Adam SCHNELLBACH, MSc

Fail-operational automotive systems

Doctoral Thesis

Graz University of Technology

Institute of Automotive Engineering
Head: Univ.-Prof. Dr. Peter FISCHER

Supervisor: Assoc.-Prof. Dr. Mario HIRZ

Graz, November 2016

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly marked all material, which has been quoted either literally or by content from the used sources.

Graz, _____ _____

Date Signature

Eidesstattliche Erklärung[1]

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Graz, am _____ _____

Datum Unterschrift

[1] Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008; Genehmigung des Senates am 1.12.2008

Acknowledgements

Foremost, I would like to express my gratitude to my supervisor, Assoc.-Prof. Dr. Mario Hirz. His profound knowledge, his critical view on my thesis and his continuous support were essential. His guidance helped me to stay focused and motivated, even in the toughest times in the past two years.

My sincere thanks goes to my superiors at Magna Powertrain, especially to Andreas Münzer. His support and encouragement was one of the triggers to start my PhD studies, and to keep on working on my thesis even if challenges at work were overwhelming.

Last but not least, I would like to thank my wife and children for supporting me throughout writing this thesis.

I don’t believe in talent. I believe in passion and work.

GEORGE KOLLIAS

Abstract

Due to historic and cost efficiency reasons E/E automotive systems rely on so called fail safe architectures. These systems are switching into an inactive safe state when detecting internal or external failure modes that require this reaction. Their safety mechanisms are designed to detect failures and to trigger a system shut down, to prevent a safety critical failure propagation. Advantages of these systems are their simple architecture and hence their low piece price, by providing the appropriate level of safety integrity. The disadvantage of these architectures is obviously their limited availability in case of internal or external failures. The expectations, behaviour and skills of today's drivers' are changing. The non-availability of systems leading to significant overall vehicle function degradation cannot be tolerated to the same extent as it was in the past. Due to these reasons the demand on fail operational systems is increasing. These are able to maintain the full or an acceptably limited function in case of internal or external failures, instead of just shutting down.

Since other domains (e.g. avionics) have gathered lots of experience with such topologies, the key challenge is to understand, carry over and tailor already existing technical solutions and design principles.

In this thesis, the current state of the art of automotive systems in various transport-related industrial domains is investigated, focusing on their related functional safety standards and technical solutions. Based on that, a detailed investigation of the ISO 26262:2011 is performed, from the perspective of fail-operational systems. The applicability of this standard is investigated, and improvement proposals are defined, wherever necessary. Besides that, typical fail-operational SW and HW architectures are investigated within the framework of the ISO 26262:2011 and an automotive application. The key finding of this investigation is that for automotive fail-operational systems, full redundancy is seldom required. The designers have to be capable of adding the right type of redundancy at the right spot in the architecture. To support the definition of these sufficiently independent redundant architectures, a generic, structured dependent failure analysis, fitting into the analysis framework of the ISO 26262:2011 is developed. Based on these findings, the redundancy allocation problem – a well-known problem in other industrial domains – is solved for an automotive application. In order to do so, a complex mathematical model is derived, to host failure-propagation modelling, and the calculation of the HW metrics of the ISO 26262:2011 as reliability metric. A straightforward genetic algorithm is developed to utilise this mathematical model. The verification of these models and algorithms is performed using realistic examples.

Contents

List of Figures

1 Introduction

1.1 Introduction of the problem and motivation of this thesis

The number, complexity, and criticality of automotive systems is rapidly increasing. The increasing demand for new functionalities, realised by a complex network of mechatronic systems, drives the automotive industry into previously unknown territory. The trend towards autonomous driving and alternative propulsion technologies, and the resulting new requirements are one of the key challenges of the next decades. Concepts, previously undoubted, need to be re-evaluated.

Current automotive systems can deactivate themselves in case of failures, showing a so called fail-safe behaviour. Related architectures are therefore directed to be able to detect failures and switch into a passive safe state within a sufficiently short time. These topologies are no longer applicable, if the availability of these systems is safety critical. Mainly due to the trend towards automated and autonomous driving, passive safe states are no longer applicable for systems involved in these functions. Fail-safe has to be replaced by fail-operational behaviour, providing crucial functionalities even in the presence of failures.

This challenge may be new to the automotive industry, but not for other industrial domains. As most obvious example, air- and spacecraft have been applying fail-operational Electric-Electronic (E/E) systems for decades. Methods and designs cannot be carried over into passenger vehicles, without adaptations, though. The requirements and constraints (e.g. regarding cost, weight and packaging space) are completely different in case of a passenger car, than in a spacecraft. Hence, great care has to be taken, when evaluating the applicability of topologies carried over from other domains.

1.2 Didactic structure of the thesis

In order to investigate this new challenge in detail, this thesis is structured as follows:

1. In chapter 2, the basics of functional safety are introduced. Key terms and concepts are explained briefly, to enhance the comprehensibility of the subsequent chapters.
2. Chapter 3 contains a thorough investigation of the current state of the art of fail-operational systems. The related functional safety standards and technical solutions of four transportation-related domains are evaluated in detail: avionics, railway, agricultural vehicles and automotive.
3. In chapter 4, the generic foundations of fail-operational automotive systems are laid. The applicability of the [ISO11] is investigated in detail. For any gaps found in the standard, improvement proposals are defined. As part of this detailed investigation, typical fault tolerant Hardware (HW) and Software (SW) architectures are analysed and evaluated from the viewpoint of the [ISO11].
4. Based on the previous results, in chapter 5, a mathematical model and an optimisation algorithm are developed, to support the optimisation of redundant system and HW architectures.

1.3 Typographic conventions

This thesis uses certain typographic conventions to enhance the comprehensibility for the reader. *Italic* is used to highlight important terms or statements.

Improvement proposal

In chapter 4 improvement proposal frames are used to highlight change or extension proposals to the [ISO11].

2 Introduction to functional safety

2.1 Basics of functional safety

The complexity of today's vehicles, especially its electric and electronic architecture, is increasing rapidly. Driven by the demand for faster, safer, more comfortable and "greener" cars, the automotive industry is pushed to develop newer and more complex functionalities than ever before. The new enhanced functions of these vehicles are mainly realised by an interconnected network of embedded systems, consisting of numerous ECU-s connected by sophisticated communication bus systems.

The automotive industry has been gathering experience over 100 years about constructing safe mechanical and hydraulic systems, e.g. conventional brake or steering systems. The development lifecycle of electronic HW and SW is a completely different challenge, though. Due to this reason, functional safety - a topic well-known in other industrial domains since the '70-s and '80-s - gained more attention. Various methods of functional safety have been applied even in the '90-s (e.g. for active cruise control, Anti-Lock Brake System (ABS), etc.), but it is still considered as a major breakthrough when state-of-the-art functional safety standards started to be applied. First, the [IEC10] - the domain independent safety standard - has been used for obviously safety related automotive products (e.g. active steering systems, Electronic Stability Program (ESP) systems). But since this standard was not tailored to the specialities of the automotive domain, its applicability was limited. Recognising this was the main motivation for the creation of the [ISO11].

As defined in the [ISO11], functional safety includes

> absence of unreasonable risk due to hazards caused by malfunctioning behaviour of E/E systems.

Note that functional safety *in terms of the [ISO11]* is focusing on E/E systems only. This point of view is carried over from the [IEC10], and is a result of the assumption that the industry is capable of constructing safe "non-E/E" systems.

At this point it is important to understand the difference between the related terms *safety*, *reliability* and *availability*:

Reliability of a system at time t is the probability that the system operates without a failure in the interval $[0, t]$, given that the system was performing correctly at time 0 [Dub13],

Availability of a system at time t is the probability that the system is functioning correctly at the instant of time t. Note, that if a system cannot be repaired, its availability is equal to its reliability [Dub13].

Note, that in some cases safety and reliability are contradictory. A completely passive ABS system, for example has 100% safety while having 0% reliability at the same time. A gun with 100% reliability pointed at my head, on the other hand, has 0% safety.

The basic principle of functional safety is that of the *risk reduction* (figure 2.1). Risk, as defined in the [ISO11] is

> combination of the probability of occurrence of harm and the severity of that harm.

Each E/E related system has an initial risk based on its function. For instance, a Steer-by-Wire (SbW) system is capable of steering without, or independently of, the intention of the driver. This failure mode is commonly known as *self steering*. Self steering is a high severity failure mode since it can obviously lead to the death of individuals. Without any counter-measures in the product or in its related development process, the occurrence of this failure mode would be unacceptably high (related to its severity). To reduce the associated risk, there are two options:

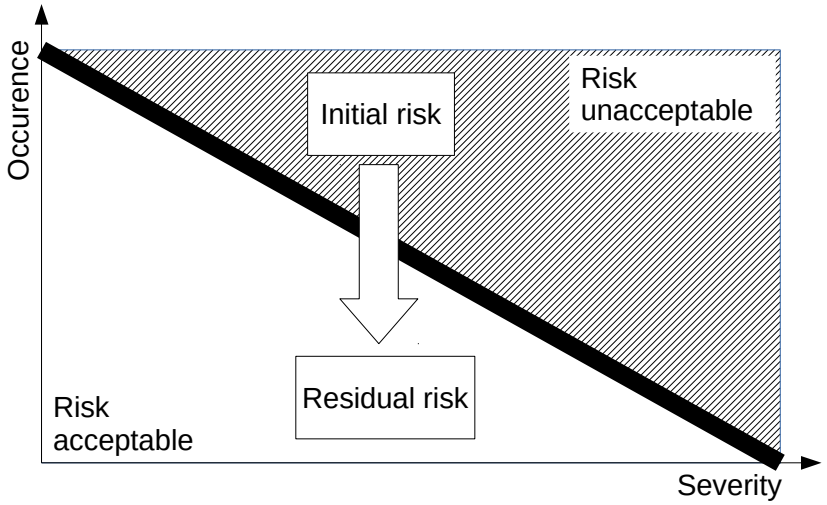


Figure 2.1: Principle of risk reduction

- Reduction of the occurrence by appropriate safety measures in the product (e.g. by a watchdog controlling severe microcontroller malfunctions) and in the process (e.g. by extensively testing the embedded SW).
- Reduction of the severity. This option is very difficult to realise, therefore seldom applied. In the SbW example this could be partially achieved by increasing the passive safety of the vehicles to an extent that a collision with other traffic participants would not lead to severe injuries or death.

