
FMEA-MSR – FMEA-Monitoring and System Reaction	+	+++	+	+	5.2.7
FMEDA – Failure Modes, Effects and Diagnostic Analysis		+	+	+	5.2.8
FTA – Fault Tree Analysis	+	+++	++	+++	5.2.9
HARA – Hazard Analysis and Risk Assessment	++++	+	+		5.2.10
HAZOP – Hazard and Operability Study	+	++++	+++		5.2.11
SOTIF – Safety Of The Intended Functionality*)	++++	+++	+	+	5.2.12
SWIFT – Structured What-If Technique	++	++++	+++		5.2.13
STPA – System-Theoretic Process Analysis	++++	++			5.2.14
TARA – Threat Analysis and Risk Assessment	++++	+++	++	+	5.2.15
*) Limited to risk analysis methods from the standard					

● ++++ (Very high):	In this phase, the method is extremely relevant and highly advantageous. It is often fundamental or a best practice.
● +++ (High):	In this phase, the method is very relevant and provides significant results. Its use is strongly recommended.
● ++ (Medium):	In this phase, the method is moderately relevant and can provide useful insights. It might be appropriate to use it, depending on the situation.
● + (Low):	In this phase, the method is only somewhat relevant and not very advantageous. Its use should be checked carefully.
● (Empty field):	In this phase, the method is not at all or just barely relevant.

5.2 Standardized method representation

5.2.1 ATAM (Architecture Tradeoff Analysis Method)

5.2.1.1 Purpose of the method

The systematic evaluation of the effects of architecture decisions on the quality attributes of a software system is performed for the purpose of identifying risks, sensitivity points and possible trade-offs between competing quality objectives. It helps us make well-founded decisions in the architecture process and creates transparency with regard to relevant influence factors.

5.2.1.2 Basic procedure

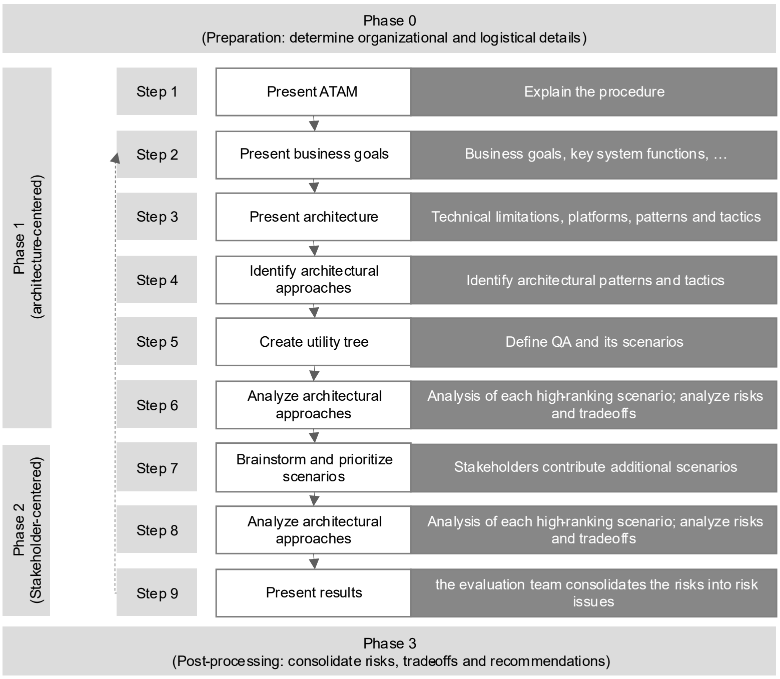

Phase 0
(Preparation: determine organizational and logistical details)
Phase 1
(architecture-centered)
Phase 2
(Stakeholder-centered)
Step 1
Present ATAM
Explain the procedure
Step 2
Present business goals
Business goals, key system functions, …
Step 3
Present architecture
Technical limitations, platforms, patterns and tactics
Step 4
Identify architectural approaches
Identify architectural patterns and tactics
Step 5
Create utility tree
Define QA and its scenarios
Step 6
Analyze architectural approaches
Analysis of each high-ranking scenario; analyze risks and tradeoffs
Step 7
Brainstorm and prioritize scenarios
Stakeholders contribute additional scenarios
Step 8
Analyze architectural approaches
Analysis of each high-ranking scenario; analyze risks and tradeoffs
Step 9
Present results
the evaluation team consolidates the risks into risk issues
Phase 3
(Post-processing: consolidate risks, tradeoffs and recommendations)

Figure 5-3: Phases of the ATAM method

The ATAM involves three teams: The evaluation team that moderates the method, holds workshops and evaluates the architecture; the core team, consisting of architects, project managers and other decision makers who provide technical and commercial insights; as well as the architecture stake-

holders, who could include, for example, developers, testers and representatives from Management and who contribute different perspectives and requirements.
The ATAM comprises four phases, as shown in Figure 5-3. During preparation (Phase 0,) organizational framework conditions are clarified, participants are designated, and the architectural documents are reviewed. Phases 1 and 2 make up the core of the method. In Phase 1, the the evaluation team meets with the core team in order to present the method, commercial objectives and architecture, identify central architectural approaches and create what is known as a "quality attribute utility tree" (see case example in Annex, Chapter 7.1.1) which structures the quality requirements in prioritized scenarios. They then analyze the architecture in detail based on these scenarios, identifying risks, compromises and strengths along the way. In Phase 2, additional stakeholders are added to the group of participants. These stakeholders devise and prioritize quality scenarios and work with the evaluation team to evaluate the architecture in-depth. This procedure ensures a comprehensive, scenario-based evaluation of the software architecture. Finally, in Phase 3, the team produces a final report that summarizes the risks, decisions, compromises and recommendations.

5.2.1.3 Instructions/special considerations/differences in use in the software context

ATAM is suitable for complex systems with stringent quality requirements. The focus is on the analysis of quality attributes and its dependencies using scenario-based evaluations.

5.2.1.4 Prerequisites (input)

Before the technical risk analysis can be performed, there must be a documented software architecture, clearly defined architectural decisions as well as active involvement of the stakeholders.

5.2.1.5 Depth of application and termination criteria

The analysis is scenario-based and adapted according to the complexity of the software architecture. It is considered completed once the prioritized quality attributes have been evaluated, central risks and trade-offs have been identified, and a consensus has been reached between the stakeholders.

5.2.1.6 Requirements for work, team and tools

The ATAM method is very laborious, involving multiple phases with workshops, scenario development and architecture evaluations.

The analysis requires an interdisciplinary team with specific knowledge:

- Evaluation team: Methodological expertise in architecture, workshop moderation and documentation.
- Core team: Provides well-founded information on software architecture, project management as well as technical and commercial connections and has experience evaluating quality attributes.
- Architecture stakeholders: Provide different perspectives and specialized knowledge, e.g. technical expertise pertaining to implementation, know-how pertaining to ensuring software quality, insights into operation and maintenance as well as understanding of commercial and organizational requirements.

Auxiliary tools for documentation and visualization can be helpful but are not required.

5.2.1.7 Benefits in the software context (advantages)

ATAM makes it possible to identify architectural risks early on, helps with transparency of quality requirements and promotes interdisciplinary communication. The method can be used preventatively and provides a comprehensible basis for decision-making.

5.2.1.8 Limitations of the method in the software context (disadvantages)

No quantitative evaluation or action planning. Results are qualitative and depend on the expertise of the participants. The method is time-consuming and requires a lot of stakeholder involvement.

5.2.1.9 Result (output)

Documented risks, sensitivity points, identified trade-offs as well as evaluated quality attribute scenarios are key results of the analysis. They make up the basis for the subsequent architectural refinement and for further technical evaluations.

5.2.1.10 References/Additional reading

Rick Kazman, Mark Klein, Paul Clements (2000). ATAM: Method for Architecture Evaluation. SEI Technical Report CMU/SEI -2000-TR-004.

Bass, L., Clements, P., & Kazman, R. (2022). *Software Architecture in Practice* (4th Edition). Addison-Wesley.

5.2.2 CCA (Common Cause analysis)

5.2.2.1 Purpose of the method

This method is used to identify and evaluate risks that have common causes and can impact multiple otherwise independent software functions at once. It is used in functional safety, in cybersecurity and when evaluating the reliability of the vehicle.

5.2.2.2 Basic procedure

The basic procedure in Common Cause Analysis (CCA) in the software context includes multiple steps for revealing systemic weak points.

1. Describing the software functionalities and systems
First of all, the software functionalities and systems to be analyzed are described in detail. The objective is to understand their dependencies, interfaces and interaction. This includes mapping all the system elements and how they are linked in a structured diagram.

2. Systematic cause analysis
The next step is to look specifically for common causes that can influence multiple actually independent software functionalities at once. This analysis can be done on different architecture levels:

- E/E architecture
- Software architecture
- Development process
- Tool level

3. Going over specific failure scenarios
Typical scenarios are simulated in order to identify potential common causes. Examples:

- What happens if the main power grid fails?
- What happens if a communication bus is malfunctioning?

- What happens if a shared sensor is providing faulty data?

4. Assessing the effects
The potential consequences of a mutual failure are assessed according to various aspects:
- Functional safety (ASIL)
- Functionality
- Performance
- Comfort
- Cybersecurity

5. Analyzing available protective mechanisms
Available redundancies, diversities, monitoring and sorting mechanisms are investigated. The objective is to assess their effectiveness against the identified common causes.

6. Action planning based on results
If the safeguarding is insufficient, specific recommendations are devised for mitigating or eliminating the risks. Possible actions can include:
- Design changes
- Implementing additional diversity
- Improving test strategies
- Adjusting development processes
- Stricter qualification of COTS software.

5.2.2.3 Instructions/special considerations/differences in use in the software context

Software functionalities often rely on shared resources (e.g. operating systems, libraries, hardware components,) development processes or communication paths. This can lead to systemic errors that could be overlooked when examining individual functionalities in isolation. Unlike hardware errors, which are often random, software errors with common causes can lead to wide-reaching and simultaneously occurring failures. The method focuses on logical dependencies and systemic weak points that go beyond purely physical failures. The scenario-based approach is essential for identifying the complex interactions and potential common failure modes in software systems.

5.2.2.4 Prerequisites (input)

E/E architecture of the vehicle: Detailed plans of the electronic/electrical architecture, linking (CAN, FlexRay, Ethernet,) control units (ECUs,) sensors, actuators and their interactions.

Software architecture and design: Precise descriptions of the software module, components, interfaces, data flows and dependencies within and between the ECUs.

Functional specifications: Precise definitions of the individual software functions (e.g. brake control, assisted steering, infotainment, battery management) and their performance characteristics as well as the safety goals.

Risk analyses (available): Result from previously performed analyses such as HARA, FMEA, FTA, TARA.

Safety and cybersecurity concepts: Information on implemented safety mechanisms (e.g. redundancies, diversities, monitoring functions, fail-operation strategies) and cybersecurity measures (e.g. encryption, authentication, secure boot.)

Environmental conditions and operating modes: Knowledge of the various operating modes of the vehicle (e.g. ignition on/off, driving, parking,) environmental influences (temperature, vibration, EMV) and their effects on the software.

Standard components and libraries: Details on the use of COTS software, open source libraries, AUTOSAR basic software or recurring software modules that are used in different functions.

Hardware details: Information on the underlying hardware (processors, memory, ASICs), since hardware error can be common causes for software errors.

5.2.2.5 Depth of application and termination criteria

Vehicle's overall system level: Identifying critical software functions or systems (e.g. drive, steering, brakes, ADAS,) that could fail due to common causes.

E/E architecture level: Analyzing common hardware resources (e.g. power supply, communication buses such as CAN/Ethernet, common sensors, central computing units,) which could impact multiple software functions if they fail.

<u>Software architecture level:</u> Deeper analysis of shared software modules, libraries, operating system services, middleware, data structures or algorithms that are used by multiple functions.

<u>Development process and tool level:</u> Investigation of potential common causes that originate from the development process itself (e.g. errors in requirement management, in the compilers, modelling tools or code generators used.)

5.2.2.6 Requirements for work, team and tools

A thorough CCA can take up a lot of time and resources, especially with large and complex vehicle systems. Identifying all potential common causes in highly interlinked software systems is a big undertaking and requires detailed knowledge of the systems. An interdisciplinary team is essential. It should include deep knowledge of the entire E/E architecture, software architecture, the system design, the utilized technologies and the development process. This includes software architects, developers, safety and security experts as well as domain experts. There are fewer specialized tools for CCA in the software area than in the hardware area. It is often necessary to resort to generic modeling or analysis tools.

<u>Tools:</u> There are fewer specialized tools for CCA in the software area than in the hardware area. It is often necessary to resort to generic modeling or analysis tools.

5.2.2.7 Benefits in the software context (advantages)

<u>Identification of hidden risks:</u> Reveals weak points that might be missed when looking at individual functions in isolation or using other analysis methods (which often focus on individual errors.)

<u>Increased vehicle safety:</u> Leads to a better understanding of system resilience to systemic failures which could lead to dangerous situations.

<u>Optimization of redundancies and diversities:</u> Helps us determine whether implemented redundancies or diversities are in fact effective or whether they could be bypassed by a common cause.

Improved E/E and software architecture: Provides valuable insights for designing a more robust, safe and reliable vehicle architecture.

More efficient testing and validation strategies: Guides the development of test cases targeting common causes which would otherwise be difficult to test (e.g. injecting errors into common resources.)

Long-term cost reduction: Identifying and remedying common-cause errors early prevents expensive recalls, field service or warranty cases.

Complying with standards: Supports compliance with applicable standards such as ISO 26262 (Functional Safety,) which explicitly address Common Cause Failures, and ISO 21434 (Cybersecurity,) where common weak points can be exploited.

5.2.2.8 Limitations of the method in the software context (disadvantages)

High complexity: Modern vehicle software is complex and highly interlinked. Identifying all potential common causes can take a lot of time and effort.

Degree of abstraction: Choosing the right degree of abstraction is difficult. Too detailed, and it becomes hard to navigate. Too broad, and you might overlook critical common causes.

Subjectivity: The identification of potential common causes can involve a certain degree of subjectivity if not all dependencies are explicitly documented.

Dynamic behavior: Software behavior is often dynamic and state-dependent. Static analyses can potentially miss some dynamically occurring common causes

5.2.2.9 Result (output)

The analysis provides a structured overview of identified common cause failures (CCF,) i.e. scenarios in which multiple software functions or systems can be impacted all at once by a common cause. This includes detailed descriptions of the common cause, affected functions and systems as well as der potential impacts on vehicle safety (ASIL), functionality, performance, comfort and cybersecurity. Available protective measures are evaluated with regard to their effectiveness. Based on this evaluation, specific risk reduction recommendations are developed. The process concludes with a qualitative or quantitative assessment of the residual risk remaining, and the analysis process is thoroughly documented.