Note, that one of the basic principles of functional safety is that there is no *absolute* safety. A residual risk is always present even in the safest systems. But as long as this residual risk is below the continuously shifting acceptance limit of our society, these products are considered safe.

2.2 Basic terms of functional safety

When it comes to functional safety, the main focus is on *failures* of the safety related product that need to be prevented and/or controlled appropriately. Regarding the definition of *faults* and *failures*, this thesis will stick to the definitions of the [ISO11] [part 1] and the recursive interpretation of the [ISO11][part 10]:

Fault: "abnormal condition that can cause an element or an item to fail"
Error: "discrepancy between a computed, observed or measured value or condition, and the true, specified or theoretically correct value or condition"
Failure: "termination of the ability of an element, to perform a function as required"
Failure mode: "a manner in which an element or an item fails"

The recursive definition mentioned above states, that the terms *fault* and *failure* can be used recursively, as shown in figure 2.2.

Faults and failures can be categorised based on their origin as systematic- or random HW faults. Systematic faults and failures are deterministic, and can be eliminated only by improving the design or the production process. Typical causes for systematic failures are poor design (e.g. wrong SW specification) or manufacturing-related issues (e.g. contaminated soldering). Random HW failures on the other hand occur in an unpredictable way, and are mainly caused by ageing or environmental factors. It is obvious that SW is prone to systematic failures only.

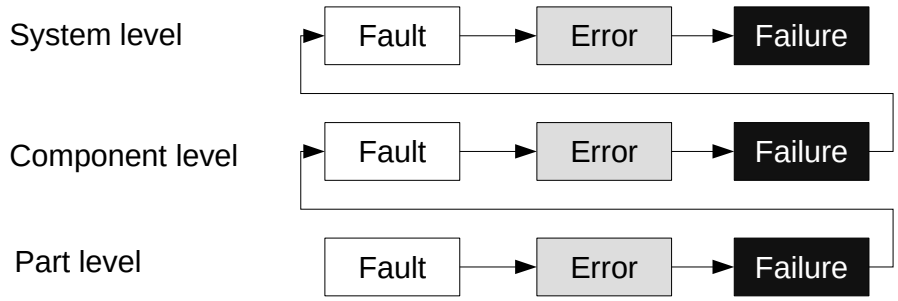


Figure 2.2: Recursive definition of faults and failures

Another classification of faults is based on their temporal behaviour. *Permanent faults* occur and remain until removed or repaired. *Transient faults*, on the other hand, occur and disappear subsequently. Transients can occur in HW elements for example due to cosmic radiation.

Related to the terms of faults and failures are the often misunderstood *hazard* and *hazardous event*. A *hazard* is defined by the [ISO11] as

> potential source of harm.

Only those failures can be hazards, where there are operational situations, in which the failure can lead to an accident. A *hazardous event* is exactly this failure (= hazard) combined with this unfavourable operational situation. For example, unintended gear change from the 5^{th} to the 2^{nd} is harmless when the vehicle is coasting at 60 km/h in a straight line. But the very same failure mode can lead to an accident when occurring during high speed cornering.

It is not only important to control failures but also to do so with the right performance. The term *Fault Tolerant Time Interval (FTTI)* gives guidance to this. The FTTI is defined by the [ISO11][part 1, 1.45] as

> time-span in which a fault or faults can be present in a system before a hazardous event occurs.

The concept is shown in figure 2.3: After a fault occurs, it will propagate through the system, leading to error(s), and that (or those) to failure(s). Some of these failures, combined with unfavourable operational situations can become to hazardous events. The goal of the safety engineer is to design safety mechanisms, that can control the faults or their effects before this hazardous event occurs.

Random HW failures can be also quantified, which leads to the next term: the *failure rate*. The failure rate is defined as the expected number of failures per unit time ([Dub13]). The failure rate of E/E HW over lifetime is represented by the *bathtub curve* (see figure 2.4). This failure rate function has three phases. Phase "A" is the infant mortality, the period of the early failures. Phase "B" is the constant random failure phase, also called useful lifetime. E/E systems are designed to be used in this phase, hence the name. The last phase, "C", is the wear out phase.

In the constant failure rate phase, the failure rate of the HW component is assumed to be λ. In this phase, the reliability of the component is $R(\lambda) = e^{-\lambda t}$. Therefore, the probability of failure is $Q(\lambda) = 1 - R(\lambda) = 1 - e^{-\lambda t}$ [Dub13]. The dimension of the failure rate is h^{-1} or (as used in many cases) *Failure In Time (FIT)*, where $1\,FIT = 1 \cdot 10^{-9} h^{-1}$ [ISO11] [part 5].

One of the key concepts of each functional safety standard is the *safe state*. The [ISO11] defines it as

> operating mode of an item without an unreasonable level of risk.

Note, that contrary to the common interpretation, the [ISO11], and also other safety standards, do not define the safe state as necessarily passive state. In case of an Electric Power Steering (EPS) system of a small passenger vehicle, the safe state is - as the term is commonly used - the passive state, where no steering support is provided. In this case the vehicle is still steerable but with increased effort. An

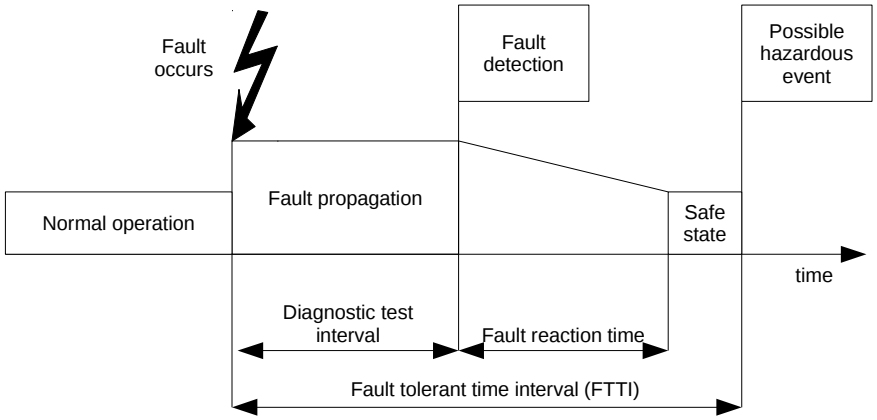


Figure 2.3: Fault tolerant time interval []

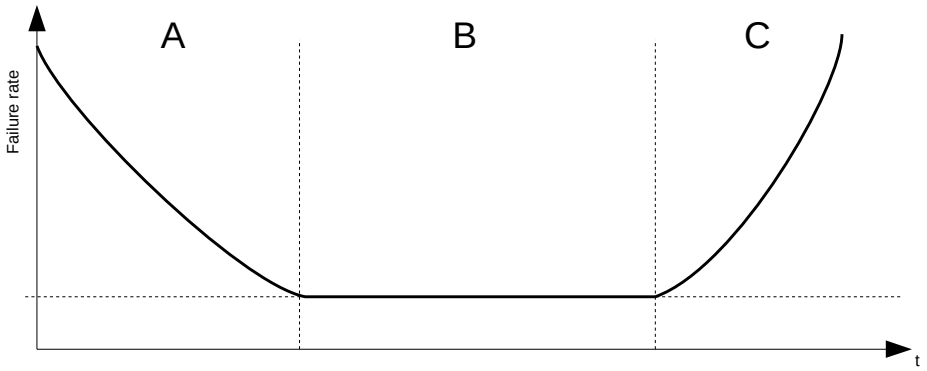


Figure 2.4: Bathtub curve [Dub13]

alternative option would be to define the completely fault-free operational state as second safe state, since also in that case, the system is without an unreasonable level of risk.

The definition of the term *safe state* leads to the different behavioural models a system can incorporate in the presence of failures. These terms are quite controversial since many interpretations exist. First, some of these existing interpretations will be shown, and then the ones that will apply in this thesis, will be presented.

[Neno7] defines the related terms as following:

Fail-operational systems remain functional in case of a subsystem failure.
Fail-silent systems enter a state that does not interfere with other safety related systems in case of a failure.

[Iseo8] defines these terms very similarly but rather as degradation steps:

Fail-operational is if one failure is tolerated and the component remains active. In his understanding this is required if no safe state exists if the component fails.
Fail-safe is, if after one or several failures, the system is brought to an active or passive safe state.
Fail-silent is, if after one or several failures, the system switches into a state where it does not interfere with other components.