5.2.2.10 References/Additional reading

International Organization for Standardization. (2018). *Road vehicles – Functional safety (ISO 26262:2018)*. Geneva, Switzerland: ISO.

International Organization for Standardization & SAE International. (2021). *Road vehicles – Cybersecurity engineering (ISO/SAE 21434:2021)*. Geneva, Switzerland: ISO.

5.2.3 CPA (Criticality and Prioritization Analysis)

5.2.3.1 Purpose of the method

The Criticality and Prioritization Analysis (CPA) is a structured method of evaluating software functionalities with regard to their criticality in the overall system. It is used to identify safety-relevant, robust and cyber-resilient functionalities early and prioritize their need for safeguarding. CPA is particularly used in the concept and architecture phase in order to identify risks, intentionally plan resources and establish the foundation for more further analyses (e.g. FMEA, Threat Modeling.)

5.2.3.2 Basic procedure

1. Defining the scope of analysis

- Determining what software functions, components or communication paths should be examined.
- Delineating the system or subsystem (e.g. ADAS-function, control unit, software module.)

2. Identifying the functions and their roles in the system

- Describing the software functions in question and their tasks in the overall system.
- Determining the functional dependencies on other components (e.g. sensors, actuators, control units.)

3. Evaluation of criticality

- Estimating how critical a function is with regard to the following aspects:
    - Functional safety: Does an error lead to a safety-critical state?

    - Operational relevance: Is the function essential for normal driving operation?
    - Cybersecurity: Could an attack on this function have safety-related consequences?
    - System availability: Does a failure impact the overall availability of the vehicle?

Typically, the evaluation is done based on criteria such as severity, probability and detectability – similarly to a FMEA.

4. Prioritization

- Functions are classed according to their criticality (e.g. high, medium, low.)
- This prioritization serves as a basis for:
    - Safety requirements (ASIL rating)
    - Test depth and strategy
    - Architecture decisions (e.g. redundancy, partitioning)
    - Resource allocation in the project

5. Documentation and traceability

- The results of the CPA are documented, e.g. in a table or matrix.
- Traceability to requirements, safety goals and system architecture is ensured.
- The analysis should be updated regularly – especially in case of changes to the design or new findings from tests or reviews.

For a case example of the CPA method as per NIST IR 8179, refer to the Annex, Chapter 7.1.3.

5.2.3.3 Instructions/special considerations/differences in use in the software context

In the software context, the Critical Path Analysis (CPA) helps evaluate software functionalities with regard to their system relevance, robustness and cybersecurity. It is used in early phases of development in order to assess risks and set priorities. Moreover, this method supports both safety and security engineering as well as test planning.

5.2.3.4 Prerequisites (input)

In order to perform a Criticality and Prioritization Analysis (CPA,) we need to have clearly delineated systems and functions, available information on architecture and requirement, defined evaluation criteria, an interdisciplinary team and appropriate tools and templates.

5.2.3.5 Depth of application and termination criteria

The Critical Path Analysis (CPA) is performed on the system and functionality level. The analysis takes place in the use context, regardless of specific implementation details, in order to be able to identify critical dependencies early and evaluate them systemically.

5.2.3.6 Requirements for work, team and tools

The time and effort required for a Critical Path Analysis (CPA) can be rated as medium, since it does mean performing a structured evaluation but without the need for a deep technical analysis. Methodological expertise in the strict sense is not necessary – a basic understanding of system behavior and safety requirements are enough. For a well-founded CPA, it is recommended to have an interdisciplinary team with expertise in software, architecture, safety and security.

5.2.3.7 Benefits in the software context (advantages)

The Critical Path Analysis (CPA) allows for early identification of critical software functionalities and helps provide a holistic view of safety, robustness and cybersecurity. It provides valuable insights for architectural decision-making and the development of safeguarding strategies. Moreover, this method can be effectively combined with other approaches such as the fault tree analysis (FTA,) scenario analyses or threat modeling.

5.2.3.8 Limitations of the method in the software context (disadvantages)

In the software context, the Critical Path Analysis (CPA) does not offer any detailed error analysis like FMEA, for example. It does not allow for any quantitative risk assessment and does not automatically determine actions. In addition, it does not account for technical interactions between system elements, which can limit its significance in complex architectures.

5.2.3.9 Result (output)

The results of a Criticality and Prioritization Analysis (CPA) in automotive software development typically consist of an evaluated and prioritized list of software functions or components, in order of their criticality with regard to safety, operational relevance, availability and, if applicable, cybersecurity.

The results of the CPA serve as a basis for many downstream activities, e.g.:

Area	**Using the CPA results**
Functional safety (ISO 26262)	Developing or refining ASIL ratings, safety requirements
Test strategy	Determining test depth, test priorities and safeguarding measures
Software architecture	Decisions on modularization, partitioning, redundancy
Cybersecurity (ISO/SAE 21434)	Identification of safety-critical attack targets
Project planning	Resource planning and scheduling based on criticality
Change impact analysis	Evaluation of the effects in case of changes to highly critical functions

5.2.3.10 References/Additional reading

NIST Interagency Report 8179: Criticality Analysis Process Model – Prioritizing Systems and Components. National Institute of Standards and Technology, 2018. DOI: 10.6028/NIST.IR.8179

5.2.4 DRBFM (Design Review Based on Failure Mode)

5.2.4.1 Purpose of the method

The DRBFM (Design Review Based on Failure Mode) has, as its objective, to detect potential design or architecture weak points in technical systems early, before more serious implementation efforts can begin. The focus is on change decisions such as new interfaces, migration paths, API contracts, architecture or deployment changes. The method allows for a transparent depiction of the development tasks from these changes in which causes, potential effects and detection mechanisms are formulated and suitable countermeasures are defined. One of the key goals is also cross-departmental communication between Development, Architecture, Safety, QA and Operation in order to ensure a common risk assessment and responsible allocation.

5.2.4.2 Basic procedure

The procedure is described in detail in SAE J2886 (04/2023)
It essentially consists of the following 5 steps:

1. **Establishing the framework of analysis**

   The first step is to establish the framework of analysis. The focus can be on individual parts, connection points, software or hardware components as well as system/subsystem functions. This focus arises, for example, in response to changes to an existing design or from the adoption of parts from new developments.

2. **Ascertaining the functions within the framework of analysis**

   Within the established framework, all the functions performed by the components and their interfaces are ascertained, and, to the extent possible, these functions are assigned systemically. Functions serve as carriers of requirements and make up the framework for the design draft.

3. **Design draft and behavioral analysis**

   The design draft is then carefully analyzed to determine what the behavior is like during all phases of the product life cycle – assembly, testing, storage, transport, installation at customer facility, operation and service. Ideally, this should be done using visualized behavior models.

4. **Identifying weaknesses and risks**

   In this step, possible weaknesses, errors and risks are probed for early so they can be eliminated later on. This is done by searching for the root causes of errors, i.e. parameters in the design or process that cause the functional limits to be exceeded. These parameters often provide indications as to how the errors can be prevented ahead of time.

5. **Planning actions for eliminating potential errors**

   Since corrective actions are the focus of a design review, this final step is particularly important. The result is an action plan that adequately handles the identified risks and weaknesses.

5.2.4.3 Instructions/special considerations/differences in use in the software context

In the software context, DRBFM workshops should primarily take place in case of significant design decisions or changes, e.g. changes of architecture, introduction of new interfaces, migration paths, performance improvements or safety-related actions. Instead of physical failure modes, software failure modes are used, i.e. breaches of the API contract, data inconsistencies, race conditions, misconfigurations of deployments or problems in the data flow. The approach is modular and starts, for example, with a clear, risky change-set and is iteratively expanded to other changes. The use of a software-specific template that includes fields for change, failure mode, causes, effects, detection, mitigation, risk assessment, managers and deadlines, helps the users with implementation. To maximize its advantages, the DRBFM can be permanently integrated into the software life cycle. This way, results are linked with requirements reviews, architecture reviews, API contract reviews, safety reviews and automated tests.
The reliability and safety goals must be adapted to the software product (e.g. failure rates, downtimes, safety gaps, user acceptance.)
Instead of physical components, software design decisions, interfaces, architectures, algorithms and configuration parameters are checked.

5.2.4.4 Prerequisites (input)

For an effective DRBFM in the software environment, we need a clear (change) description that explains the “what,” “why” and “how.” This includes architecture or design diagrams such as component diagrams, sequence or package diagrams as well as data flow diagrams that allow us to trace the effects of the change. We also need interface and contract documents, e.g. API contracts, data models, contract tests or schematic changes. Risk criteria help us to prioritize the issues, e.g. safety relevance, customer benefit or risks, that could result from changes. A detection plan with planned metrics, logs, tests, canary strategies or static/dynamic analysis is helpful, and so is having clearly defined responsibilities: Who is the change owner? Who is in charge of architecture, security, QA and operation? If available, prototypes or pilot results can be used for reference in order to get a better idea of the practical effect.

5.2.4.5 Depth of application and termination criteria

The depth of application depends on the risk of the software solution or change: A deep application includes all documented failure modes, causes, effects, detection mechanisms and mitigation measures. A medium depth focuses on a clear list of failure modes and their detection as well as countermeasures. A superficial application is suitable for less risky solutions or changes and focuses on the essential risks and countermeasures. The termination criteria include that all identified failure modes are evaluated, along with the causes, effects, detectors, countermeasures, responsibilities and deadlines. The risk assessment should be clearly prioritized, and detection mechanisms in the form of tests, monitoring or canary releases should be established. All results should be linked in relevant documents or recorded in build/release documentation. Open items or dependencies must be escalated or scheduled in the next review meetings.

5.2.4.6 Requirements for work, team and tools

The necessary time and effort depends on the depth of the DRBFM session: Low means a meeting that takes about one hour with 1-2 change requests and minimal documentation. Medium indicates a session of 90 to 150 minutes with 2-4 changes, structured documentation and one or two reviews. High requires multiple sessions, more than five changes, comprehensive documentation and integration in release plans and, if necessary, additional security or data protection reviews. The team spectrum typically includes a change owner from Development or Architecture, Backend or Frontend architects, a QA/test lead, a security owner for safety-related issues, DevOps/Operation and, if necessary, a product owner and a compliance representative. The tools used are: central documentation in Wikis, diagram tools such as Visio, Lucidchart or diagrams.net, ticket systems such as Jira, Azure DevOps or GitHub Issues, tests and monitoring tools (contract tests, integration tests, logging/observability platforms) as well as templates especially for software DRBFM.

5.2.4.7 Benefits in the software context (advantages)

The main advantage of DRBFM in the software context is that it makes it possible to identify risks early and define targeted countermeasures before an implementation gets expensive or goes live. This boosts the quality and reliability of the system, because it allows us to deliberately respond to potential problems. Clearly defined responsibilities and deadlines reduce the necessary coordination work and enhance collaboration across Development, Architecture, Safety, QA and Operation. DRBFM supports better release planning, since risks can be factored into planning; canary deployments or feature flags can be utilized in a more targeted manner. In addition, it results in a comprehensible decision-making documentation that makes the reasons for design decisions transparent and provides a clear audit history.

It is especially technology- and practice-oriented, since it performs the risk assessment when important and relevant changes are about to be decided, and the subsequent implementation can provide feedback on the decision quickly.

5.2.4.8 Limitations of the method in the software context (disadvantages)

DRBFM can especially lead to a significant amount of review work in case of extensive changes with many failure modes and comprehensive documentation. The estimates regarding causes and effects are based largely on the experiences of the team, which can result in subjectivity, and carries the risk of under- or overestimation. The risk assessment likewise results from the experience of the team and is based on the potential effects of the identified failure modes with a high/medium/low risk scale; an assessment of the occurrence is generally not considered for the prioritization of risks: The team goes over all the identified risks until a basis for decision-making is achieved.

Long-term or architecture risks following individual changes get less consideration. Unless the defined action plan is consistently followed up on, the benefit may be lost. Not all organizations need a complete DRBFM session; the method must be adapted as appropriate for the context.

5.2.4.9 Result (output)

The key result of a DRBFM meeting in the software context is a comprehensive change register containing the change requests with descriptions, followed by the identified failure modes per change, the suspected causes, the potential effects as well as detection mechanisms and appropriate mitigation measures. The output is helped by a risk assessment per failure mode, as well as clear responsibilities and deadlines. Links to architecture documents, API contracts, test suites and release plans allow for seamless tracking. An additional output is provided by a release or gate level which indicates whether or not the change may pass the next step in the release process, e.g. according to code or security reviews. Optionally, templates with specific fields such as change/design decision, potential failure modes, causes, effects, detection, mitigation, risk scale, owner/deadline as well as notes/open items can be used in order to standardize the documentation.

5.2.4.10 References/Additional reading

SAE International. *Design Review Based on Failure Modes (DRBFM)*, SAE Standard J2886_202304, Reaffirmed April 2023, Issued March 2013. DOI: 10.4271/J2886_202304

5.2.5 ETA (Event Tree Analysis)

5.2.5.1 Purpose of the method

Event tree analysis (ETA) is a method of systematically analyzing and representing the logical links of component and subsystem failures in order to reveal the effects of potential undesirable events as well as their functional relationships. Unlike fault tree analysis, ETA follows a bottom-up procedure – the analysis starts with an initial event and moves on to the potential subsequent events, branching off from there. The objective is to record the various development paths of a system following a given event and evaluate their consequences.

5.2.5.2 Basic procedure

1. **Defining the initiating event**
    - Example: “Software module goes down” or “sensor outputs erroneous data.”
2. **Identifying relevant safety functions or barriers**
    - What protective measures or responses are set up? (e.g. failure detection, fallback level, restarting mechanism)
3. **Structure of the event tree**
    - A path with two possible outputs is modelled for every barrier: **Success** (barrier works) or **failure** (barrier fails.)

   - This produces various **event paths**, each leading to a certain final outcome.

4. **Describing the outcomes**
   - Each path ends in an event, e.g. “System stabilizes,” “subfunction fails,” “total system failure.”

5. **Quantitative evaluation**
   - We can assign a probability to each path based on the probability of success/failure of the barriers.
   - This gives us the **probability of every outcome**.

6. **Interpretation and action plan**
   - Identification of high-risk paths.

   Development of improvement measures (e.g. additional barriers, increasing the reliability.)

5.2.5.3 Instructions/special considerations/differences in use in the software context

In the software context, the event tree analysis should be geared to identifying the critical path. The starting point is an initial event – for example, a breach of functional requirements or an undesired system state – from there, we can analyze potential subsequent events and their ramifications. Crucial to the application of the method is the determination of suitable basis events, i.e. the question of what specific events or states in the software system might occur and how they could logically progress. The method is especially suitable for representing reaction chains and for analyzing system behavior under various conditions, e.g. in case of incorrect inputs, communication breakdowns or unmet requirements.

5.2.5.4 Prerequisites (input)

In order to use event tree analysis in the software context, certain prerequisites need to be met. There must be a software functionality to analyze and a function architecture – also known as functional architecture or composition – that makes up the logical connections within the system. In addition, some undesired events of individual components or functionalities need to have already been identified in order to serve as the starting point for the analysis. Only when these elements are clearly described is it practical to use ETA to systematically examine potential development paths and their consequences.

5.2.5.5 Depth of application and termination criteria

The application depth of event tree analysis depends on the identification of a suitable start level that serves as the starting point for the analysis. The depth of the analysis is not generally limited. Rather, it largely depends on the complexity of the given system and the scope of the event paths to be explored. The more detailed the examination of the branches and subsequent events, the greater the time and effort required for modeling and evaluation.

Reasonable limits can be set for the analysis through logical encapsulation, especially when the analysis of the dynamic and static architecture is complete. In such cases, it makes sense to modularize or segment the event paths in order to keep things clear and manageable and make the analysis more efficient.

5.2.5.6 Requirements for work, team and tools

Using event tree analysis in the software context requires a medium to high degree of time and effort. Since the basic principles of ETA are easy to understand and structured in application, the required method expertise is relatively low. However, the analysis requires careful modeling of the event paths and clearly defined starting and subsequent events.

Solid execution typically requires an interdisciplinary team that understands both the functional architecture and the system behavior in case of error. The collaboration of software developers, system architects and safety experts makes a big difference in the quality and significance of the analysis. The use of auxiliary tools for visualization and documentation can facilitate the analysis process but is not strictly necessary.

5.2.5.7 Benefits in the software context (advantages)

In the software context, event tree analysis offers several specific advantages. It is well-adapted for analyzing errors in encapsulated components and systematically tracking their consequences. The method allows for a structured representation of possible reaction paths following the occurrence of an initial event, thus making it easier to track complex system behavior. Due to its strong focus, ETA can be used for the targeted analysis of critical processes and is suitable for both preventive use – for early detection of potential risks – and for reactive analysis of malfunctions that have already occurred. The method is particularly effective in combination with fault tree analysis (FTA.) Combining the two approaches allows us to comprehensively map both causes and consequences of undesired events.

5.2.5.8 Limitations of the method in the software context (disadvantages)

Despite its strengths, event tree analysis has several methodological limitations in the software context. The analysis is not comprehensive. Rather, it focuses on individual event paths and isolated risks. No risk mitigation measures are defined, nor is there any systematic definition of options for action or any guidance for handling the identified risks. The method also does not include any risk evaluation for the purpose of prioritization or weighting. So ETA primarily provides a structured mapping of possible system reactions to an initial event without integrating any further steps in the risk management.

5.2.5.9 Result (output)

The result of an event tree analysis consists of a structured representation of possible system reactions to an initial event as well as the associated branches and subsequent events. In the software context, this output can be used to derive evaluated risks, typically based on a qualitative evaluation. Unlike with hardware, there are no reliable failure probabilities available for software, so a quantitative risk estimate is not possible in most cases. However, ETA does provide valuable insights into potential failure effects and is useful for the systematic analysis of reaction paths within complex software architectures.

5.2.5.10 References/Additional reading

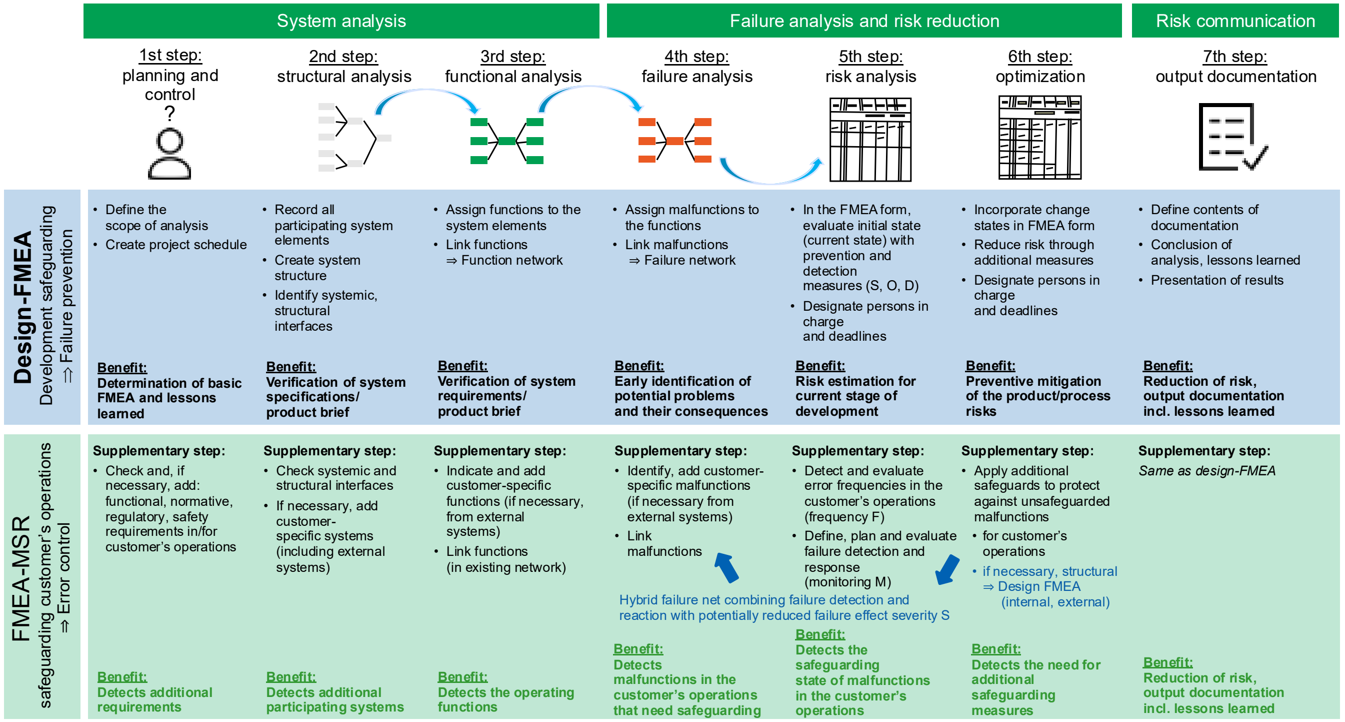
IEC 62502:2010 „Analysis techniques for dependability – Event Tree Analysis (ETA)“, International Electrotechnical Commission

5.2.6 FMEA (failure mode and effects analysis)

5.2.6.1 Purpose of the method

Failure mode and effects analysis (FMEA) is an established, systematic method of early detection and evaluation of potential sources of error in the product development process. Its objective is to identify weak points in the technical functionality in the early stages of development and analyze their possible effects on the overall system. In a risk analysis process, potential errors, their causes and their consequences are identified and evaluated in a structured manner. Based on the results, we can develop a targeted action plan with the aim of preventing or mitigating errors. The use of FMEA thus greatly helps to proactively manage risks and increase the reliability of software-based functions long-term.

5.2.6.2 Basic procedure



	System analysis			Failure analysis and risk reduction			Risk communication
	1st step: planning and control	2nd step: structural analysis	3rd step: functional analysis	4th step: failure analysis	5th step: risk analysis	6th step: optimization	7th step: output documentation
Design-FMEA Development safeguarding ⇒ Failure prevention	• Define the scope of analysis • Create project schedule **Benefit:** **Determination of basic FMEA and lessons learned**	• Record all participating system elements • Create system structure • Identify systemic, structural interfaces **Benefit:** **Verification of system specifications/ product brief**	• Assign functions to the system elements • Link functions ⇒ Function network **Benefit:** **Verification of system requirements/ product brief**	• Assign malfunctions to the functions • Link malfunctions ⇒ Failure network **Benefit:** **Early identification of potential problems and their consequences**	• In the FMEA form, evaluate initial state (current state) with prevention and detection measures (S, O, D) • Designate persons in charge and deadlines **Benefit:** **Risk estimation for current stage of development**	• Incorporate change states in FMEA form • Reduce risk through additional measures • Designate persons in charge and deadlines **Benefit:** **Preventive mitigation of the product/process risks**	• Define contents of documentation • Conclusion of analysis, lessons learned • Presentation of results **Benefit:** **Reduction of risk, output documentation incl. lessons learned**
FMEA-MSR safeguarding customer's operations ⇒ Error control	**Supplementary step:** • Check and, if necessary, add: functional, normative, regulatory, safety requirements in/for customer's operations **Benefit:** **Detects additional requirements**	**Supplementary step:** • Check systemic and structural interfaces • If necessary, add customer-specific systems (including external systems) **Benefit:** **Detects additional participating systems**	**Supplementary step:** • Indicate and add customer-specific functions (if necessary, from external systems) • Link functions (in existing network) **Benefit:** **Detects the operating functions**	**Supplementary step:** • Identify, add customer-specific malfunctions (if necessary from external systems) • Link malfunctions Hybrid failure net combining failure detection and reaction with potentially reduced failure effect severity S **Benefit:** **Detects malfunctions in the customer's operations that need safeguarding**	**Supplementary step:** • Detect and evaluate error frequencies in the customer's operations (frequency F) • Define, plan and evaluate failure detection and response (monitoring M) **Benefit:** **Detects the safeguarding state of malfunctions in the customer's operations**	**Supplementary step:** • Apply additional safeguards to protect against unsafeguarded malfunctions • for customer's operations • if necessary, structural ⇒ Design FMEA (internal, external) **Benefit:** **Detects the need for additional safeguarding measures**	**Supplementary step:** *Same as design-FMEA* **Benefit:** **Reduction of risk, output documentation incl. lessons learned**

Figure 5-4: Basic process of design FMEA and FMEA-MSR

A design FMEA is a seven-step method. First, in the planning and preparation phase, the scope of analysis is established, a project plan is created, and the documentation contents are defined. Next, in the structural analysis, the system elements for the scope of analysis are recorded and mapped in a system structure. Both systemic and structural interfaces are identified. In the functional analysis, the functions are associated with the system elements, and these functions are linked in a function network. In the failure analysis, we build on these results by identifying potential malfunctions, associating them with functions and linking them up in an error network. The risk analysis evaluates the identified risks based on the criteria Severity (S,) Occurrence (O) and Detection (D) in the current state and documents them in the FMEA template. In the optimization phase, a risk mitigation action plan is developed, and responsibilities and deadlines are set. After implementation of the action plan, potential occurrence and detection are re-evaluated. Finally, the results of the analysis are compiled and documented.

5.2.6.3 Instructions/special considerations/differences in use in the software context

When applying FMEA to software-based functionalities, the focus is not on individual lines of code or physical failures but rather on the functional processes within the software and on logical failures. The objective is to systematically identify potential weak points in the software logic (e.g. faulty state transitions, incomplete requirements, incomplete interface definitions or inadequate error control) and to assess their effects on the overall functionality. The analysis is performed on an abstract level that is geared to the functional architecture and intended operating states. As a result, risks can be detected early, and a targeted action plan can be developed for improving the robustness and reliability of software-based systems.

5.2.6.4 Prerequisites (input)

The effective application of FMEA to software-based functionalities requires the availability of central system information. This includes the description of the software functionalities as well as system and software architecture along with its interfaces, data flows and operating states. Also required is knowledge of the functional architecture, also known as function architecture or function composition. This information forms the basis for a structured analysis of potential causes and effects of failure. As with the fault tree analysis (FTA) procedure, clearly defining these input variables allows for a targeted and clear risk analysis in the software context.

5.2.6.5 Depth of application and termination criteria

When applied in the software context, FMEA focuses on the software functions involved in the implementation or execution of the relevant system requirements. The main focus is on the functional processes and how they contribute to the overall functionality of the system – not the technical implementation at the code level. The analysis continues until all relevant functions have been evaluated with regard to potential causes and effects of failure and appropriate risk mitigation measures have been defined. As a termination criterion, there must be a clear risk assessment for all analyzed functions, and there must not be any open, unmanaged risks left which could endanger compliance with the system requirements.

5.2.6.6 Requirements for work, team and tools

Performing an FMEA in the software context takes a lot of work and organization. Due to the complexity of software-based functionalities and the many possible causes of failure, this method requires a high level of expertise in risk analysis as well as well-founded knowledge of the software architecture and functional relationships. The analysis should be performed by an interdisciplinary team that combines both system knowledge and software development and quality assurance expertise. To help with structured implementation and documentation, we need suitable tools that allow us to model functional processes, evaluate risks and keep track of the action plan.

5.2.6.7 Benefits in the software context (advantages)

The application of FMEA to software-based functionalities provides structured and comprehensible support for the development of strategies for preventing, detecting and mitigating failures. The systematic analysis of functional processes and potential causes of failure allows us to identify risks early and zero in on them. This not only makes the software more robust but also promotes a deeper understanding of critical connections within the system architecture. So this method is very helpful for quality assurance and for improving the functional safety of software-intensive systems.

5.2.6.8 Limitations of the method in the software context (disadvantages)

FMEA does not guarantee the completeness of the risk analysis. It is based on the systems, processes and functions known at the time of implementation, so it can only provide insights within this defined context. Risks resulting from undetected, unspecified or unmodelled states are excluded from the scope of analysis. This limitation is especially significant in the software context, where complex state transitions and emerging behaviors can occur. The significance of the analysis thus depends greatly on the quality and completeness of the input information.

5.2.6.9 Result (output)

The output of the FMEA in the software context consists of the systematic identification of critical and unsecured states within the analyzed functions and processes. The structured evaluation of potential failure causes and effects specifically identifies weak points that can pose a risk to the reliable implementation of the system requirements. The analysis provides specific safeguards that help prevent, detect or mitigate failures, thus strengthening the functional safety and robustness of software-based systems.

5.2.6.10 References/Additional reading

AIAG & VDA (2019): FMEA Handbook. Design-FMEA, Process FMEA, FMEA-MSR (monitoring and system response). First edition, June 2019.

5.2.7 FMEA-MSR (FMEA monitoring and system response)

5.2.7.1 Purpose of the method

FMEA-MSR (monitoring and system reaction) takes conventional FMEA and adds the analysis of risks in the customer's actual operations. The goal of the method is to identify critical operating states that can occur during product use and lead to safety-related, functional or regulatory risks. The systematic analysis of these states and the existing monitoring and reaction mechanisms is used to check if potential risks are adequately safeguarded against. FMEA-MSR thus helps demonstrate that the systematic works robustly and safely even under actual operating conditions.