Based on these definitions, this thesis will use the following terms as *types of behaviour*:

- Elements (i.e. systems, HW- or SW-elements) showing *fail-safe* behaviour enter an active or passive *safe state*, and cease to perform their functions, in case of a certain amount of failures.
- Elements (i.e. systems, HW- or SW-elements) showing *fail-silent* behaviour enter a *safe state* with no interference with other elements, and cease to perform their functions, in case of a certain amount of failures.
- Elements (i.e. systems, HW- or SW-elements) showing *fail-operational* behaviour continue to perform a defined set of their intended functions to a defined extent, with a defined performance, for a defined time in case of a certain amount of failures.

For example the term *fail-operational* system denotes thereby a system that shows fail-operational behaviour in case of a defined number of failures.

Safety related systems (and fail-operational ones, in particular) often rely on redundancy. Redundancy always denotes an additional mean to the means that would be necessary for the intended function. Redundancy can be *space redundancy* or *time redundancy* ([Dub13]). Space redundancy means additional components, functions or data. Time redundancy is always based on repetition of calculation or data transmission. An important attribute of redundancy is if it is homogeneous or diverse. Diversity always refers to different solutions to fulfil the same requirement. Diversity can be applied on various levels and to various extent, like in the following examples:

- Redundant components: two angle sensors measuring the same magnetic field, once by using the Hall-effect, while the second instance applies the magneto-resistive principle.
- SW redundancy: two SW-functions, designed and implemented based on the same specification, but by different teams, using different programming languages and compilers.
- Information redundancy: storing the same safety critical byte twice, where the second instance is inverted.

Homogeneous redundancy on the other hand means simply replicating the same elements. Homogeneous redundancy is used to cope with random HW failures, while diverse redundancy is applied to cope with systematic ones. The higher the degree of diversity is, the better the protection against systematic failures.

Related to diverse redundancy is the term of *dependent* failures. A and B are dependent failures if the probability of their simultaneous occurrence is $P(A \cap B) \neq P(A)\,P(B)$. Dependent failures can be split into two categories (see figure 2.5):

- *Common cause failures*, where two (or more) elements fail simultaneously due to one single event (i.e. the common cause).
- *Cascading failures*, where the failure of an element leads to a failure of another element.

Both types of dependent failures are unfavourable in redundant architectures. The existence of uncontrolled dependencies invalidates the independence argument, and the whole point of the redundant architecture. The reason is obvious: the primary goal of a diverse redundant architecture is to gain advantage of the N parallel elements. If they can fail simultaneously due to any dependency, this advantage is lost.

Safety related systems incorporate numerous diagnostic functions, that are intended to detect faults or failures, in order to cope with them. A very important attribute of diagnostic functions is the *Diagnostic Coverage (DC)*. The DC describes the fraction of the detected failures: $DC = \frac{\sum \lambda_{detected}}{\sum \lambda}$. It is important to note that the DC is no mathematical probability. It is common practice to use four discrete values: no coverage (= 0%), low (= 60%), medium (= 90%) and high (= 99%) DC. On the other hand, the [ISO11] defines informative methods to calculate the DC value more precisely.

2.3 Introduction to safety standards

Motivated by the increasing product complexity and various industrial accidents, the core functional safety standard, the [IEC10] was established, as domain independent standard consisting of seven parts:

Part 1 General requirements

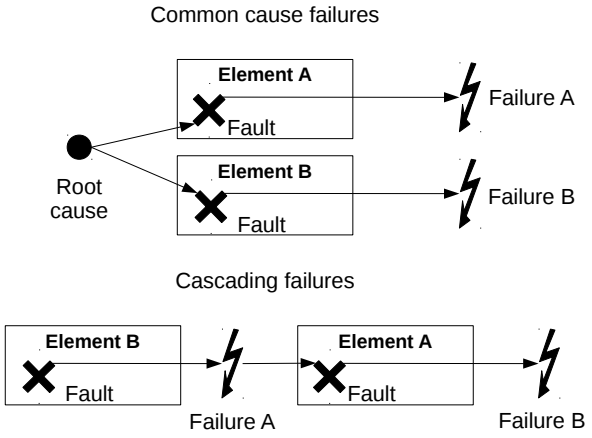


Figure 2.5: Dependent failures [ISO11]

Part 2 Requirements for electrical/electronic/programmable electronic safety-related systems
Part 3 Software requirements
Part 4 Definitions and abbreviations
Part 5 Examples of methods for the determination of safety integrity levels
Part 6 Guidelines on the application of IEC 61508-2 and IEC 61508-3
Part 7 Overview of techniques and measures

This standard already applied the concept of a safety lifecycle based on the classic V-model for the system, HW and SW. Due to the specifics of these three domains, the V-models are slightly different, even if identical in their cores and basic principles.

Based on the [IEC10], various safety standards have been derived to deal with functional safety aspects of specific domains, e.g.:

IEC 61513 for nuclear power plants
IEC 61511 for the process industry sector
ISO 25119 for tractors and agricultural machinery
ISO 26262:2011 for the automotive industry. Note that this standard is valid only for serial production passenger vehicles up to 3.5t.

The key point of functional safety standards is to provide guidance to:

- Processes for the development, manufacturing and operation of safety related products (e.g. extensive SW unit level testing is required).
- Safety related product features and characteristics (e.g. watchdog is required to monitor the microcontroller).
- Applied methods (e.g. equivalence class analysis is required for the definition of test cases).

An important aspect of the functional safety of safety related systems is their safety integrity. The safety integrity is the (not necessarily mathematical) probability, that the safety related E/E-system fulfils its safety requirements, under the given circumstances in the defined time-frame. Based on the concept of safety integrity, each safety standard defines some sort of safety integrity levels:

[ISO11] defines the Automotive Safety Integrity Level (ASIL), as one of four levels, from ASIL A to ASIL D, where D is the most stringent level.
[IEC10] defines the Safety Integrity Level (SIL), as one of four levels, from SIL 1 to SIL 4, where 4 is the most stringent level.

[ASI14] defines the Agricultural Performance Level (AgPL), as one of five levels, from AgPL a to AgPL e, where e is the most stringent level.

The higher the safety integrity level of a system, the more stringent requirements apply for them. These requirements include process, method and product related requirements, as mentioned above.

2.4 Introduction to the [ISO11]

2.4.1 Lifecycle model of the [ISO11]

The safety standard in focus of this thesis, the [ISO11], is based on the [IEC10], tailored in particular for the needs of the automotive industry. The main adjustments are the following:

- automotive safety lifecycle (including safety management, development, production, operation, service and decommissioning) has been introduced. The [IEC10] assumes low volume systems, that are developed, built, tested and installed at the plant. Contrary to that, passenger vehicles are produced in high volume, over a longer time period and need to be operated, maintained and decommissioned in a safe way.
- automotive-specific, risk-based safety integrity determination method, and the concept of the ASIL associated to safety requirements and safety goals have been introduced.
- required methods represent the state-of-the-art in the automotive industry
- requirement-oriented development process model
- methods to cope with distributed developments

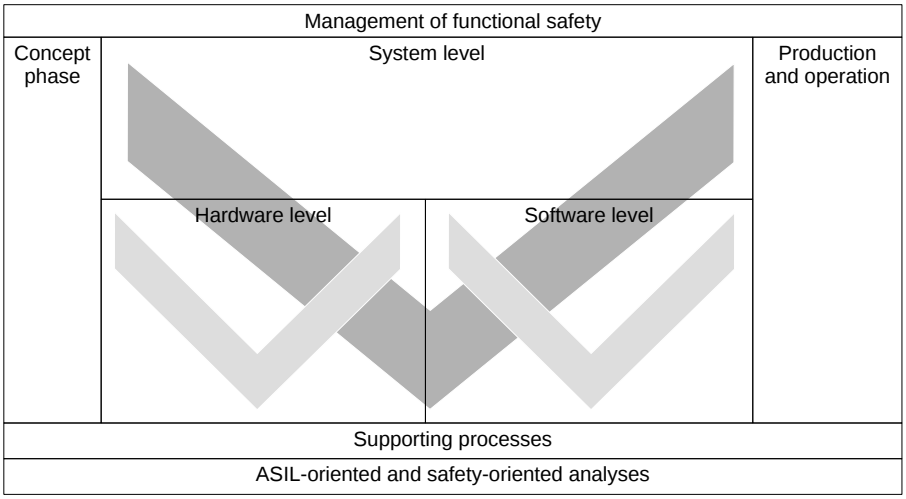


Figure 2.6: Lifecycle of the [ISO11]

The structure of the [ISO11] follows the lifecycle shown in figure 2.6:

- Part 1: Vocabulary
- Part 2: Management of functional safety
- Part 3: Concept phase
- Part 4: Product development at the system level
- Part 5: Product development at the hardware level
- Part 6: Product development at the software level
- Part 7: Production and operation

- Part 8: Supporting processes
- Part 9: Automotive Safety Integrity Level (ASIL)-oriented and safety-oriented analyses
- Part 10: Guideline on ISO 26262

As already mentioned above, the [ISO11] strictly follows the concept of requirement-based development. The hierarchy of the safety requirements is shown in figure 2.7 This requirement based approach

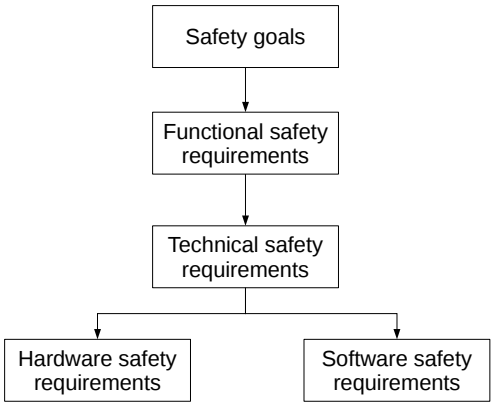


Figure 2.7: Requirement abstraction levels of the [ISO11]

is one of the cornerstones of the V-models used on the three abstraction levels. Only by appropriate (i.e. atomic, comprehensible, verifiable, unique and consistent) requirements is it possible to perform the verification and validation activities properly.