5.2.7.2 Basic procedure

FMEA-MSR follows the same seven-step method as design FMEA (see Figure 5-4) but additionally includes specific aspects that are important in the customer's operations. In the planning and preparation phase, additional customer-specific requirements and operating conditions are accounted for. The structural analysis is expanded to include external and customer-operated systems and interfaces. In the functional analysis, operating functions are identified and integrated into existing function networks. The failure analysis includes the identification of customer-specific malfunctions that are linked with existing error networks. The risk analysis is expanded to include the evaluation of error frequency in the customer’s operations (F) and monitorability (M). Concepts for failure detection and response are planned and evaluated. In the optimization phase, unsafe malfunctions are targeted and secured with additional safeguards, especially with regard to their occurrence in customer operations. The output documentation includes the expanded analysis results, monitoring concepts and lessons learned for safeguarding the operational status.

5.2.7.3 Instructions/special considerations/differences in use in the software context

In the software context, FMEA-MSR is directed at malfunctions that can occur in actual customer operations and lead to safety-related, regulatory or functional risks. The focus is not only on the software functions themselves but particularly on their behavior under operating conditions, e.g. erroneous state transitions, inadequate plausibility checks or missing reaction mechanisms. This method helps systematically evaluate whether or not critical states are detected and safeguarded by means of suitable monitoring and reaction strategies, e.g. with warnings, contingency strategies or fallback levels.

5.2.7.4 Prerequisites (input)

A solid execution of FMEA-MSR in the software context requires comprehensive information on safety-related, regulatory and functional requirements. In particular, this includes the safety goals and their ASIL classifications, normative and legal specifications as well as detailed system and functional descriptions. Also necessary are interface descriptions for adjacent systems, the relevant operating states, known possible failures from upstream analyses (e.g. D-FMEA or FTA) and failure networks. If available, information on diagnostic capabilities and existing reaction strategies are also factored in. The analysis is supplemented with qualitative, attributive and (if available) quantitative evidence of the safeguarding of the risks in the customer’s operations.

5.2.7.5 Depth of application and termination criteria

The FMEA-MSR is typically performed at the system and subsystem level, since this is the best level at which to detect the relevant operating states and their potential risks in the customer's operations. For a deeper understanding of potential errors, however, it can be helpful to also observe the component or module level – especially if critical states occur there or are monitored there. The analysis is considered complete once all the relevant operating states have been evaluated with regard to their monitorability and response capability and there are no more unmanaged risks remaining that could endanger the safety-related, regulatory or functional requirements.

5.2.7.6 Requirements for work, team and tools

Performing an FMEA-MSR takes a great deal of time and effort and requires a high level of methodological expertise. Due to the complexity of the analysis (particularly with regard to the evaluation of operating states, diagnostic capabilities and reaction strategies,) an interdisciplinary team is necessary. This team should have expertise in the areas of functional safety, software development, system architecture and quality management. For structured implementation and documentation, we need suitable tools for modelling operating states, performing risk analysis and developing and tracking of the action plan.

5.2.7.7 Benefits in the software context (advantages)

In the software context, the FMEA-MSR provides structured support for the development and evaluation of strategies for preventing, detecting and mitigating failures in the customer's actual operations. Through the targeted analysis of critical operating states and the establishing of corresponding safeguards, this method helps us reliably meet safety and regulatory requirements, including under operating conditions. It promotes a deeper understanding of the effectiveness of diagnostic and reaction mechanisms and helps to continuously improve the functional safety of software-heavy systems.

5.2.7.8 Limitations of the method in the software context (disadvantages)

FMEA-MSR does not provide any information as to the completeness of the risk analysis. It is based on known and defined operating states and therefore cannot detect risks that stem from unknown or unspecified states. Moreover, this method has no impact on the probability of occurrence of malfunctions – it only evaluates their effects and the effectiveness of available monitoring and reaction strategies. The significance of the analysis thus depends heavily on the quality and completeness of the input information and the system description.

5.2.7.9 Result (output)

The output of the FMEA-MSR consists of the proof of quality of the safe-guarding of safety, regulatory and normative requirements in the customer's operations. Through the structured evaluation of critical operating states and the available monitoring and reaction mechanisms, we can see whether potential risks are appropriately detected and managed. The analysis thus provides a solid foundation for the evaluating the functional safety and robustness of software-based systems under actual use conditions.

5.2.7.10 References/Additional reading

AIAG & VDA (2019): FMEA Handbook. Design-FMEA, Process FMEA, FMEA-MSR (monitoring and system response). First edition, June 2019.

5.2.8 FMEDA (Failure Modes, Effects and Diagnostic Analysis)

5.2.8.1 Purpose of the method

FMEDA (Failure Modes, Effects and Diagnostic Analysis) is used as a quantitative evaluation of the functional safety of technical systems in accordance with ISO 26262. The objective is to systematically identify potential failure modes, analyze their effects on the safety goals and evaluate the effectiveness of available diagnostic mechanisms. This is used as a basis for calculating safety-related parameters such as the SPFM (Single Point Fault Metric) and the LFM (Latent Fault Metric,) which are required for safety approval and compliance with regulatory requirements. This method is primarily used in electronic system architectures. Its potential for application to purely software-related scopes is very limited.

5.2.8.2 Basic procedure

The procedure includes the following steps:

1. **Identifying the components (hardware or software elements):** This analysis is based on a complete system description in which all the relevant components (both hardware and software) are listed.

2. **Determining the possible failure modes:**
   For every component, potential failure modes are systematically identified, e.g. short-circuit, memory error or loss of communication.

3. **Analyzing the effects on the safety goals:**
   The effects of the failures on the defined safety goals are evaluated, particularly as they relate to the breach of functional requirements or danger to persons.

4. **Allocation of diagnostic mechanisms and their coverage:**
   For every failure mode, we check if there is a diagnostic mechanism in place and how reliably it detects the error. The diagnostic coverage is a key parameter for the subsequent evaluation.

5. **Calculating safety parameters (e.g. SPFM, LFM):**
   Based on the failure rates and diagnostic coverages, we can calculate quantitative indicators like the Single Point Fault Metric (SPFM) and the Latent Fault Metric (LFM) that are required for safety approval.

6. **Documenting the results for the safety case:**
   The analysis results are recorded in structured documentation and are used as documented evidence for customers, auditors or certification bodies. They are an integral part of the safety concept.

5.2.8.3 Instructions/special considerations/differences in use in the software context

Unlike with hardware analysis, when the FMEDA is applied to software, the focus is not on physical failures but on systematic errors – for instance, faulty algorithms, data corruption or inadequate error control. In this context, the diagnostic mechanisms are mostly logically structured, e.g. through plausibility checks, watchdog functions or redundancy checks. Particularly challenging is the quantification of failure probabilities, since they cannot be measured directly in the case of software. It is common, therefore, to conduct a qualitative evaluation or make assumptions based on empirical values. Using FMEDA for software requires close collaboration with the development and quality assurance teams in order to make the analysis realistic and reliable.

5.2.8.4 Prerequisites (input)

A well-implemented FMEDA requires several pieces of input information. These include, first of all, the system architecture and the defined safety requirements that provide the framework for the analysis. A complete list of the software components and their functions is the basis for the identification of potential failure modes. Also required is a database of failure modes, which is often produced in earlier FMEA activities. The available diagnostic mechanisms and their effectiveness must also be known in order to be able to evaluate the coverage. With hardware, we have reliable failure rates to use. With software, however, the analysis usually relies on error assumptions or empirical values, since reliable statistics are often not available.

5.2.8.5 Depth of application and termination criteria

FMEDA is typically performed down to the element level, i.e. for individual hardware components like integrate circuits (ICs) or for specific software modules. This depth of detail allows for a precise evaluation of the failure modes and the effectiveness of diagnostic mechanisms. The analysis is considered complete once all the relevant failure modes have been identified, their effects have been evaluated, suitable diagnostic measures have been assigned, and the required safety metrics (like SPFM and LFM) have been calculated. The results are incorporated into the safety case and form the basis for the approval of safety-critical systems.

5.2.8.6 Requirements for work, team and tools

Implementing FMEDA requires an interdisciplinary team, comprising safety experts, software and hardware developers and, if necessary, system architects. This analysis is often performed using table calculations or specialized FMEDA tools that facilitate the structured tracking and evaluation of failure modes, diagnostic coverages and safety metrics. This method usually takes a lot of work, since it requires a detailed examination at the component level and incorporates both functional both diagnostic aspects. Complex systems with high standards of safety require particularly careful planning and coordination.

5.2.8.7 Benefits in the software context (advantages)

Although it was originally developed for hardware components, FMEDA has real advantages in the software context. It provides quantitative safety cases that can be used to help meet the requirements of ISO 26262. It is also helpful for the validation and evaluation of logical diagnostic mechanisms like plausibility checks or monitoring functions. The structured analysis significantly increases our understanding of critical failure paths within the software architecture, which in turn facilitates the development of safety measures.

5.2.8.8 Limitations of the method in the software context (disadvantages)

The application of FMEDA to software comes with specific challenges. Unlike with electronic components, there are no standardized failure probabilities available for software elements, which makes quantitative evaluation difficult. We often have to make assumptions based on empirical values, which can make the analysis less reliable. With complex software, the method also takes a great deal more time and effort, since it requires a detailed examination of all relevant modules and failure paths. Without a clearly defined objective, there is a risk of over-documentation – documentation that is comprehensive but does not actually provide added value for the safety assessment.

5.2.8.9 Result (output)

The main output produced by FMEDA is a structured table documenting the identified failure modes, the associated diagnostic mechanisms and their degrees of coverage. This data is used to calculate the safety metrics, e.g. the SPFM (Single Point Fault Metric), the LFM (Latent Fault Metric) and the PMHF (Probabilistic Metric for Hardware Failures.) These metrics serve as quantitative proof of safety conformity as per ISO 26262 and are essential for the approval of safety-critical functions and systems.

5.2.8.10 References/Additional reading

ISO 26262:2018 – Road Vehicles – Functional Safety. International Organization for Standardization (ISO), Geneva.

SAE J2980:2018 – Considerations for ISO 26262 ASIL Hazard Classification. SAE International.

Ross, Hans-Leo: *Functional Safety for Road Vehicles – New Challenges and Solutions for E-Mobility and Automated Driving*. Springer, 2016. ISBN: 978-3-319-33361-8

5.2.9 FTA (Fault Tree Analysis)

5.2.9.1 Purpose of the method

The fault tree analysis (FTA) is used for systematic analysis and mapping of malfunction-related logical links of component and subsystem failures. Its objective is to provide a clear picture of the effects of possible undesired events and their functional relationships. The method follows a top-down approach: Starting from a defined top event (e.g. a safety-critical system failure,) the underlying root causes are traced in a tree-type structure. The logical links (or gates) of the malfunctions are drawn based on Boolean logic, using AND- or OR-gates, for example.

Analyses can be both **qualitative** and **quantitative**:

- A qualitative analysis centers on the identification of critical failure paths and weak points.
- A quantitative additionally allows for the evaluation of a software error's probability of occurrence, provided that the necessary data (e.g. failure rates, test coverages) is available.

5.2.9.2 Basic procedure

The procedure includes the following steps:

1. Define the top event

- The top event is the undesired behavior at the system level that can be caused by a software error.
- Example: *"Inappropriate activation of the emergency brake function"* or *"Failure to detect an obstacle by the ADAS[4]."*

2. Obtain an understanding of the system

- Gather information on the system (e.g. circuit diagrams, function descriptions, expertise.)
- Analyze the affected software function in the context of the overall system:

[4] ADAS stands for "Advanced Driver Assistance Systems." These systems assist the driver with automatic functions like lane assist, brake assist or intelligent speed assistance. They are considered a precursor to automated and autonomous driving and are a main component of modern vehicle safety and comfort technology.

    - architectural diagrams, function descriptions, software and hardware interfaces.
    - Connect with specialists (e.g. function developers, software architects, safety engineers.)
    - Integrate the requirements stipulated in standards such as ISO 26262.

3. Create the fault tree

- Break the top event down into possible causes using logic gates:
    - UND-gate: multiple conditions must be met at once.
    - OR-gate: one of multiple conditions is enough.
- Typical causes in the software context:
    - Incorrect requirements or specifications.
    - Logic errors in algorithms.
    - Communication errors between software modules.
    - Incorrect sensor values or inadequate validation.
    - Timing problems or race conditions.

4. Continue mapping down to the basic elements

- The analysis is refined until the basic elements are reached:
    - Individually software modules, functions, interfaces or external influences.
    - Human errors (e.g. in implementation or configuration) can also be basic elements.

5. Perform the analysis

- Qualitative: Identify failure paths, critical components and logical dependencies.
- Quantitative: Evaluate the probability of occurrence of the top event to the extent that data such as failure rates, test coverage or diagnostic effectiveness is available.

6. Evaluate and interpret

- Analyze the minimal cut sets for the identification of:
    - Critical failure combinations.

    - Very relevant individual causes of failure.
    - Weak points in the software or in the error control.

7. Document and review

- Fully document the fault tree, the assumptions and the results.
- Perform reviews with interdisciplinary teams (software, safety, test, architecture.)
- Ensure traceability to requirements and safety goals.
- Update the fault tree in case of changes to software, architecture or requirements.

5.2.9.3 Instructions/special considerations/differences in use in the software context

In the software context, the use of the fault tree analysis (FTA) is possible with certain limitations and should be targeted at the identification of the critical path. The starting point is a defined critical top event from which the relevant logical links are analyzed. The focus is on the determination of suitable basis events, i.e. the question of what specific malfunctions or undesired states in the software system might occur. Since software faults often cannot be attributed to physical failures but rather on logical or systematic causes, it is essential to precisely define the basis events.

Boolean logic is still used in the software context. However, the quantitative portion of the FTA, e.g. the calculation of probabilities of occurrence, is usually not applicable to software, since reliable statistics usually cannot be obtained for software errors. This makes the method primarily suitable for qualitative analysis, especially for structured identification of failure paths and for developing action plans for failure prevention or mitigation.

5.2.9.4 Prerequisites (input)

In order to use fault tree analysis in the software context, certain prerequisites need to be met. First, we need a software functionality to analyze and an undesired event that is typically described as a violation of the defined goal. We also need a function architecture – also known as functional architecture or composition – in order to clearly map the logical connections and dependencies within the system.

The method should be used proactively, i.e. in the early phases of development, in order to systematically identify potential risks. The analysis is performed by negating functionalities and requirements or a possible violation of the safety goal in order to reveal possible undesired events. So when the method is applied preventively, the focus is not on the retrospective analysis of the failure after it has occurred but rather on proactively identifying critical paths and weak points in the system design.

5.2.9.5 Depth of application and termination criteria

The depth of application of the fault tree analysis is essentially unlimited and depends on the complexity of the given system as well as the relevance of the analyzed top event. Depending on the objective, the analysis can be pursued down to the deeper functional levels in order to also detect indirect or cross-system relationships. The decision as to the depth of analysis depends on the situation and is based on the severity and criticality of the top event in the overall system.

Termination criteria are typically met when all the relevant basis events are identified and the logical linking has been completed – or when there are no more insights to be gained from going into further detail. In the software context, the analysis is terminated once the critical paths have been adequately described and potential weak points in the functional architecture have been identified.

5.2.9.6 Requirements for work, team and tools

Performing a fault tree analysis in the software context takes a lot of time and effort, regardless of the actual complexity of the system. Although quantitative method expertise is low due to the limited applicability of statistical methods, qualitative analysis requires a deep understanding of functional relationships and a high level of methodological know-how. Creating and maintaining the tree structure, especially with complex software architectures, is resource-intensive and demands careful and structured procedures.

Using this method does not necessary require a dedicated team, but including specialists in software architecture, safety or system development can significantly improve the quality of the analysis. The use of auxiliary tools for modeling and visualization can be helpful but is not required. The decisive factor is the ability to precisely identify and map the logical linking and functional dependencies.

5.2.9.7 Benefits in the software context (advantages)

In the software context, fault tree analysis (FTA) offers several specific advantages. This method is highly focused and is particularly well-suited for doing a targeted deep dive into functional relationships and root causes. Structure top-down analysis allows us to systematically identify critical path and ascertain what has high value, especially in complex software architectures. FTA can be used both preventively (for early detection of potential risks) and reactively (to analyze malfunctions that have already occurred.) Its advantage is in the ability to do a targeted deep dive while shedding light on logical links.

5.2.9.8 Limitations of the method in the software context (disadvantages)

Despite its strengths, fault tree analysis does have significant methodological limitations in the software context. The analysis is not comprehensive. Rather, it focuses on individual failure paths and isolated risks. No risk mitigation measures are defined, nor is there any systematic definition of options for action or any guidance for handling the identified risks. The method also does not include any risk evaluation for the purpose of prioritization or weighting.

FTA thus mainly provides a structured mapping of root causes and their logical linking, without integrating any further steps in risk management, however. As an exception, the quantitative application of FTA (provided that reliable data is available) can be used to make a statement on the probability of failure of the top event. In such cases, it is possible to make an accurate risk estimate, which can be especially significant in safety-critical systems.

5.2.9.9 Result (output)

The output of a fault tree analysis consists of a structured representation of the logical relationships between malfunctions and their causes, starting from a defined top event. In the hardware context, this can be used to develop evaluated risks, since it is often a source of reliable data for calculating failure probabilities. In the software context, however, such a quantitative evaluation is not possible, since there are no statistically sound failure probabilities for software functions. The analysis therefore mainly provides qualitative insights into critical paths and potential weak points which can be used as a basis for other safety-related evaluations and measures.

5.2.9.10 References/Additional reading

VDA Volume 4, Section 2 “Risk Analyses”, 3rd edition, August 2020

5.2.10 HARA (Hazard Analysis and Risk Assessment)

5.2.10.1 Purpose of the method

The objective of a Hazard Analysis and Risk Assessment (HARA) is to identify potential hazards that can arise due to malfunctions of software-based functionalities. After that, each risk is assessed in order to develop appropriate safety goals that reduce the risk to an acceptable level. Finally, the Automotive Safety Integrity Level (ASIL) is determined. This represents the level of safety required in order to implement the safety goals.

5.2.10.2 Basic procedure

The procedure includes the following steps:

1. Definition and description of the analyzed system or subsystem (item definition,) indicating the relevant functions, operating states and system boundaries.
2. Identification of hazards: Analysis of potential malfunctions of the system that can lead to hazards in driving. The environment, the driver and other road users are taken into account.
3. Evaluation of hazards: For each hazard, the parameters severity (S), exposure (E) and controllability (C) are determined.
4. Risk classification: Based on the combination of S, E and C, we can determine the Automotive Safety Integrity Level (ASIL).
5. Formulation of safety goals: For hazards with an unacceptable level of risk, functional safety goals are developed in order to bring the risk down to an acceptable level.
6. Documentation and release: Results are documented and released.
7. Iterative improvement: In case of changes to the system concept or newly identified hazards, the HARA is updated in order to ensure that all the safety goals are up-to-date and complete.

5.2.10.3 Instructions/special considerations/differences in use in the software context

HARA is a high-level-analysis method based on the functional behavior of the vehicle. A detailed system or software architecture is not required for its implementation. As a result, the application of HARA as a risk analysis method for software-based functionalities is only possible to a limited extent.

5.2.10.4 Prerequisites (input)

HARA implementation requires an item definition. This document describes the analyzed functionality at the vehicle level and serves as a basis for the identification of potential hazards.

5.2.10.5 Depth of application and termination criteria

For each hazard identified, a functional safety goal is formulated, and the associated ASIL is determined. The HARA is considered complete, once all known hazard have been evaluated and covered by corresponding safety goals. If any additional hazards are identified in the course of development, they will need to be covered by additional safety goals.

5.2.10.6 Requirements for work, team and tools

In practice, the HARA is often documented as a table. In individual cases, specialized analysis tools are also utilized. Compared to other risk analysis methods, the time and effort required for HARA implementation can be categorized as low. Producing the analysis is typically the task of the safety officer. System, Hardware and Software Development assist the process by providing specialized input.

5.2.10.7 Benefits in the software context (advantages)

HARA helps with early detection of potential hazards the effects of which on system and software functions can be targeted in later phases of the development process. It creates a traceable connection between the identified hazards, the safety goals developed based on them and the corresponding software safety requirements. The risk analysis at the system level enables us to detect and assess safety-critical aspects at an early stage of the software architecture. As part of the safety case, the HARA also provides important input data and justifications for safety-related design decisions in the software context.

5.2.10.8 Limitations of the method in the software context (disadvantages)

In the software context, the HARA has certain methodological limitations. Risks that are not safety-related are not detected by HARA, nor can they be reasonably evaluated by this method. Moreover, depending on what is being developed, HARA implementation is often the responsibility of the customer, which can additionally restrict the depth of analysis and the influence on software development.

5.2.10.9 Result (output)

As the output of the HARA, a functional safety goal is formulated for each hazard identified. For each of these safety goals, an ASIL (Automotive Safety Integrity Level) is determined, indicating the required safety level. Safety goals are not actually specific technical solutions but rather requirements for system behavior – for example: “The system must prevent the battery from overcharging.”

5.2.10.10 References/Additional reading

International Organization for Standardization. ISO 26262: Road vehicles – Functional safety. Second edition, ISO 26262:2018, Geneva: ISO, 2018

5.2.11 HAZOP (Hazard and Operability Study)

5.2.11.1 Purpose of the method

The purpose of the HAZOP method is the preventive analysis of potential hazards and operational risks that can occur due to errors or malfunctions – especially in software functions. The study systematically checks whether or not appropriate controls and protective mechanisms are in place in order to prevent or at least mitigate the potential consequences of such deviations.

5.2.11.2 Basic procedure

The procedure comprises four sequential steps:

1. **Definition**
   First, the study is initiated, the scope of analysis and the goal are established, and the roles and responsibilities in the team are defined.

2. **Preparation**
   The study is planned, relevant data and documentation are gathered, and the guidewords necessary for the analysis as well as potential deviations are determined.

3. **Analysis**
   The system is divided into smaller subsystems. For each subsystem, the design expectations and relevant parameters are defined. Using the guidewords, potential deviations are identified, the causes and consequences of which are analyzed, and existing protective mechanisms and possible risk mitigation measures are set down.

4. **Documentation and traceability**
   The results of the analysis are systematically recorded, documented and prepared for further processing as well as follow-up.

5.2.11.3 Instructions/special considerations/differences in use in the software context

The HAZOP method can also be used effectively on software, provided that the underlying software functionality, its parameters and the potential consequences of deviations are well-understood at the system and software level. Only with an adequate understanding of the functional relationships can the method be applied effectively in order to identify potential risks and develop suitable risk mitigation measures.

5.2.11.4 Prerequisites (input)

For an effective implementation of the HAZOP study in the software context, there must be a function architecture (also called functional architecture or composition) or a detailed design of the analyzed functionality. Only if the structure and behavior of the software is adequately described is it possible to systematically identify and evaluate potential deviations.

5.2.11.5 Depth of application and termination criteria

HAZOP analysis on software should be performed based on a detailed software design. Only after the functional details, parameters and system behavior are described in full can the study be considered complete. The depth of application depends on the degree of functional complexity and the risk associated with the given software function.

5.2.11.6 Requirements for work, team and tools

Performing a HAZOP study on software takes a great deal of time and effort. It requires an interdisciplinary team with in-depth system and software know-how as well as the use of appropriate tools for structured analysis and documentation. Without methodological support and clearly defined roles, the method is difficult to implement effectively.

5.2.11.7 Benefits in the software context (advantages)

The HAZOP method offers special advantages when applied to software: It allows for a targeted examination of each individual parameter within a software function and helps the team perform structured brainstorming using guidewords. Potential fault conditions can thus be systematically identified, and appropriate risk mitigation measures and reaction mechanisms can be developed for the fault state.

5.2.11.8 Limitations of the method in the software context (disadvantages)

When applied to software, the HAZOP method comes up against certain limitations. Risks stemming from the interaction between different components can only be detected by this method to a limited extent. For such scenarios, it is more appropriate to use supplementary methods like event tree-analysis (ETA) or fault tree analysis (FTA).

HAZOP also cannot be used on its own for comprehensive risk identification in complex systems. Rather it always has to be used in combination with other suitable methods.

Another disadvantage is that HAZOP does not assess the risk itself but only identifies potential deviations and their consequences.

The success of the analysis greatly depends on the expertise of the moderator and the participating experts – both technically and with respect to the methodological procedure.

5.2.11.9 Result (output)

The output of a HAZOP study in the software context includes the systematic identification of potential functional faults, fault states and failures. In addition, the resulting consequences are used to develop mitigation measures in order to increase the robustness and safety of the software functionality.

5.2.11.10 References/Additional reading

IEC 61882:2016. Hazard and operability studies (HAZOP studies) – Application guide. 2nd Edition, International Electrotechnical Commission, Geneva, 2016. ISBN: 978-2-8322-3208-8

5.2.12 SOTIF (Safety of the Intended Functionality – ISO21448)

5.2.12.1 Objective of the method (or standard)

The objective of SOTIF is to identify and reduce safety risks stemming from the intended functionality of a system, i.e. in cases where there is no technical malfunction but where the system can nevertheless demonstrate unsafe behavior. This is especially relevant for systems with sensors or machine learning such as those used in today’s driver assistance systems.

SOTIF builds on classical functional safety as per ISO 26262 by focusing on risks that stem from inadequate perception, interpretation or decision-making despite the system technically working correctly. SOTIF is not exclusively limited to ASIL-classified systems. It is also relevant for applications that are not rated as safety-critical, especially if the intended functionality can lead to potentially unsafe behavior.

5.2.12.2 Basic procedure

The procedure comprises six steps:

1. Defining the intended functionality
   Clear description of what the system should be capable of, including all functions that could be relevant to safety.
2. Identifying potentially unsafe and unforeseeable scenarios
   Analysis of situations in which the system could demonstrate unsafe behavior despite working correctly (e.g. misinterpretation by sensors, inadequate perception of the environment, scenario coverage, unclear system boundaries.)
3. Assessing the risks
   Estimating the safety-critical effects of these scenarios – including without classic malfunction.

4. <u>Developing a risk mitigation action plan</u>
   Developing technical or functional measures in order to minimize the identified risks (e.g. redundancies, plausibility checks, limitations of the operating conditions.)
5. <u>Validation and verification</u>
   Proof that the measures are effective and the intended functionality is safely executed under real operating conditions.
6. <u>Iterative improvement</u>
   Continuously adjusting and expanding the analysis, especially in the case of learning systems or new use scenarios.

5.2.12.3 Instructions/special considerations/differences in use in the software context

In the software context, SOTIF can be applied to algorithms for environment perception and decision logic. Unlike classic failure modes (e.g. memory errors,) here, we are dealing with systematic uncertainties, e.g. due to insufficient training data, inaccurate sensor data interpretation or unknown boundary conditions. The challenge is to detect uncertainties despite the absence of any malfunction as such.

5.2.12.4 Prerequisites (input)

Defining the intended functionality

It must be clearly described what function(s) the system should perform – especially with regard to safety aspects.

System boundaries and operating conditions

The environmental conditions under which the software is operated must be known and specified (e.g. weather, lighting, traffic scenarios.)

Specifying the perception and decision logic

The software architecture must clearly present how information is processed and how decisions are made – especially in the case of ML-based functions.

Scenario database of potentially unsafe states

Typical and critical scenarios must be available in which an unsafe behavior can occur despite correct function.

Validation strategy

A concept must exist of how the safety of the intended functionality can be verified – e.g. through simulations or tests.

Differentiation from functional errors

The analysis must focus specifically on risks that are not caused by classical malfunctions but rather by uncertainties inherent to the system.

5.2.12.5 Depth of application and termination criteria

The application depth of a SOTIF analysis depends on the safety requirements and the complexity of the function under examination. The analysis is considered complete once unsafe scenarios have been identified and assessed, appropriate risk mitigation measures have been established, the remaining residual risk is categorized as acceptable and has been documented, and a validation by means of tests and simulations has been planned.

5.2.12.6 Requirements for work, team and tools

Time and effort
SOTIF implementation requires a great deal of methodological work. The identification of potentially unsafe scenarios without classical malfunctions is complicated and requires in-depth knowledge of the system.

Team
For a high-quality SOTIF analysis, an interdisciplinary team with broad expertise is essential. Know-how and experience in the following areas is required:

- Machine learning (ML)
- Functional safety and SOTIF
- Modeling of functional relationships
- Evaluation of realistic application scenarios

Tools
The use of appropriate tools supports the analysis but is not necessarily required. Useful tools:

- Modeling tools for mapping function architectures and scenarios (e.g. SysML, UML)

- Simulation environments for validation of safety-critical scenarios
- Test frameworks for AI/ML-functions for assessing uncertainties
- SOTIF-specific add-ons

5.2.12.7 Benefits in the software context (advantages)

- Extended safety analysis: SOTIF allows us to analyze risks that do not stem from classical software errors but from functional uncertainties – e.g. misinterpretation of sensor data, incomplete decision logic or undefined environmental conditions
- Preventive risk identification: The method supports early detection of potentially unsafe scenarios – including ones not preceded by malfunctions – and allows for a predictive safety assessment.
- Relevance beyond ISO 26262: SOTIF is not significant exclusively in the context of functional safety as per ISO 26262. For systems without an ASIL classification (e.g. comfort functions or AI-based assistance systems) SOTIF offers a valuable methodological framework for analysis and functional risk mitigation
- Strengthening of software architecture through targeted risk analysis: SOTIF supports the development of robust software solutions by specifically revealing uncertainties in the perception, interpretation and decision logic. This allows for weak points to be detected early and addressed systematically – regardless of the presence of classical malfunctions.
- Improvement of validation strategies: SOTIF promotes the development of targeted testing and validation methods, especially for complex or learning systems that operate in dynamic and unpredictable environments

5.2.12.8 Limitations of the method in the software context (disadvantages)

- No consideration of technical malfunctions: SOTIF focuses exclusively on risks stemming from the intended functionality – technical software errors such as memory access breaches or runtime errors are not taken into account.