The core development processes (i.e. system, HW and SW development) are handled by the parts 4 – 6 . All three are based on the V-model, with domain- and safety-specific adaptations. These core development processes are backed up by an appropriate safety management process and various supporting processes (e.g. configuration management, requirements management, documentation). The technical development is started on the vehicle level with the concept phase. After the product has been developed successfully, part 7 of the standard gives detailed guidance how to produce, maintain, operate and decommission these systems safely.

Concept phase

The first technical phase of the safety lifecycle is the concept phase. In this phase, the system boundaries are defined, and the Hazard Analysis and Risk Assessment (HARA) is performed. As primary output of this phase, the safety goals with their respective ASIL-s are defined. Using these safety goals, the functional safety concept is defined on the vehicle architectural level. This concept is based on the definition of functional safety requirements defining safety related functions of the elements of the preliminary architecture.

System level

The development on the system level mainly focuses on the definition of the technical safety concept; on the system integration and testing; and on the safety validation. The technical safety concept defines the safety mechanisms implemented in the HW, SW or both (e.g. redundant data storage in the volatile memory with crosscheck). On the right side of the V-model, the system is integrated and tested step-by step (i.e. HW-SW-integration, system integration, vehicle integration), focusing on the correct implementation, robustness and performance of these safety mechanisms. One level higher, the safety validation provides evidence of compliance with the safety goals and the functional safety requirements,

and of the correctness of those. The assessment of the achieved functional safety and the following release for production are also part of the system level development. The safety integrity of the system lies mainly in the applied methods and processes. The higher the ASIL, the more sophisticated are the required methods.

Hardware level

Starting with the technical safety requirements allocated to the HW, the HW development lifecycle is initiated. The HW safety requirement specification further refines these input requirements. The development continues with the HW design, split into two phases: the architectural and the detailed design phase. After the design is finished, the HW architecture is analysed using a quantitative analysis technique. This analysis calculates the so called SPFM and LFM. These metrics quantify the robustness of the HW architecture against single point and latent faults, respectively. Subsequently, the probability of safety goal violations is analysed, by using one of two alternative methods. Currently the calculation of the PMHF is widely accepted, as the metric quantifying the residual failure rate of the HW. Note, that these three metrics do not calculate real-life failure rates but evaluate the safety integrity of the HW, by allowing the comparison of different designs. The evaluation of these three metrics is required for ASIL C and D and recommended for ASIL B. As final step of this development phase, the HW integration and testing is performed. These activities include for instance functional, environmental, EMC, robustness, fault-injection and life-time testing. The safety integrity of the HW lies on one hand in the applied methods and processes but also in the required target values for the three metrics mentioned above. The higher the ASIL the more stringent are the target values.

Software level

Starting with the technical safety requirements allocated to the SW, the SW development lifecycle is initiated. The SW development starts with a refinement of the technical safety requirements allocated to the SW. The output of this step is the SW requirement specification. These requirements are the reference for the SW design. The design phase consists – as in the case of the HW – of an architectural and a detailed unit design sub-phase. In the architectural design sub-phase, the SW is analysed to identify safety critical failure modes on an architectural level. Based on this information, safety mechanisms on the architectural level (e.g. range checks in case of ASIL A, diverse programming in case of ASIL D) are defined. The SW is then implemented in accordance with its design. The verification of the SW starts with the static verification of the units and of the architecture. Subsequently the three-step dynamic verification is performed, testing the actually implemented SW on the unit, integration and embedded SW level. The dynamic unit testing provides evidence that the implementation corresponds to the unit design. This verification step has the advantage of both test depth and width, since the units can be stimulated very well on their inputs. The integration and testing step proves that the SW is implemented in accordance with its architectural design specification. The SW units are integrated in (preferably) multiple steps, while the static and dynamic aspects of the SW architectural design are verified. The main focus here is on the correct implementation of the interfaces, the robustness of the SW and on the correct implementation of the integrated functions. The final dynamic verification step verifies the implementation against the SW requirements specification. The safety integrity of the SW lies mainly in the chosen process and methods for the respective ASIL. Dedicated safety mechanisms are required only at higher ASIL-s, like for instance the control flow monitoring for ASIL C or the diverse programming for ASIL D. This principle is identical to that of Software Process Improvement and Capability Determination (SPICE) ([VDA14]), as are the main features of the applied V-model.

Safety management

Part 2 of the [ISO11] gives guidance to project independent and project specific safety management, during and after the development phase. The cornerstone of the project independent safety management is the establishment of a corporate safety culture, that fosters the development and production of safe systems. This includes an appropriate organisation, a thorough functional safety training program and a company-wide safety process landscape. The project specific safety management uses the safety plan as

main document. This work product plans the safety relevant process steps, with their objectives, inputs, outputs, responsibilities and necessary resources. This is the key document for the safety manager to plan and track the functional safety activities. During the development, the *safety case* is compiled. This work product provides evidence and argument, why the safety objectives (e.g. the safety goals or functional safety requirements) of the product are met.

Production, operation, maintenance and decommissioning

One of the main improvements compared to the [IEC10] was the recognition, that it is not sufficient to *develop* a safe system, it also has to be *produced* as such; and it has to be operated and maintained safely. Even as the last event in the lifetime of a safety related system, it has to be decommissioned in a safe way. These aspects are all addressed in part 7 of the [ISO11].

Supporting processes

All activities of the core development processes (part 3 – part 6) rely on the supporting processes of part 8. These processes apply to the entire safety lifecycle (wherever applicable), and include the following topics:

- Interfaces within distributed developments
- Specification and management of safety requirements
- Configuration management
- Change management
- Verification
- Documentation
- Confidence in the use of software tools (i.e. software tool qualification)
- Qualification of software components
- Qualification of hardware components
- Proven in use argument

Safety analyses

Part 9 of the standard defines generic requirements to the safety analyses (e.g. FMEA, FTA, dependent failure analyses) and the rules of the ASIL-decomposition (see 2.4.3).

2.4.2 Hazard analysis and risk assessment (HARA)

In order to understand how the safety goals and the ASIL are determined, figure 2.8 explains the process of HARA, described by the [ISO11][part 3, clause 7].

The process follows the principle behind the term *hazardous event*: failure modes are investigated in various operational situations. Potential hazards are identified and their related risk is evaluated using three parameters:

Exposure describes the probability being in that particular *operational situation*, from E0 (= incredible) to E4 (= high probability). Note that the term exposure is not related to the probability of the hazard.

Controllability represents the probability that the driver, passengers or other traffic participants can avoid the specific harm in the hazardous event, from C0 (= controllable in general) to C3 (= difficult to control or uncontrollable).

Severity describes the extent of harm, from S0 (= no harm to any person) to S3 (= severe injuries, survival uncertain).

Based on these three parameters the ASIL of a hazardous event can be determined using table 2.1. The structure of the table corresponds to that of a classic risk-graph. Beside the four ASIL-s (from A to D), the level QM denotes that it is not required to cope with the requirements of the [ISO11], a state-of-the-art quality management system is sufficient.

To explain how this procedure works in practice, let's take for example an electric drive system:

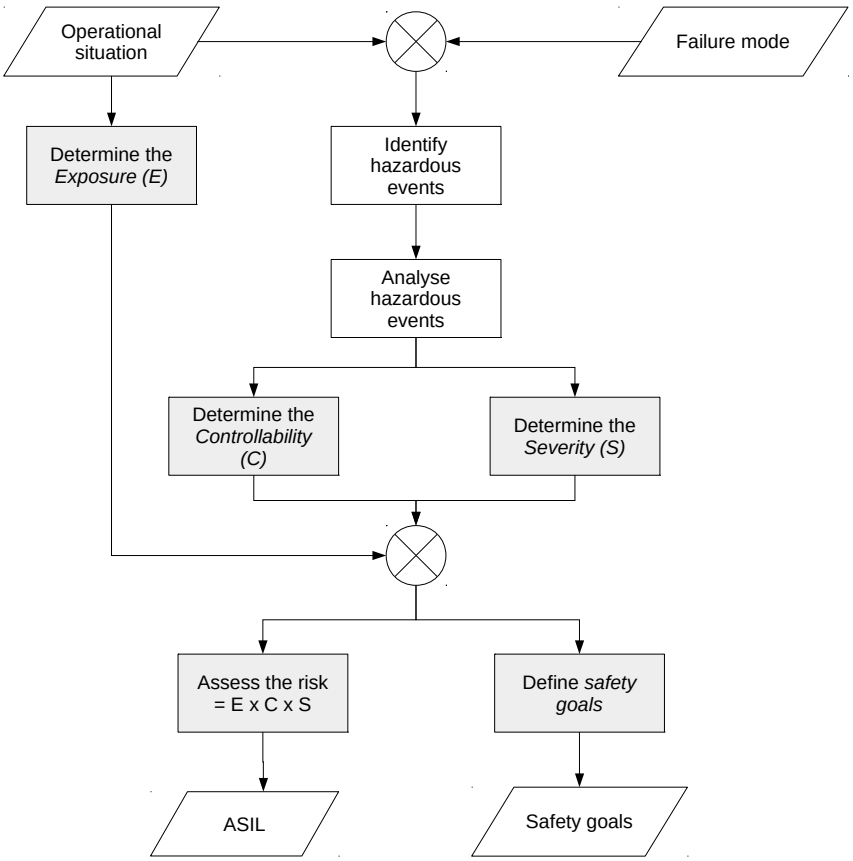


Figure 2.8: Process of the hazard analysis and risk assessment

- Operational situation: standing at a crossing in a city. Based on the Table B.2 of the [ISO11], its exposure is E4 (= high probability).
- The failure mode is that the electric drive system produce unintended propulsion torque that leads to a drive off. The controllability of this is estimated to be C2 (= normally controllable), since it can be assumed that more than 90% of the drivers would apply the brakes intuitively and immediately to avoid a collision with the crossing traffic.
- But if the control intervention of the driver is not successful the hazardous event could lead to a side collision with up to 50 km/h speed. The severity of this S3 (= life-threatening injuries (survival uncertain), fatal injuries).
- Combining $E4 \times C2 \times S3 = ASIL\,C$ (based on the Table 4 of [ISO11] [part 3])
- A resulting safety goal is: *Prevent unintended propulsion torque when none is requested by the driver*

Other typical examples for the four ASIL levels include:

- ASIL A: Clutch-based all-wheel drive system of a vehicle with longitudinal engine layout. In case of an unintended sudden decrease of the torque transmitted to the front axle, the vehicle may tend to oversteer. Severe oversteer can only occur on low friction surface (e.g. snow), therefore: $E2 \times C2 \times S3 = ASIL\,A$.
- ASIL B: Unintended disengagement of the parking brake. In this case it is often assumed that the exposure for parking on a hill with enough space to roll away and get to speed is rather low. Therefore the rating is $E2 \times C3 \times S3 = ASIL\,B$.
- ASIL C: See the example of the electric drive system above.

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

Table 2.1: ASIL determination

- ASIL D: Self-steering in case of an EPS or steer-by-wire system. Since this hazard can be fatal in almost any driving situation, and its controllability is very low, the rating is $E4 \times C3 \times S3 = ASIL\,D$.

Note that the exposure combined with controllability yields the occurrence of a hazardous event. The occurrence combined with the severity is the risk, as already defined in section 2.1 and in figure 2.1.

2.4.3 ASIL decomposition

One of the key features of the [ISO11] is the possibility of an ASIL decomposition. ASIL decomposition allows the designer to benefit from a sufficiently independent redundant architecture. ASIL decomposition is a measure to cope with systematic failures by *decomposing* a single safety requirement into two sufficiently independent requirements, and by implementing these two resulting requirements by two sufficiently independent architectural elements. The benefit is that the resulting two requirements have lower ASIL-s than the initial one. The principle behind this is that if there are two independent architectural elements realising the same function, than the probability that both fail simultaneously is lower even if their safety integrities are lower than that of the original requirement.

The [ISO11] offers the following decomposition schemes:

- Starting from ASIL D:
  - $ASIL\,D \rightarrow ASIL\,B(D) + B(D)$
  - $ASIL\,D \rightarrow ASIL\,C(D) + A(D)$
  - $ASIL\,D \rightarrow ASIL\,D(D) + QM(D)$
- Starting from ASIL C:
  - $ASIL\,C \rightarrow ASIL\,B(C) + A(C)$
  - $ASIL\,C \rightarrow ASIL\,C(C) + QM(C)$
- Starting from ASIL B:
  - $ASIL\,B \rightarrow ASIL\,A(B) + A(B)$
  - $ASIL\,B \rightarrow ASIL\,A(B) + QM(B)$
- Starting from ASIL A:
  - $ASIL\,A \rightarrow ASIL\,A(A) + QM(A)$

This procedure can be applied on any level (i.e. system, HW, SW) and even recursively.

2.4.4 Quantitative metrics of the [ISO11]

To understand the HW metrics of the [ISO11] already mentioned in section 2.4, first the fault classes of the standard have to be understood. Figure 2.9 shows the basic flowchart explaining how HW faults are categorised based on their fault propagation mechanisms.

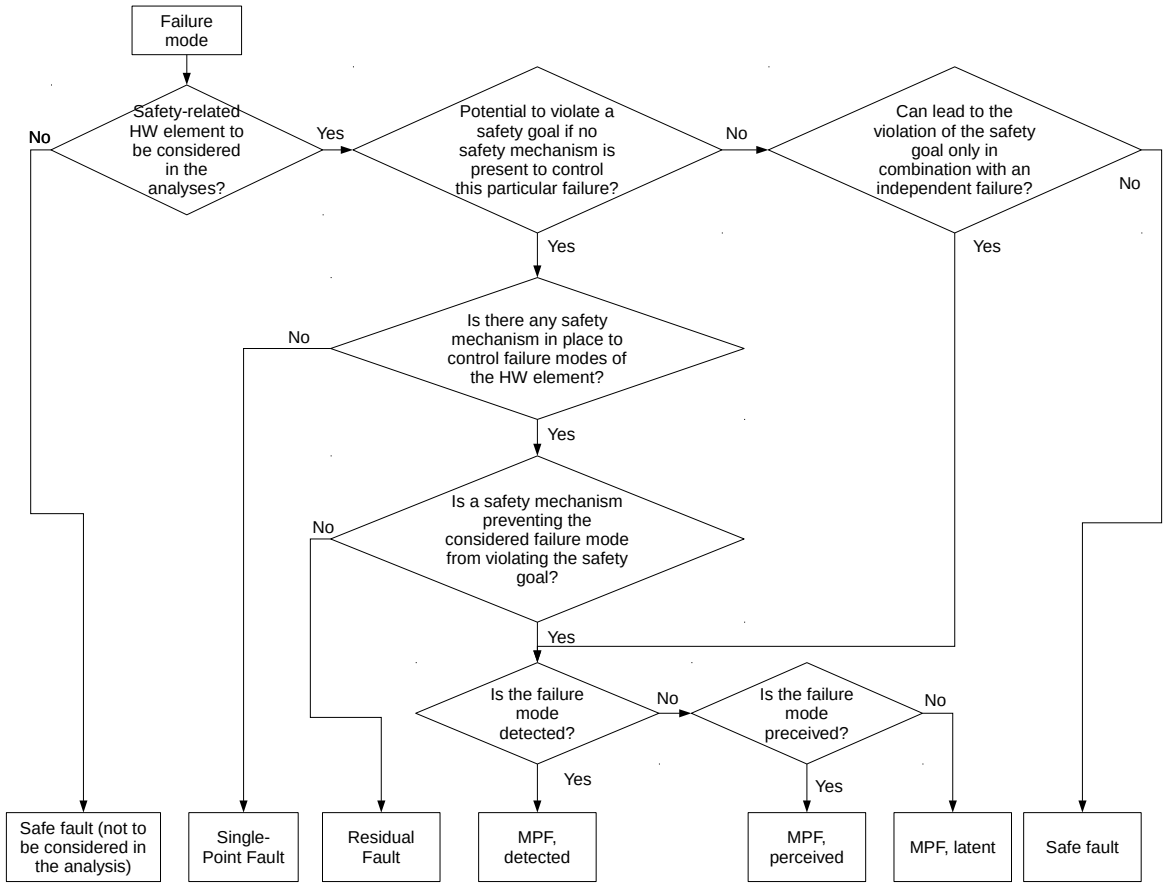


Figure 2.9: Fault categories of the [ISO11]

Based on these categories, the [ISO11] defines two architectural metrics: the SPFM and the LFM. The SPFM reflects the robustness of the system to single-point and residual faults, and is calculated as

$$SPFM = 1 - \frac{\sum_{SR,HW}(\lambda_{SPF} + \lambda_{RF})}{\sum_{SR,HW}\lambda} = \frac{\sum_{SR,HW}(\lambda_{MPF} + \lambda_S)}{\sum_{SR,HW}\lambda} \tag{2.1}$$

where:

- λ_{SPF} - Single Point Fault (SPF) failure rate [FIT]
- λ_{RF} - Residual Fault (RF) failure rate [FIT]
- $\sum_{SR,HW}\lambda$ - failure rate of the Safety Related (SR) HW [FIT]
- λ_{MPF} - Multiple Point Fault (MPF) failure rate [FIT]
- λ_S - safe failure rate [FIT]

The unit FIT is equal with $10^{-9}1/h$.

As visible in equation 2.1, the SPFM can be improved by the following means:

- Increase the $\sum_{SR,HW}\lambda_{MPF}$ by improving the coverage of the diagnostic mechanisms. Failures that are controlled by safety mechanisms are ”converted” from SPF or RF to MPF.

	ASIL B	ASIL C	ASIL D
Single Point Fault Metric	$\geq 90\%$	$\geq 97\%$	$\geq 99\%$

Table 2.2: SPFM target values

	ASIL B	ASIL C	ASIL D
Latent Fault Metric	$\geq 60\%$	$\geq 80\%$	$\geq 90\%$

Table 2.3: LFM target values

- Increase the $\sum_{SR,HW} \lambda_S$ by designing an architecture with primarily safe faults, that need no safety mechanisms, but still lead to a safe state on their own.
- Decrease the $\sum_{SR,HW} \lambda$ by creating a design using less components, or using components with lower failure rates.