- Lack of quantitative risk assessment: The standard does not offer any systematic prioritization or weighting of the identified risks. SOTIF does not include any quantitative risk analysis
- No direct derivation of risk mitigation measures: SOTIF identifies functional uncertainties but does not provide any specific risk management measures. Its implementation requires supplementary procedures like FMEA or FMEDA
- Complexity of scenario analysis: The identification of potentially unsafe scenarios (especially with ML-based functions) requires in-depth knowledge of the system and methodologically challenging
- Dependence on environmental assumptions: The risk analysis is largely based on defined operating conditions. Undetected or changed environments can limit the significance of the analysis
- Limited tool support and methodological maturity: Compared to established safety standards, the tool landscape for SOTIF is less developed. The application of the standard can be inconsistent and strongly depends on project-specific interpretation

5.2.12.9 Result (output)

The output of the application of the SOTIF standard consists of a systematic identification and documentation of risks that can arise from the intended functionality of a system – especially in cases where there is no technical malfunction but an unsafe behavior is still possible.

In the software context, SOTIF primarily provides:

- a structured description of potentially unsafe scenarios
- an assessment of the functional limits of perception and decision logic
- as well as a basis for developing supplementary safety measures by other procedures (e.g. FMEA)

Since SOTIF does not provide for any quantitative risk assessment, the emphasis is on qualitative insights that help to improve the software architecture and safeguard complex system functions.

5.2.12.10 References/Additional reading

International Organization for Standardization. *ISO 21448:2022 – Road vehicles — Safety of the intended functionality*. Geneva: ISO, 2022.

5.2.13 STPA (system-theoretic process analysis)

5.2.13.1 Objective of the method (or standard)

The STPA method (system-theoretic process analysis) is a system-theoretic approach to the risk analysis of complex, interconnected systems. Unlike classical failure analyses, STPA not only considers component failure but also faulty control actions, inadequate checking and systemic interactions. The objective is to identify unsafe control actions (UCAs) and use them to develop safety requirements that factor in functional safety as well as robustness und cybersecurity.

5.2.13.2 Basic procedure

The procedure is implemented in four steps:

1. Defining the analysis goal

- Defining which **losses or hazards** should be avoided (e.g. accident, malfunction, data loss.)
- Deriving from this: Which **unsafe control actions** could lead to these losses?

2. Modeling the control structure

- Representing the systems as a **control model** with:
  - Control units (e.g. software modules, control devices)
  - Control actions (e.g. braking command, lane assist activation)
  - Feedback (sensor readings, status information)
- Focus is on **interactions**, not only on component faults.

3. Identifying unsafe control actions

- Analysis of when a control action is **unsafe**:
  - Is **not performed**, although necessary.
  - Is **performed incorrectly** (too early, too late, to intense.)

    - Is **performed**, although not necessary.
    - Is **performed under wrong conditions**.

4. Defining safety requirements

- For every unsafe control action, **preventive requirements** are formulated.
- These requirements are integrated into the safety concept, the architecture and the test strategy.

5.2.13.3 Instructions/special considerations/differences in use in the software context

Using the STPA in the software context comes with specific strengths and special considerations. This method is especially well-adapted to control software, imbedded systems and automated functions. Unlike classical failure analyses, STPA does not only look at technical malfunctions but also faulty interactions and control losses within complex system architectures. As a result, it helps us develop preventive safety requirements that can take effect even before the occurrence of specific faults.

5.2.13.4 Prerequisites (input)

The implementation of an STPA is subject to several prerequisites: The analysis is based on a clearly defined objective. It is also necessary to have a basic understanding of the system and its control design, as well as access to existing requirements and safety goals. The analysis should be performed by an interdisciplinary team and supported by suitable tools so that the complexity of the relationships can be appropriately described.

5.2.13.5 Depth of application and termination criteria

The STPA method can be applied at different levels – from the system level to individual components, to interfaces. It is especially well-adapted to functions with complex control logic and external dependencies.

5.2.13.6 Requirements for work, team and tools

The effort required in order to perform an STPA is medium to high and depends largely on the complexity of the system as well as the number of the control paths involved. A well-founded analysis requires basic knowledge of the STPA methodology, system theory and the structure of control systems.

The analysis should be implemented by a multi-disciplinary team that combines expertise in software, functional safety, system architecture and IT security. The use of appropriate tools supports the systematic tracking and evaluation of the analysis results.

5.2.13.7 Benefits in the software context (advantages)

In the software context, STPA offers a comprehensive risk analysis that goes beyond the examination of technical failures. It allows for the identification of systemic weaknesses and unsafe control actions that can occur in complex software architectures. On this basis, it is possible to develop specific safety requirements and constraints. This method is particularly valuable due to its preventive effect: It can be used with unknown root causes and thus helps improve functional safety early on.

5.2.13.8 Limitations of the method in the software context (disadvantages)

Despite its strengths, the STPA does have a few limitations as well in the software context. This method requires a detailed understanding of the system as well as a structured modeling of the control logic. Risk mitigation measures are not developed automatically. They need to be written up and assessed manually. The documentation takes much longer than with classical methods. For a quantitative risk assessment, e.g. using a risk priority number (RPN,) it is necessary to additionally apply the FMEA.

5.2.13.9 Result (output)

The output of a STPA consists of the identification of unsafe control actions, the development of systemic loss scenarios and resulting safety requirements. These results are additionally used in order to improve the system design, safety concepts, and test strategies as well as to supplement classical methods like FMEA, ISO 26262, SOTIF and cybersecurity analyses.

5.2.13.10 References/Additional reading

Leveson, Nancy G.; Thomas, John P. (2018): *STPA Handbook – MIT STAMP-001*. Practical guidance for applying STPA in real-world projects, including control structures and UCA tables. Massachusetts Institute of Technology.

Abidi Nasri, S. (2018). *Application of STPA methodology to an automotive system in compliance with ISO 26262* (bachelor thesis, University of Stuttgart). Institute of Software Engineering, University of Stuttgart.

Abdulkhaleq, A., Wagner, S., Lammering, D., Boehmert, H., & Blueher, P. (2017). *Using STPA in Compliance with ISO 26262 for Developing a Safe Architecture for Fully Automated Vehicles*. In: Dencker, P., Klenk, H., Keller, H. B. & Plödereder, E. (Hrsg.), *Automotive – Safety & Security 2017*. Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn.

Kaiser, B. (2021). *Applying STPA in the Context of SOTIF for ADAS and Automated Vehicles*. ANSYS Germany.

Leveson, N. G. (2012). *Engineering a Safer World: Systems Thinking Applied to Safety*. Cambridge, MA: MIT Press. ISBN: 9780262016629. DOI: 10.7551/mitpress/8179.001.0001

5.2.14 SWIFT (Structured What-If Technique)

5.2.14.1 Objective of the method (or standard)

SWIFT was originally developed as a simpler alternative to HAZOP. This is a systematic, team-oriented method in which a series of “guidewords” or prompts are used by the moderator in a workshop to help participants identify risks. The moderator and the team use standardized “what if” questions in combination with the guidewords to analyze how a system, a plant, an organization or a process could be affected by deviations from normal operation and behavior.

5.2.14.2 Basic procedure

1. Before the analysis starts, the moderator prepares an appropriate list of key words or phrases (either based on a standard set or are specially created for this purpose) to allow for a comprehensive review of hazards or risks.
2. In the workshop, the external and internal context of the element, system, change or situation as well as the scope of analysis are discussed and coordinated.
3. The moderator asks participants to incorporate and discuss the following:
    - known risks and hazards;

- previous experiences and incidents;
- known and existing controls and protective measures;
- regulatory requirements and restrictions.

4. The discussion is launched with a “what if” question featuring a keyword or topic. The “what if” phrases used to formulate questions are: “What if...,” “What would happen if...,” “Could someone or something...,” “Has anyone or anything ever...” The objective is to get the team to examine potential scenarios and their causes, consequences and effects.
5. The risks are summarized, and the team examines the controls in place.
6. The description of the risk, its causes, consequences and the expected controls is coordinated with the team and documented.
7. The team checks if the controls are adequate and effective and agrees on a statement on the effectiveness of the risk control. If the risk control is less than satisfactory, additional risk management measures are discussed, and potential additional controls are defined.
8. During this discussion, more “What if” questions are asked in order to identify additional risks.

5.2.14.3 Instructions/special considerations/differences in use in the software context

The method requires adapting the key phrases to software-specific errors, e.g. “What if an interface did not work as specified?” or “What if an update were to fail?”

5.2.14.4 Prerequisites (input)

The implementation of an SWIFT analysis is subject to several prerequisites. First, we need a description of the system, module or process to be analyzed. In order to perform a well-founded analysis, we also need the existing requirements as well as the architecture or design documents. A vital role is also played by the use of guidewords or questions (e.g. “failure,” “delay” or “misuse”) that act as an impulse for the systematic identification of potential risks.

5.2.14.5 Depth of application and termination criteria

The application depth of the SWIFT analysis depends on the purpose of the analysis and the complexity of the system. It is typically performed at the system or subsystem level but can also be applied down to the software module level. As a criterion for completion, all the relevant “What if...?” questions for the areas analyzed must be answered and documented, and no more new risks must be identified.

5.2.14.6 Requirements for work, team and tools

The time and effort required for a SWIFT analysis is low to medium as compared to other methods like HAZOP. Since no set list of guidewords is used, the preparation and documentation work is greatly reduced. There are no particular requirements for the team or special tools. The method can be implemented flexibly and by simple means in interdisciplinary groups.

5.2.14.7 Benefits in the software context (advantages)

Thanks to its speed and simplicity, the SWIFT method is also efficiently utilized as part of regular review processes. It is especially suitable for dynamic development environments that require a fast, structured risk assessment.

5.2.14.8 Limitations of the method in the software context (disadvantages)

Compared to established procedures like FMEA or HAZOP, the SWIFT method is less systematic and does not go as in-depth the analysis. Its effectiveness greatly depends on the experience and specialized knowledge of the participating team. If this expertise is lacking, the team runs the risk of overlooking relevant scenarios. In addition, this method does not provide for any quantitative assessment of the identified risks, rendering objective prioritization difficult.

5.2.14.9 Result (output)

The application of the SWIFT method results in a documented overview of identified risks and weak points as well as suggestions for possible remedies. This output list forms the basis for further assessments and decisions in conjunction with risk management.

5.2.14.10 References/Additional reading

IEC/ISO 31010:2019 – Risk Management – Risk Assessment Techniques. International Organization for Standardization, Geneva.

5.2.15 TARA (Threat Analysis and Risk Assessment)

5.2.15.1 Objective of the method (or standard)

The purpose of the threat analysis and risk assessment (TARA) is to systematically identify and assess cyber-threats and develop appropriate risk management measures. In line with the assessment, the identified risks are analyzed with regard to the ease of a potential attack, the effects on operational safety, the security of personal data, the product availability as well as financial and reputation-related consequences. Based on this assessment, targeted countermeasures are defined in order to ensure an acceptable level of residual risk.

5.2.15.2 Basic procedure

1. Context definition: The starting point is the description of the system, its interfaces and its communication paths. Relevant functions, participating control units, networks and external interfaces (e.g. cloud, diagnostics, app connections) are identified in the process.
2. Identification of assets: Determining the assets to be protected, e.g. safety-related functions, communication messages, software components, cryptographic keys or personal data.
3. Threat analysis: Systematic identification of potential attack vectors and threats that can affect the identified assets. Known attack scenarios, weak points and potential attacker profiles are taken into account. The analysis can be assisted using threat databases, STRIDE or HEAVENS approaches.
4. Risk assessment: Assessment of the identified threats based on relevant criteria like
    a. difficulty of attack and technical feasibility,
    b. potential effects on operational safety, data protection, availability and finances,

   c. probability of occurrence, taking into account the attacker capabilities and motivation.
   d. The result is a prioritized list of risks.
   e. Decision on how to deal with the risks (avoid, reduce, transfer or accept the risk)
5. Defining cyber-security goals: For all risks that cannot be transferred or accepted, higher-level cyber safety goals are formulated in order to reduce the risk to an acceptable level.
6. Developing technical safety requirements: Based on the cyber-safety goals, specific technical requirements are developed for the architecture, implementation and operation of the system (e.g. authentication, encryption, access control.)
7. Validation and traceability: Ensuring that all risks are addressed by corresponding measures and the requirements developed based on the risks are traceably implemented in the system and software development.
8. Continuous updates: The TARA is not a one-time process. New threats, weak points or system changes require the analysis to be continuously reviewed and adjusted in order to ensure that the residual risk remains at an acceptable level long-term. The TARA especially has to be reviewed in case of software updates, architectural changes or the introduction of new communication interfaces.

5.2.15.3 Instructions/special considerations/differences in use in the software context

The TARA has a clear focus on software-based threat scenarios and is closely tied to safety-related development practices like secure coding, vulnerability management and the patch and update processes. Its application in the software context allows for a targeted assessment of risks specifically resulting from weak points in digital components and helps with the development of technical and organizational measures for safeguarding software-heavy systems.

5.2.15.4 Prerequisites (input)

The foundation for TARA implementation is a complete item definition that describes the system or function being assessed. There should also be a HARA (Hazard Analysis and Risk Assessment) already available as well as an evaluation of data protection aspects (data privacy evaluation) in order to allow for a well-founded and context-specific risk analysis.

5.2.15.5 Depth of application and termination criteria

For all identified risks that cannot be classed as acceptable, cyber safety goals and, if necessary, technical safety requirements are developed. Since threat scenarios in the digital environment occur dynamically and are continuously subject to change, the TARA should be seen as a "living analysis." Its application is therefore continuous and iterative. It is never intended to be fully terminated. Instead its completion is based on individual analysis cycles in each project phase.

5.2.15.6 Requirements for work, team and tools

The TARA is often documented as a table. In some cases, specialized analysis tools are used. These tools help with the systematic assessment using graphic representations, e.g. as attack trees. The time and effort required for TARA implementation largely depends on the complexity of the system being analyzed as well as the availability and quality of existing threat databases. In particular, the identification of new, previously undocumented risks can significantly increase the amount of work required for the analysis. A solid implementation requires close collaboration with experts from system and software development as well as the testing department.

5.2.15.7 Benefits in the software context (advantages)

The TARA method allows for a structured protection requirement analysis across various types of digital assets. This includes functional aspects like safety-related system functions (e.g. airbag activation,) critical notifications like warnings or control commands (e.g. apply parking brake,) sensitive data like personal user data or access information as well as software-based components, including cryptographic keys. This wide-ranging applicability enables TARA to assist with the targeted identification and assessment of risks in software-heavy systems and create a foundation for effective protective measures.

5.2.15.8 Limitations of the method in the software context (disadvantages)

The quality of the TARA results strongly depends on the experience and expertise of the team as well as the completeness and up-to-datedness of the threat database. Without a targeted focus on realistic or especially critical scenarios, there is a chance that the analysis could become unnecessarily bloated or overly complex and confusing. Clear prioritization and structuring is therefore essential in order to guarantee the reliability and efficiency of the method.

5.2.15.9 Result (output)

For all risks that are categorized as unacceptable in the TARA, specific cybersecurity goals must be defined. These goals can then be used to develop technical safety requirements that contribute to risk management. The results of the TARA can directly affect functional safety and should therefore be coordinated with the HARA. Missing hazards arising from the cybersecurity perspective also need to be added to the HARA. The two methods should be seen as closely meshed, since safety gaps (e.g. due to targeted tampering) can also result in functional hazards such as the unintended activation or deactivation of safety-critical vehicle functions.

5.2.15.10 References/Additional reading

ISO/SAE 21434:2021 – Road Vehicles – Cybersecurity Engineering. International Organization for Standardization (ISO) und SAE International, Geneva.

6 Risk analysis along the integration chain

6.1 Introduction

Risk analyses can be performed in parallel along the integration chain (currently ongoing project) or sequentially for subsequent projects.
In light of increasingly interlinked of supply chains and rising complexity as well as the re-use of software modules or entire systems, the following aspects must be considered:

- Integration of risk analyses in ongoing projects
- Incorporation in new projects
- Sharing between different companies in the supply chain.

6.2 Objective

The objective is to develop recommendations for how identified risks can be systematically assessed and then integrated into adjacent risk analyses.

6.3 Motivation and advantages

- Increased efficiency: Reduction of time, effort and cost by avoiding redundant analyses in case of similar functions or systems.
- Quality improvement: Use of proven analyses and insights from existing projects for early, comprehensive risk identification and assessment
- Acceleration of development: Faster development times thanks to the availability of preliminary analyses which allow for targeted prioritization and efficiently shorten development cycles.
- Consistency: Ensuring consistent consideration of risks across project and company boundaries, especially for standard functions.

6.4 Challenges

- Context dependency: The risks classified and the effectiveness of safeguards strongly depend the specific project context (hardware, environmental conditions, overall architecture, application case, legal/normative requirements.)

- Differing methods and depths of analysis: Different companies or projects may utilize different methods, tool, evaluation criteria and degrees of detail for risk analysis.
- Data quality and up-to-dateness: The quality, completeness and up-to-dateness of the risk analysis must be guaranteed for the specific project
- Basis of information: The risk analysis to be incorporated must be prepared in an agreed format and with an adequate degree of detail.
- Interfaces and dependencies: Identifying the interfaces and dependencies of the software functionalities in the (new) system context.
- Confidentiality and IP protection: Information in risk analyses can be subject to intellectual property protection, making it difficult to exchange.
- Tool compatibility: Lack of compatibility of risk analysis tools between the participants can hamper data exchange.
- Traceability: The seamless traceability of adjustments and amendments to the risk analysis must be ensured.

6.5 Incorporating risk analyses

Risk analyses can be integrated into ongoing/new projects or shared between different companies in the supply chain. To this end, we recommend the following phases:

Phase 1: Request and review of the input documentation

- **Specification and request:** Specifying the requirements for the "risk analysis output" to be provided, e.g.:
    - Defining the purpose and scope of the risk analysis to be incorporated
    - Coordinating format, structure and assessment method
    - Determining the interfaces, contact persons and communication paths and responsibilities for the risks.
    - Planning review meetings and approval processes
    - List of "assumptions of use"
    - Output of the analysis (e.g. risks identified, safety requirements developed)

- **Formal and methodical review (plausibility check)**
  The objective is to ensure that the submitted risk analysis is complete and formal. The activities involved are:
    - Incoming inspection of the delivered documents (completeness, version, approval status)
    - Checking for agreed format and degree of detail
        - Description of system boundaries and interfaces
        - Consistency: Is the documentation free of contradictions?
        - Compliance with method: Was the agreed method (e.g. FMEA as per AIAG&VDA FMEA manual) correctly applied?
        - Traceability: Is the deduction of causes, consequences and actions logically documented?

Note: In case of discrepancies, clarification loops must be implemented with the integration partners

Phase 2: Technical assessment of the delivered analysis in the project context

- The degree of detail of the analysis and its incorporation are based on the criticality of the component (e.g. ASIL classification,) the novelty of the technology and the degree of maturity (e.g. assessments according to Automotive SPICE®.)

- Context comparison (system level): Comparison of scope of function, architecture, interfaces, environmental conditions as well as application cases and legal and normative requirements, etc.

    - *Example of environmental conditions: a sensor provides stable results up to a maximum operating temperature of 85°C. In the vehicle, however, the sensor is installed in an area that can reach up to 100°C. This discrepancy (or “gap”) must be evaluated as a new or modified risk.*

    - *Example of interfaces: A control unit sends data. The supplier’s risk analysis covers the failure of the transmission process. However, the customer has to analyze what happens if control units of other makes transmit faulty data on the bus into the integration chain and a component receives this data.*

Phase 3: Integration, consolidation and feedback
This phase consists of three steps:

1. **Integrating the risk analyses:**
    - **Incorporating validated risks:** Risks categorized as valid are incorporated in the customer’s overall risk analysis with traceability.
    - **Handling Delta risks:** New and modified risks from the Delta analysis are integrated and assessed in the overall analysis.
2. **Action planning and feedback:** The consolidated analysis is used to develop an action plan. New requirements that affect the scope of supply are relayed to the suppliers (e.g. “sensor must be specified up to 100°C,”) which can lead to contract modifications, among other things.
3. **Documentation of incorporation:** The entire process is documented in an integration report as proof (e.g. for a safety case, assessment/audit.)

Phase 4: Verification and validation
The consolidated risk analysis and the measures developed based on its results must be verified and validated.
The format and contents of the consolidated risk analysis and the associated action plan are approved (through reviews, etc.) in the target project.

6.6 Practical recommendations

- Early integration: Early incorporation of the risk analysis in the project planning and consideration in contract negotiations.
- Information transfer: Training and exchange between the teams in order to gain a shared understanding of the analyses and contexts, including lessons learned.
- Iterative process: Incorporation with feedback loops between the participants of the integration chain
- Standards: Check if suitable, existing standards can be used for the risk analysis before new ones are developed.

7 Annex – example cases

The example cases presented in this publication are intended for illustration and reference purposes only. They do not constitute binding recommendations for action or normative specifications. The German Association of the Automotive Industry (VDA) accepts no liability for the up-to-datedness, accuracy, completeness or applicability of the presented contents in specific operational situations.

Users are exclusively responsible for the assessment, selection and implementation of appropriate measures. The VDA cannot be held liable for damages resulting from the use or non-use of the example cases.

7.1 Practical examples for the application of the methods

7.1.1 ATAM – lane assist

System: lane assist
The lane assist feature is a safety-related software function in today's driver assistance systems. It detects lane markings using a front camera. If the driver inadvertently drifts from the lane, it assists by intervening in the steering or warning the driver. This function closely interacts with other system like the steering control unit, camera system, brake control and stability control.

Phase 0:
An interdisciplinary team of software architects, developers, safety experts and representatives from Quality Management is formed for the assessment of the lane assist. The relevant architectural documents are checked, and workshops are scheduled.

Phase1:

- Step 1: Explain the ATAM procedure
- Step 2: Identify business goals and most important system functions
    - Accident prevention using lane assist
    - Enhanced comfort (gentle steering interventions, verbal and visual warnings)
    - Fulfillment of legal requirements
- Step 3: Present architecture
- Step 4: Identify architectural approaches

    - Layered architecture: Separate into sensor, logic and actor layers
    - Pipes-and-filters pattern: Gradual filtration of camera and sensor data
    - Feedback control loop: Closed control loop for steering correction
    - Event-driven architecture: Event-driven response to lane deviation
    - Fault containment pattern: Isolated error control in subsystems
    - Watchdog pattern: Monitoring of critical components with automatic failure detection
    - Observer pattern: Observation and notification in case of system state changes

- Step 5: Create utility tree (quality attributes and their scenarios are defined and prioritized)
  In step 5, a utility tree is created as a structured representation of the most important quality attributes and their scenarios. In collaboration with the stakeholders, these scenarios are then rated on a High-Medium-Low (H/M/L) scale based on their importance and severity or difficulty/risk.
  The critical scenarios with high importance and high severity are thus identified and prioritized for further analysis.

Table 7-1: utility tree

Safety	**Steering intervention reaction time < 50 ms**
	Gentle steering correction without oversteering < 0.5 m/s²
	Camera failure warning within 100 ms
Performance	Image data processing < 50 ms latency
	Processing rate > 20 frames/second
...	...

- Step 6: For each high-ranking scenario, risks and tradeoffs are analyzed

Example: Warning in case of camera failure within 100 ms

- Architectural approach: Fault containment pattern with watchdog monitoring
- Analysis: A watchdog continuously monitors the data stream from the camera. In case of failure, a fallback mode is immediately activated and warns the driver without intervening.
- Risk: False positive failure detections could lead to unnecessary system deactivations and impact the driver's trust in the system.
- Trade-off: Robustness vs. availability – stricter monitoring increases safety but can often lead to false alarms.
- Sensitivity point: Timeout-thresholds of the watchdog
- Recommendation: Multi-level monitoring with plausibility check over multiple frames

Phase 2:
Here, Steps 5 and 6 are repeated with additional stakeholders in order to incorporate additional perspectives on the architecture. Afterwards, the identified risks are summarized and compared to the business goals for reference.

Phase 3:
Risks, trade-offs and recommendations are summarized

Conclusion
The ATAM analysis of the lane assist feature reveals that the combination of event-driven architecture, pipes-and-filters, feedback control loop, layered architecture and fault containment with watchdog monitoring does fulfill primary quality goals, but it also results in specific risks and tradeoffs. Adjustments such as the prioritization of safety-critical events, adaptive filter pipelines and multi-level monitoring reduce latencies, increase accuracy and minimize false alarms. The results produce a clear basis for effective architectural and software improvements and tests in the safety-critical environment.

Note: For construction site situations, it is recommended to perform a supplementary technical risk analysis with SOTIF.

7.1.2 ETA – OTA update

Scenario: OTA update for a safety-critical control unit

The goal is to analyze the possible events that can occur after the start of an OTA update for a safety-critical control unit – for example, a brake system.

Initial event: OTA update is started.

Event tree

1. **Update file correctly downloaded?**
    - **Yes** → Move on to Step 2.
    - **No** → **Consequence:** Update cancelled, vehicle keeps old software version (effect: no new function but safe.)
2. **Integrity check (checksum) successful?**
    - **Yes** → Move on to Step 3.
    - **No** → **Consequence:** Update is discarded, vehicle keeps the old software version (effect: safe but no update.)
3. **Installation successful?**
    - **Yes** → Move on to Step 4.
    - **No** → **Consequence:** Control unit possibly in inconsistent state → **Risk: Vehicle not ready to drive**.
4. **Restart of control unit successful?**
    - **Yes** → Update complete, vehicle operational.
    - **No** → Control unit fails → **Risk: safety-critical failure (e.g. brakes)**.

Event tree (text diagram)

```
 Start OTA-Update (P=1.0)
   |
   +------ Download OK?          (P=0.99)   ------> Next
   |               |
   |               +------ No          (P=0.01)   ------> Abort (safe)
   |
   +------ Integrity check OK? (P=0.999) ------> Next
   |               |
   |               +------ No          (P=0.001) ------> Abort (safe)
   |
   +------ Installation OK?          (P=0.995) ------> Next
   |               |
   |               +------ No          (P=0.005) ------> Inconsistency (risk)
   |
   +------ Restart OK?               (P=0.999) ------> Success
                   |
                   +------ No        (P=0.001) ------> Control unit failure (critical)
```

Figure 7-1: event tree (text diagram)

Probabilities and consequences

Each potential event within a decision tree can be assigned a probability of occurrence – for example 0.01 for a download error or 0.001 for an integrity error. The resulting consequences are evaluated according to their effects, e.g. with the categories “critical,” “safe” or “functionally limited.”

Table 7-2: ETA table: OTA update for control unit

Step/event	Condition	Probability	Consequence
Start OTA-Update	-	1.0	Starting point
Download successful?	Yes	0.99	Next
	No	0.01	Abort (safe)
Integrity check OK?	Yes	0.999	Next
	No	0.001	Abort (safe)
Installation successful?	Yes	0.995	Next
	No	0.005	Inconsistency (risk: vehicle not ready to drive)
Restart successful?	Yes	0.999	Success
	No	0.001	Control unit failure (critical)

Calculating the overall probabilities

- **Successful update:**
  P=0.99×0.999×0.995×0.999≈0.983P = 0.99 \times 0.999 \times 0.995 \times 0.999 \approx 0.983P=0.99×0.999×0.995×0.999≈0.983
  → **98.3%**

- **Critical failure (restart failed):**
  P=0.99×0.999×0.995×0.001≈0.00099P = 0.99 \times 0.999 \times 0.995 \times 0.001 \approx 0.00099P=0.99×0.999×0.995×0.001≈0.00099
  → **0.099%**

- **Inconsistency after installation:**
  P=0.99×0.999×0.005≈0.00495P = 0.99 \times 0.999 \times 0.005 \approx 0.00495P=0.99×0.999×0.005≈0.00495
  → **0.495%**

Conclusion

The event tree analysis (ETA) for the OTA update of a safety-critical control unit shows that the update process overall has a very high probability of success – 98.3%. However, the analysis also identified potential risks: A critical failure of the control unit following restart is rare (0.099%) but can have serious safety-related consequences. There is also a medium risk of inconsistencies following installation (0.495%) that impact vehicle availability.

The ETA reveals that the biggest risks are not in the download or the integrity check but in the last steps of the installation and the restart. This indicates the need to strengthen diagnostic mechanisms and fallback strategies, especially for these phases.

Overall, the ETA provides a transparent basis for the prioritization of risk mitigation measures and supports the argument in safety and quality audits.