The LFM reflects the robustness of the system to latent faults, and is calculated as

$$LFM = 1 - \frac{\sum_{SR,HW} (\lambda_{MPF,L})}{\sum_{SR,HW} (\lambda - \lambda_{SPF} - \lambda_{RF})} = \frac{\sum_{SR,HW} (\lambda_{MPF,D} + \lambda_{MPF,P} + \lambda_S)}{\sum_{SR,HW} (\lambda - \lambda_{SPF} - \lambda_{RF})} \tag{2.2}$$

where:

- $\lambda_{MPF,L}$ - Multiple Point Fault, Latent (MPF,L) failure rate
- $\lambda_{MPF,D}$ - Multiple Point Fault, Detected (MPF,D) failure rate

As visible in equation 2.2, the LFM can be improved by the following means:

- Increase the $\sum_{SR,HW} \lambda_{MPF,D}$ by improving the coverage of the diagnostic mechanisms, and by implementing an appropriate driver warning concept. Multiple point failures, that are reported to the driver, are "converted" from MPF,L to MPF,D.
- Increase the the $\sum_{SR,HW} \lambda_{MPF,P}$ by using a design, that allows the driver to perceive the presence of MPF-s. By these mechanisms MPF,L-s are "converted" to Multiple Point Fault, Perceived (MPF,P)-s.
- Increase the $\sum_{SR,HW} \lambda_S$ by designing an architecture with primarily safe faults, that need no safety mechanisms, but still lead to a safe state on their own.
- Decrease the $\sum_{SR,HW} \lambda$ by creating a design using less components, or using components with lower failure rates.

Both the SPFM and the LFM are, as visible in equations 2.1 and 2.2, relative metrics. Their target values, as required by the [ISO11] can be obtained by using the respective tables of the standard, or by deriving from the hardware architectural metric calculation applied on similar well-trusted design principles. Currently the former is common practice. The target values are shown in the tables 2.2 and 2.3.

The [ISO11], part 5, clause 9 defines two ways to evaluate the safety goal violations due to random hardware failures. The objective of this evaluation is to provide evidence, that the residual risk of a safety goal violation, due to random hardware failures, is sufficiently low. Both methods evaluate single-point faults, residual faults, and plausible dual-point faults, but in completely different ways. The first method calculates a probabilistic metric, the PMHF, by using usually a quantitative FTA.

The target values for the PMHF are derived from the respective tables of the [ISO11], or from the field data from similar well-trusted design principles, or from quantitative analyses applied to similar well-trusted design principles. Currently, the first source is common practice. The target values are shown in table 2.4.

All three metrics consider the E/E components of the system only and are calculated based on recognised industry failure rate and failure mode distribution sources (e.g. IEC/TR 62380, IEC 61709, MIL HDBK 217 F notice 2 [ISO11] [part 5, 8.4.3]), field data with appropriate confidence level or data based on expert judgement. Currently, the first one is the common practice.

	ASIL B	ASIL C	ASIL D
Probabilistic Metric for Random Hardware Failures	$< 10^{-7}\,h^{-1}$	$< 10^{-7}\,h^{-1}$	$< 10^{-8}\,h^{-1}$

Table 2.4: PMHF target values

Therefore, it is important to emphasize (as it is stated in the [ISO11]), that the PMHF and its related target values do not have any absolute significance, and are only meant to compare different designs. The resulting probabilistic metric must not be regarded as a real-life residual failure rate of the system. The main reasons for this are the following:

- The common practice is to use recognised industry failure rate sources. These sources contain failure rates, that are in some cases several multitudes higher than real-life values. These sources are still preferred, because their wide acceptance supports the main goal: to compare different designs.
- The diagnostic coverage of safety mechanisms, used in the calculations, is not a mathematical probability (as stated already in section 2.2).

The second method (see [ISO11] [part 5, clause 9.4.3]) is based on the individual evaluation of each residual and single-point fault, and of each dual-point failure leading to the violation of the safety goal in focus, in accordance with the following:

- *Single point faults* are acceptable, if their failure rate is below a certain limit, based on the ASIL (see [ISO11] [part 5, table 7]).
- *Residual faults* are acceptable, if their failure rate and the related part level diagnostic coverage are sufficient (see [ISO11] [part 5, table 8]).
- *Dual-point faults* are acceptable, if their failure rate and the related part level diagnostic coverage are sufficient (see [ISO11] [part 5, table 9]). Only dual-point faults need to be investigated, that are considered as *plausible*, in accordance with the [ISO11] [part 5, clause 9.4.3.8].

Note, that the [ISO11] [part 5, clause 9.4.3.2, NOTE 2] states, that MPF of a higher order than 2 need to be investigated, if the safety concept is based on redundant safety mechanisms.

2.5 Introduction to safety analysis techniques

In order to develop safe systems, it is essential to know their safety critical faults and failures, to be able to control them. Safety analysis techniques offer various ways to identify and evaluate the criticality of faults and failures within the system. Based on these results, safety mechanisms to control them can be defined, and their coverage can be evaluated. There are numerous safety analysis techniques, but this thesis will stick to the introduction of the ones commonly used in the automotive industry and by this thesis: Failure Mode and Effects Analysis (FMEA), FTA and FMEDA.

But first, before dwelling into the description and explanation of these three methods, let's take a look on the attributes of analysis techniques in general. Each one of these three methods has the following goals:

- Identify failure modes of the system and its subsystems and components
- Identify the causes and effects of these failure modes
- Support the definition of appropriate safety mechanisms to control these failures on different levels

There are various ways to categorise analysis methods. The main properties of them are the following [Wer12]:

- Analysis direction: inductive (= bottom-up), deductive (top-down) or bidirectional
- Qualitative or quantitative: quantitative analysis techniques use and calculate failure rates and failure probabilities. Qualitative analyses, on the other hand, evaluate the failure modes' related risk using purely qualitative criteria (e.g. severity rating from 1 to 10).

	FMEA	FMEDA	FTA
Analysis direction	Bi-directional	Inductive	Deductive
Qualitative or Quantitative	Qualitative	Quantitative	Quantitative
Single - or multiple point failures	Single	Single	Multiple
Application domain	Systems, mechanics, HW, SW	HW	Systems, mechanics, HW, SW

Table 2.5: Properties of the three most common automotive safety analysis techniques

- Single- or multiple failure analysis capability
- Application domain (systems, HW, SW)

According to these parameters, the analysis methods mentioned above can be described as shown in table 2.5.

2.5.1 FMEA

The FMEA is a generic failure analysis technique, with an immensely broad application field: it is applied for systems, HW, SW and processes. The FMEA has been developed by the US Army in 1949 for reliability evaluation. Subsequently it was applied in the Apollo project by NASA. The first automotive application was by Ford in 1977. One major milestone in the development of the FMEA was 1996, when the Verband der Automobilindustrie (VDA) method guideline has been published. This guideline provided the state-of-the-art framework, that made the FMEA able to cope with the broad spectrum of application fields it can handle today.

The five steps of a VDA-conform FMEA are the following (first three steps also shown in figure 2.10):

1. *Structural analysis:* The architectural model of the system in focus is created using a structural tree.
2. *Functional analysis:* Functions of the architectural elements are defined and then linked into the functional net. The functional net shows, which lower level functions (on the right side) contribute to which higher level functions (on the left side). These diagrams can also be regarded as functional breakdown diagrams.
3. *Failure analysis:* Failure modes are derived based on the functions using appropriate failure models (e.g. ”too high”, ”too low”, ”overflow” for calculation functions). Subsequently, these failure modes are connected into the failure net, where the causes are on the right, their effects on the left side of the net. It is important to emphasize that an FMEA can have multiple levels, not just three as in the example in figure 2.10. The failure net is called a *net* instead of a *tree*, because each node can have various causes and effects. This is one of the main differences to an FTA (see section 2.5.3). Top level failure modes (on the left-most level) are rated according to their *severities* (parameter S). This severity is rated from 1 to 10, where 10 denotes the highest severity. The lower level failure modes inherit their severities from one level above.
4. *Definition and analysis of the detection and prevention measures:* For each node in the failure net (but in the practice in most cases for the bottom level causes only), detection and prevention measures are defined. Prevention measures, as the name suggests, are intended to prevent failure modes from occurring. These measures include appropriate design methods, reviews, calculations, etc. Detection measures, on the other hand, detect failures *when they already occured*. These measures include in the design process mainly verification and validation activities; in the field operation domain the safety mechanisms. In this step the quality and coverage of the prevention and detection measures is rated by the parameters *occurrence* (O) and *detectability* (D), respectively. The occurrence is rated from 1 to 10, where 10 denotes the highest occurrence probability. The detectability is also rated from 1 to 10, but in this case 10 denotes the lowest level. In this step the risk associated to the failure modes is rated by the combination of S, O and C. One common – but very controversial – combination method is the Risk Priority Number (RPN), which is simply calculated as $S \times O \times D$.

5. *Optimisation:* In this step, where the risk is unacceptably high, optimisation is necessary. This optimisation can mean additional measures (e.g. better design methods) but also a design change.

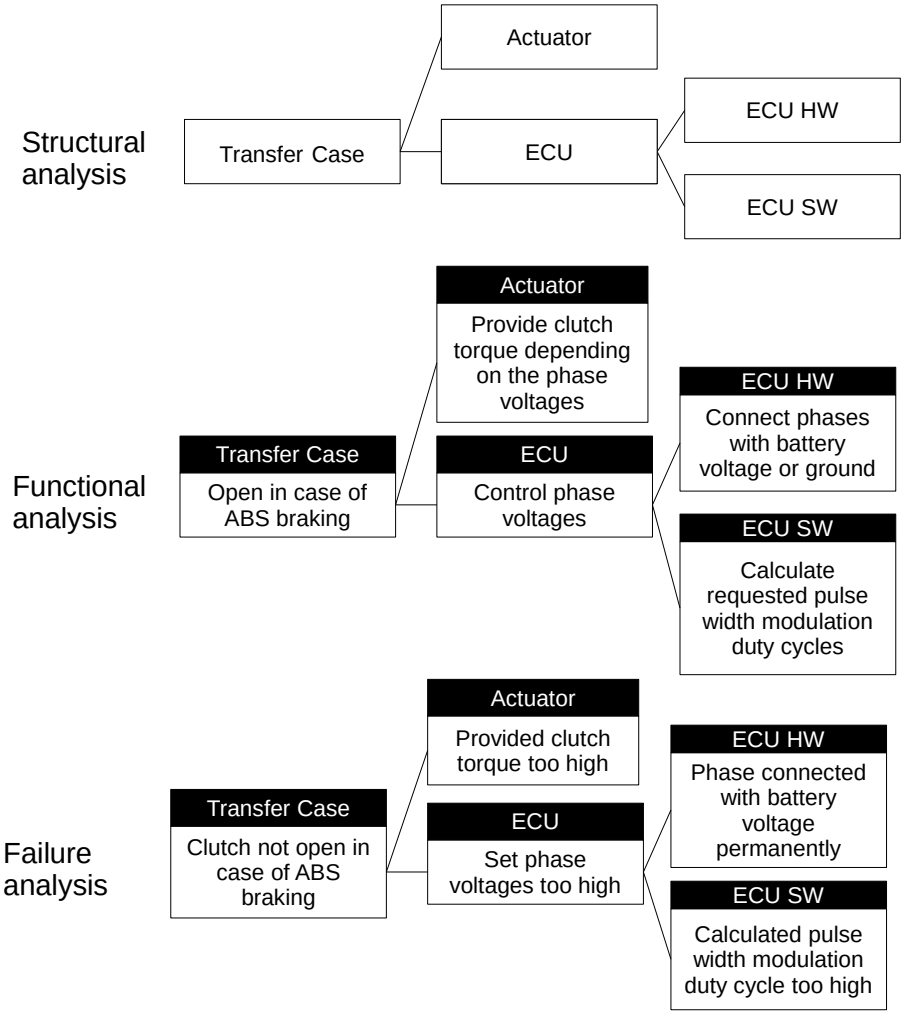


Figure 2.10: Steps 1-3. of the VDA FMEA method of an exemplary transfer case

An FMEA shall be started in an early project phase as preventive analysis and continuously updated parallel to the development. Only this way it can contribute to the reliability and safety of the product. Corrective FMEA-s on the other hand can only verify the correctness of an already existing design.

The biggest advantage of the FMEA technique is its broad applicability, the high number of literature and guidelines, and that it is the most widespread analysis method in the automotive industry. Its application for safety purposes, for HW and SW, has been proven in the past decade. Since the method gives a rough but solid framework only, the specific application can be tailored easily by experts to fulfil many different needs. The biggest drawback of the FMEA is, that it can handle single point failures only. Since many safety related systems rely on redundancy, their analysis requires that multiple point failures can be analysed and their risk evaluated properly.

2.5.2 FMEDA

The FMEDA is a table-based HW analysis method. It is a systematic analysis technique to calculate the HW metrics of the [IEC10] and the [ISO11]. The FMEDA method was developed in the 1980's

[Gob07], based on the form-sheet based FMEA process. The additional information (compared to the FMEA) includes the quantitative failure rate, the failure mode distribution of the components and the diagnostic capability of the safety mechanisms (i.e. the DC). The method has been further refined after the [IEC10] has been published. With the [IEC10] and the related optimisations, the FMEDA was fit for purpose.

The inputs of the FMEDA are:

- The system and HW architectural design
- The HW schematics, including the bill of materials
- The failure modes of each component (based on industrial standards and established literature)
- The failure rates of each component (in the practice based on industrial standards and databases)
- The technical safety concept, including the safety mechanisms and their DC
- Mission profile (i.e. temperatures, currents, voltages)

The FMEDA process consists of the following steps:

1. Enter bill of materials into the FMEA, including all components, their type (e.g. metal film resistor) and their functions.
2. Enter component failure rates in accordance with the selected component failure rate database (e.g. SN29500).
3. Enter component failure modes and their distribution in accordance with the chosen source.
4. Identify the effects of the component failure modes, and evaluate if they can lead to safety goal violations on their own (i.e. single point failure analysis), or in combination with other component failure modes (multiple point failure mode analysis).
5. Calculate the SPF and RF failure modes for the single point failure analysis.
6. Calculate the MPF,L failure rate for the multiple point failure analysis.
7. Rate the components if they are safety related. A conservative approach is to consider only those parts as safety related, that can lead to a safety goal violation on their own, or in combination with other failed components.
8. Calculate the HW metrics: the SPFM and the LFM in accordance with equations 2.1 and 2.2 respectively.

The FMEDA is calculated using tables or table-based specific tools. A basic example calculating the SPFM of a simple circuitry (consisting of a resistor and a capacitor) is shown in 2.11. The calculation of the LFM is performed in the same manner.

HW part	Function	Failure rate [1e-9 1/h]	Failure modes	Distribution	Failure mode failure rates [1e-9 1/h]	Effect	Potential to violate a safety goal?	Safety mechanis to control this failure mode?	DC	Lambda_SPF/RF
R001	RC-filter resistor	1	Open	40,00%	0,4	No signal transmitted to the microcontroller	1	Signal range check	99,00%	0,004
			Drift 0,5x	30,00%	0,3	No significant effect	0			0
			Drift 2x	30,00%	0,3	No significant effect	0			0
C001	Filter capacitor	10	Open	20,00%	2	Signal too noisy	1	Signal range check	60,00%	0,8
			Short	20,00%	2	Signal stuck at 0V	1	Signal range check	99,00%	0,02
			Drift 0,5x	30,00%	3	No significant effect	0			0
			Drift 2x	30,00%	3	No significant effect	0			0
				Lambda =	11				Lambda_SPF /RF =	0,824
				SPFM =	92,5%					

Figure 2.11: Example - ISO 26262:2011 compliant FMEDA

The FMEDA is a very structured mathematical calculation based on a clear evaluation criteria for the HW failure modes. The [ISO11] and the [IEC10] have two major differences in the calculation principle:

1. The [ISO11] has a HW-metric related to multiple point failures, while the [IEC10] only considers single point failures.
2. The [ISO11] takes only E/E-HW into account, while the [IEC10]'s focus is also extended to mechanical, hydraulical, etc. parts (e.g. valves).

2.5.3 FTA

The FTA is a purely deductive, systematic analysis technique. It can be both qualitative or quantitative depending on the goal and purpose of the analysis. Its major advantage compared to the FMEA is the ability to analyse multiple point failures using Boolean operators. The FTA has been developed by Bell Telephone Laboratories in 1962 [Wer12], and has been optimised by Boeing for the analysis of an intercontinental ballistic missile. In the 1970's and 1980's it has been used for the analysis of nuclear plants. The FTA provides a graphical and logical representation of the causes and their combinations leading to an undesired top event. If the FTA is used for safety analysis purposes, this top event is the violation of a safety goal (in terms of the [ISO11]) or a top level hazard (in terms of the [IEC10]).

Figure 2.12 shows the most common event symbols:

Event can be top or intermediate event representing a failure mode on the top or on an intermediate level.

Basic event is representing an event on the bottom level of the analysis, that cannot be broken down further. These events are part or component failure modes in most cases.

Undeveloped event is an event, that is not broken down further intentionally, or due to the lack of information.

External event is not always representing a failure mode, but an event that is expected to occur.

These events are linked into the failure net by logical gates. The most typical gate types are shown in figure 2.13.

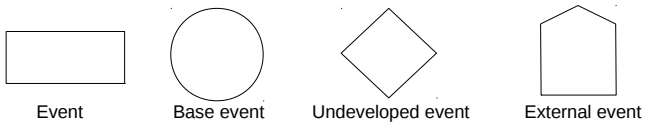


Figure 2.12: FTA events

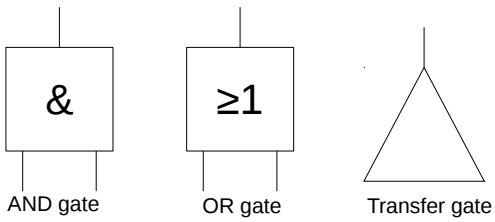


Figure 2.13: FTA gates

The FTA is constructed in the following way:

1. Define top event.
2. Identify causes, that can lead to the top event (on their own or in combination with others).
3. Identify the logical gates, that describe the combinations of these events leading to the top event, and link the causes with these gates.
4. Apply the first three steps recursively, until the level of the base events is reached. The level of base events can be chosen arbitrarily depending on the overall analysis concept.