7.1.3 CPA – traffic sign recognition

Traffic sign recognition is a key software functionality in today's driver assistance systems and automatic driving functions. It is used to recognize and interpret traffic signs like speed limits, "no passing" signs or road work signs. The correct processing of this information is essential for compliance with regulations and safety in traffic.

The provided CPA template (Criticality and Prioritization Analysis (CPA)) allows for a structured assessment of this functionality. This is a qualitative evaluation based on expert assessments and can be supplemented with quantitative methods. The results serve as a basis for architectural decisions, test planning and further analyses like FMEA or threat modeling.

CPA criterion	**Risk identification**	**Criticality**	**Priority of safeguarding**
Software functionality	Traffic sign recognition	High	High
System dependency	Speed assist, navigation system	Medium	High
Use context	Expressway, city, country road, construction sites	High	High
Criticality	Misinterpretation can cause speeding violations	High	High
Robustness requirement	Recognition in case of poor lighting, dirt, weather	High	High
Cybersecurity relevance	Tampering with camera data or software possible	Medium	Medium

Redundancy/safeguarding	Combination with map data, V2X-communication	Medium	High
Error tolerance/recovery	Warning of driver, temporary deactivation of the function	Medium	Medium
Testing requirement/scenarios	Recognition of temporary signs, construction sites, night drives	High	High
Standard reference (optional)	ISO 26262 ASIL B/C, UNECE R155	Medium	High
Assessment scale (optional)	Criticality: high / Priority: high	High	High

Conclusion

The assessment of the traffic sign recognition indicates a high criticality and priority in nearly all relevant areas. In particular, the effects in case of misinterpretations and the robustness and test coverage requirements stress the safety significance of this function. It is recommended to reinforce the safeguarding with redundant information sources like map data and V2X communication and define targeted test scenarios for temporary and hard-to-recognize traffic signs. The results of this CPA analysis should be integrated in the safety architecture and test planning.

7.1.4 HAZOP – door opener app

The door opener mobile app enables the user to open and close the vehicle using their smartphone. The app sends an opening signal to the vehicle over a secure connection (e.g. Bluetooth, NFC or cellphone signal,) after which the user is authenticated. The app thus takes the place of a conventional car key and offers additional comfort functions like status display, remote control and, if necessary, access logging.

This is a qualitative evaluation based on expert assessments. The results serve as a basis for architectural decisions, test planning and further analyses like FMEA or threat modeling.

System: Mobilephone-to-car app, transmission of the opening signal

Guide-word	**Deviation**	**Cause**	**Consequence**	**Action**
None	No opening signal sent	App crash, network error	Vehicle stays locked	Check app stability and connection
More	Multiple opening signal	Double-click, software error	Door opens more than once, malfunction	Signal debouncing, logging
Less	Opening signal weak	Bluetooth faulty, bandwidth	Vehicle does not respond	Check signal strength, alternative paths
Wrong	Wrong signal sent	Software bug, tampering	Unauthorized opening, safety risk	Authentication, encryption
Delayed	Opening signal arrives late	Network latency, app performance	Door opens with delay	Timeout, user notification

Conclusion
The HAZOP analysis of the door opener mobile app has shown that various deviations and error sources (e.g. connection problems, lack of authentication or software errors) can affect the function and safety of the vehicle opening operation. Through the systematic consideration of guidewords and scenarios, it was possible to identify risks and develop corresponding risk

mitigation measures. The analysis underlines the importance of stable communication, reliable authentication and robust error control in order to ensure both the user friendliness and the safety of the app.

7.2 Method selection in a safety-related project

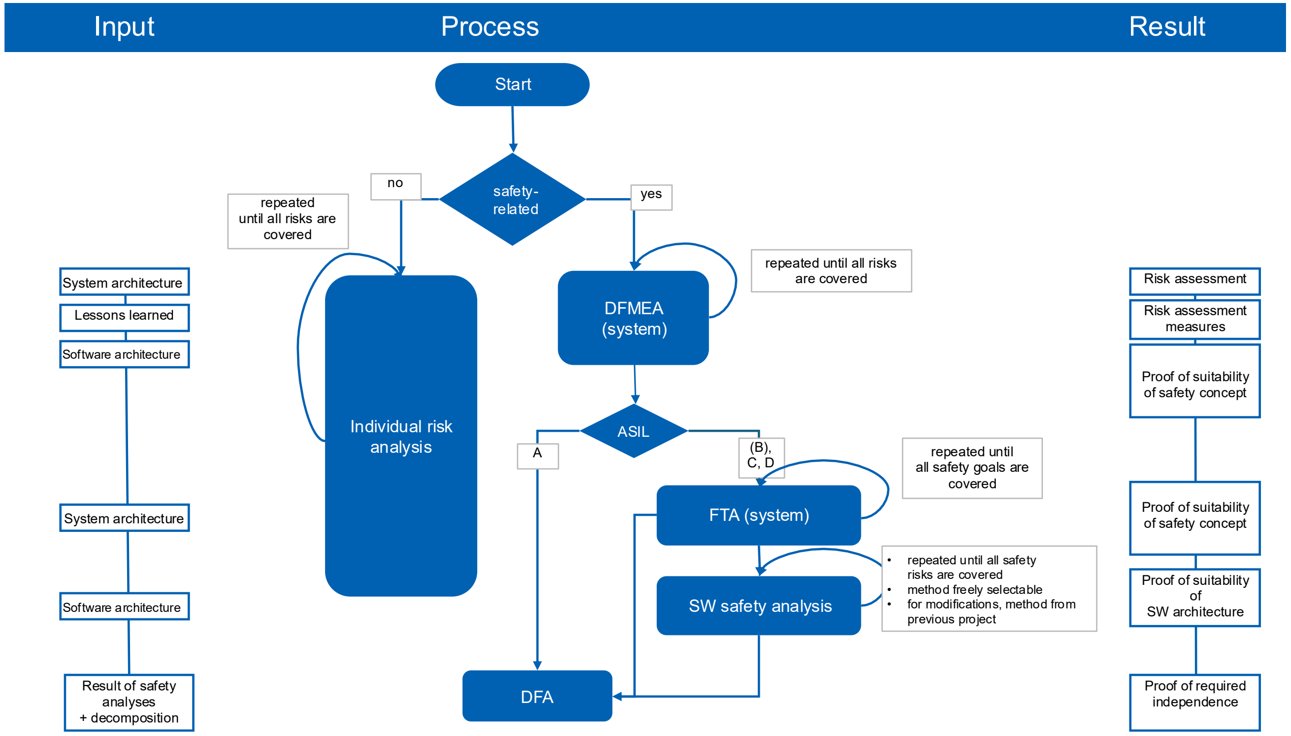

Input
Process
Result
Start
safety-related
no
yes
repeated until all risks are covered
Individual risk analysis
DFMEA (system)
repeated until all risks are covered
ASIL
A
(B), C, D
repeated until all safety goals are covered
FTA (system)
SW safety analysis
• repeated until all safety risks are covered
• method freely selectable
• for modifications, method from previous project
DFA
System architecture
Lessons learned
Software architecture
System architecture
Software architecture
Result of safety analyses + decomposition
Risk assessment
Risk assessment measures
Proof of suitability of safety concept
Proof of suitability of safety concept
Proof of suitability of SW architecture
Proof of required independence

Figure 7-2: method selection process in a safety-related project

The process sequence depicted consists of three main areas – input, process and output and describes the structured procedure for risk analysis and safety assessment in line with system development.
The process starts at the starting point, where it is checked whether the given system or element is safety-related. If it is not, an individual risk analysis (see Chapter 5) is performed iteratively until all risks have been appropriately assessed and addressed.
If the system is safety-related, a DFMEA (Design Failure Mode and Effects Analysis) is performed at the system level. This analysis is used to identify potential root causes and their effects in the overall system.
In case of non-safety-related risks, the DFMEA is typically still performed at system level. Risks from software-based functionalities could be analyzed using DFMEA, and appropriate detection or avoidance measures could be developed based on the results.

Depending on the classification of the Automotive Safety Integrity Level (ASIL,) ISO 26262 requires further risk analyses – e.g. for higher ASIL ratings (B, C or D):

- An FTA (Fault Tree Analysis) at the system level for structured analysis of root causes.
- A software safety analysis that focuses on the safety aspects of the software architecture.

Remark: The hardware safety analyses are not listed.

If all safety-related measures are successfully implemented, the results der DFMEA, FTA and SW safety analysis provide the proof for the suitability of the safety concept or the software architecture. The final DFA – independent of ASIL – should ensure the necessary independence according to the safety requirements.

Notes on method application and optimization:

Failure modes should be logically clustered, e.g. using guidewords.

- Example: The differentiation between driver, passenger and side airbag is functionally different, but the signal flow is equivalent. The failure modes can therefore be combined.

- Equivalent failure types like data overwriting should likewise be grouped together in order to avoid redundancies.

Lessons learned from current use phase (especially from the DFMEA) should be actively incorporated into new projects. For this reason, a methodological allocation for the use phase is also sensible and necessary.

8 Annex – applicability of the methods (overview)

This overview describes additional risk analysis methods and their suitability in addition to the methods listed in this volume (identified by a reference to the preceding chapter.)

Method (standard)	Applicability to technical risk analysis of software-based functionalities	Standard/ reference	Description
ATAM (Architecture Tradeoff Analysis Method) Chapter 5.2.1	✅ Suitable	SEI	Evaluates architectural decisions with regard to quality and safety risks; useful for software architectures but not classical risk analysis.
CCA (common cause analysis) Chapter 5.2.2	🔸 Applicable to a limited extent	ISO 26262-5/6, IEC 60812	Identification of common causes of failures in redundant systems; e.g. common logic faults across modules.
CPA (Criticality and Priority Assessment) Chapter 5.2.3	✅ Suitable	NIST Interagency Report 8179	Structured method of evaluating software functionalities with regard to their criticality in the overall system.
CPA (Critical Path Analysis)	❌ Not suitable	PMBOK, DIN 69901	Project management method for scheduling, not for risk analysis of software.

Method (standard)	**Applicability to technical risk analysis of software-based functionalities**	**Standard/ reference**	**Description**
DFA (Design Failure Analysis / Design for Assembly)	◆ Applicable to a limited extent	AIAG/VDA	Originally for mechanical components; only indirectly applicable for software-based functionalities, e.g. with modular software architecture or interface integration.
DRBFM (Design Review Based on Failure Mode) Chapter 5.2.4	✅ Suitable	AIAG/VDA	Focus on changes in design or software modules in order to identify new risks early; very useful for software changes.
ETA (Event Tree Analysis) Chapter 5.2.5	◆ Applicable to a limited extent	ISO 26262-5, IEC 61511	Deductive method of analyzing event consequences; can be used for software in conjunction with safety measures but limited applicability to logic errors.
FMEA (Failure Mode and Effects Analysis) Chapter 5.2.6	✅ Suitable	AIAG-VDA FMEA, ISO 26262-5/6	Inductive, structured method for systematic detection of failure types, causes and effects, including for software-based functionalities. Software FMEA not standardized as term.

Method (standard)	**Applicability to technical risk analysis of software-based functionalities**	**Standard/ reference**	**Description**
FMEA-MSR (Monitoring and System Response) Chapter 5.2.7	✅ Suitable	ISO 26262-5	Expansion of the FMEA for software and system monitoring, analysis of diagnostic and reaction mechanisms for risk mitigation.
FMEDA (Failure Modes, Effects and Diagnostic Analysis) Chapter 5.2.8	🔸 Applicable to a limited extent	ISO 26262-5	Quantitative analysis for hardware components only; not as suitable for software.
FSA (Functional Safety Assessment)	❌ Not suitable	ISO 26262-2	Audit to check safety processes, not a risk identification method.
FTA (Fault Tree Analysis) Chapter 5.2.9	🔸 Applicable to a limited extent	ISO 26262-5, IEC 61025	Deductive root cause analysis, good for finding root causes, less suitable for structured risk identification in software.
HARA (Hazard Analysis and Risk Assessment) Chapter 5.2.10	🔸 Applicable to a limited extent	ISO 26262-3	Systematic identification and assessment of hazards at the system level. Software can be a cause, focus is on functional safety.

Method (standard)	Applicability to technical risk analysis of software-based functionalities	Standard/ reference	Description
HAZOP (Hazard and Operability Study) Chapter 5.2.11	✅ Suitable	IEC 61882	Systematic risk analysis of process and function deviations; adaptable for software-based functionalities.
ORA (Operational Risk Assessment)	🔸 Applicable to a limited extent	UNECE R156, ISO/SAE 21434	Assessment of operational risks, e.g. in operation, partially applicable to software.
PHA (Preliminary Hazard Analysis)	🔸 Applicable to a limited extent	ISO 12100, ISO 26262-3	Qualitative, early hazard analysis, also appropriate for software functions, for early detection of risks.
PRA (Process Risk Assessment)	❌ Not suitable	IATF 16949, ISO 9001	Evaluation of production and process risks.
Reliability Prediction / Weibull Analysis	❌ Not suitable	IEC 61709, MIL-HDBK-217	Statistical methods of hardware reliability prediction; not applicable to software.
Safety Case / Safety Argumentation	❌ Not suitable as technical risk analysis method	ISO 26262-10, UL 4600	Documentation and proof of functional safety; not a standalone risk identification method.

Method (standard)	**Applicability to technical risk analysis of software-based functionalities**	**Standard/ reference**	**Description**
SOTIF (Safety Of The Intended Functionality) Chapter 5.2.12	✅ Suitable	ISO 21448	Analysis of risks from non-functional safety or system behavior in the intended mode of operation; especially relevant for autonomous systems and software.
SRA (Security Risk Assessment)	✅ Suitable (cybersecurity)	UNECE R155, ISO/SAE 21434	Evaluation of cybersecurity risks in vehicle software and systems.
SWIFT (Structured What-If Technique) Chapter 5.2.13	✅ Suitable	AIAG/VDA	Qualitative method for identification of potential faults or risks; also applicable to software; more exploratory.
STPA (system-theoretic process analysis) Chapter 5.2.14	✅ Suitable	Leveson, MIT	Safety analysis at the system level based on control and feedback theory; can comprehensively identify risks for software and systems.
TARA (Threat Analysis and Risk Assessment) Chapter 5.2.15	✅ Suitable (cybersecurity)	ISO/SAE 21434	Structured identification and assessment of threats and weak points in software and IT-systems.

✅ **Comments/notes on the table:**

1. **Limited applicability (🔶)** = Method is more deductive or hardware-related but can partly be applied to software or interfaces.
2. **Suitable (✅)** = Method is explicitly intended for or well-suited to software-risk analysis.
3. **Not suitable (❌)** = Method covers other areas (hardware, process, audit); not for software risk analysis.