Figure 2.14 shows a simple example of a parallel hybrid electric vehicle. To lose propulsion torque, both power sources (i.e. the internal combustion engine and the electric motor) must have a failure or the mechanical connection from the power sources to the wheels may be interrupted (i.e. gearbox shifts to neutral).

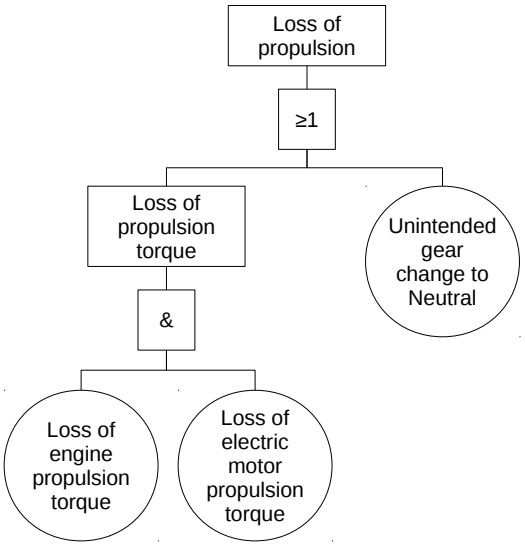


Figure 2.14: Example - qualitative FTA of a parallel hybrid electric vehicle

This very simple example already shows the concept of the cut sets. A cut set is a set of base events that lead to the top event if occurring simultaneously. A cut set is minimal if it cannot be reduced further. Cut set analysis is one of the most important procedures of a qualitative fault tree analysis. The cut set analysis is crucial to identify common cause failures of parallel paths. In the example the cut sets are {"Loss of engine propulsion torque"; "Loss of electric motor propulsion torque"} and {"Unintended gear change to Neutral"}. Note that both are minimal cut-sets.

After the qualitative fault tree is created, the probability of the base events is calculated based on their failure distribution functions (see chapter 2.2 for further details). After this has been calculated, the probabilities of the higher level events are calculated using the following basic probability equations (assuming that A and B are independent):

$$P\left(\overline{A}\right) = 1 - P\left(A\right) \tag{2.3}$$

$$P\left(A \cap B\right) = P\left[A\right] P\left[B\right] \tag{2.4}$$

$$P\left(A \cup B\right) = P\left[A\right] + P\left[B\right] - P\left(A \cap B\right) \tag{2.5}$$

Using these equations, the PMHF metric of the [ISO11] can be calculated for each safety goal violation. In that case the top event is the violation of the safety goal. The base events can be HW part failures but higher level failure modes as well, if their failure rates are calculated using an inductive method (e.g. FMEDA).

For higher level ASIL-s both an inductive and a deductive analysis is required. The principle behind this is that the top-down and bottom-up methods have their advantages, and that the simultaneous application of both yields the best analysis coverage. In the automotive industry, this requirement is fulfilled by a combination of FMEA, FTA and FMEDA in most cases. Note, that since the FTA can only analyse the violation of safety goals, it is not applicable to calculate the SPFM and LFM. Therefore, an FMEDA is always necessary. For single channel systems with almost no dual point failures, a simplified PMHF calculation method is proposed by the [ISO11][part 10]. This way a more conservative value is calculated but with less effort, using an FMEDA.

3 Results of the literature research - the state of the art

3.1 Fail-operational behaviour and fault tolerance in general

3.1.1 Fail-operational aspects in the [IEC10]

As already described in section 2.3, the [IEC10] is the "mother" of all industrial functional safety standards. Therefore, it is essential to understand whether and how this standard addresses the issue of fault tolerance and fail-operational systems.

The [IEC10][part 4, 3.1.13] defines the safe state as a state of the equipment under control, where it is safe. In the note below it also states that in some cases a continuous control is needed to maintain the safe state, and that this continuous control can be required for a short or for an undefined time period.

In addition to this, the [IEC10][part 1, 7.10.2.5] requires the following to be defined:

- how the safe state of the equipment under control is reached or maintained,
- whether and for how long it is necessary to provide continuous control to reach or maintain the safe state of the equipment under control.

Contrary to other standards, the [IEC10][part 4, 3.6.3] defines the term *fault tolerance* explicitly. It denotes the ability of a *functional unit* to maintain a *required* function even in the presence of failures or errors.

These definitions and this approach suit perfectly for fail-operational systems: the safe state can be an active state, this active state can be limited to a set of functions, and these can be maintained only for a short period of time.

On the architectural level, the [IEC10][part 2, 7.4.4.1.1] defines the term Hardware Fault Tolerance (HFT). HFT N means that it takes at least $N + 1$ failures to corrupt the safety function in focus. In the same requirement, the standard also states that it is allowed to exclude some failures from the determination of the HFT if their failure rates are insignificant in relation with the overall safety integrity of the system. This is an aspect that is recurring in the literature (e.g. [Iseo8]), although mainly due to cost efficiency reasons. Note, that the [IEC10] (contrary to the [ISO11]) defines the SIL in terms of acceptable residual failure rate (among other parameters). Note as well, that the HFT is not explicitly required for the sake of fail-operational systems. The HFT is required simply to yield safer systems, without special focus on their fault tolerance capabilities.

An interesting aspect of the [IEC10] is what needs to be performed if a failure is detected. The standard requires that in case of a $HFT > 0$ one of the following shall be performed:

- a reaction shall be triggered to reach or maintain a safe state,
- the part of the system, affected by the failure, shall be isolated to ensure the safe operation of the equipment under control, for the time the failed part is repaired.

This means that this standard already considers and even refers to one of the major fault tolerance techniques: fault isolation (see also [Dub13]).

An interesting aspect of safety standards is, whether they address typical fault tolerance techniques (see section 3.1.2), and whether they address them with the intention of requiring fail-operational behaviour. This aspect will be investigated for all related standards in this thesis.

The [IEC10][part 2, tables A.2-A.14] address the following techniques, related to HW:

- Monitored redundancy, as monitoring function for electrical components,
- 16 bit signature (or Cyclic Redundant Code (CRC)), as as monitoring function non-volatile memory,

- Block replication, as monitoring function for non-volatile memory,
- Multi-channel parallel output, as a diagnostic measure for external interfaces,
- Input comparison/voting, as a diagnostic measure for external interfaces and sensors,
- HW redundancy, as a diagnostic measure for internal communication,
- Transmission- and information redundancy, as a diagnostic measure for internal communication,
- Switch to secondary power supply, as a reaction measure to power supply failures.

These diagnostic methods are not listed as measures for fail-operational behaviour or fault tolerance, but as simple diagnostic functions. The referred tables are only intended to give estimates about their diagnostic coverage.

Fault tolerance mechanisms related to the SW are mentioned in a very similar way. The [IEC10][part 3, table A.2] addresses the following SW architectural measures:

- Diverse redundancy (different types with different degree of diversity),
- Recovery blocks (see section 3.1),
- Regeneration by repetition,
- Graceful degradation, meaning that the faulty parts of the SW can be deactivated.
- Artificial intelligence - fault correction,
- Dynamic reconfiguration, meaning that in case of a HW fault, SW functions executed by this faulty unit are allocated to another, correctly functioning HW resource.

Note 2 of table A.2 even remarks that fault tolerance techniques of this table shall be applied together with corresponding techniques for the HW, defined in [IEC10][part 2]. A curious aspect of this table is that most of these techniques are not highly recommended or there is even a recommendation against them (e.g. artificial intelligence - fault correction). Hence, it is very unlikely that these methods are preferred against others with higher degree of recommendation.

3.1.2 Basics of fail-operational behaviour and fault tolerance

[Dub13] and [Iseo8] give a very thorough and comprehensible overview of fault tolerant architectures and fault tolerance techniques in general. A system is considered fault tolerant if it can perform its intended function also in the presence of faults [Dub13]. Fault tolerance is always based on some sort of redundancy, since single channel systems are prone to single point failures. There are various ways to enhance the fault tolerance of a system:

- *Fault masking* is a technique where it is ensured that the system produces correct outputs only even in the presence of a fault. Majority voting is a typical example for this.
- *Fault detection* with subsequent *fault containment* ensures that the fault is isolated and cannot propagate. Fault detection can be based on various diagnostic principles, like cross-checks, plausibility checks, etc.
- Or alternatively *fault detection* with subsequent *fault recovery* ensures that the system recovers appropriately after a fault, to continue normal operation.

The following sections will focus on four typical domains where redundant designs are applied [Dub13]:

- HW and systems
- information redundancy
- time redundancy
- SW

HW and system redundancy

As stated above, fault tolerance and fail-operational behaviour always rely on redundancy. Redundant HW and system architectures can be characterised in many ways. [Dub13] differentiates between *passive*, *active* and *hybrid* redundancy. Passive redundant are systems, where fault tolerance is achieved by masking the failures without explicit fault detection functions. In case of active redundancy, on the

other hand, faults are detected and controlled by appropriate actions. Hybrid redundancy combines both previous types. A similar differentiation is used by [Iseo8]. *Static redundancy* is based on parallel modules that are simultaneously active. Their outputs are continuously compared by a voter, masking incorrect values. In case of *dynamic redundancy*, the primary module is in charge as long as no failures is detected by the diagnostic functions. In this case, the backup module takes over the role of the primary one, which is deactivated to encapsulate the failure. Dynamic redundancy can be split into *hot standby* and *cold standby* architectures. In case of a hot standby, the backup system is operating in parallel to the primary one. Contrary to that, in case of cold standby, it is only activated if the primary unit fails.

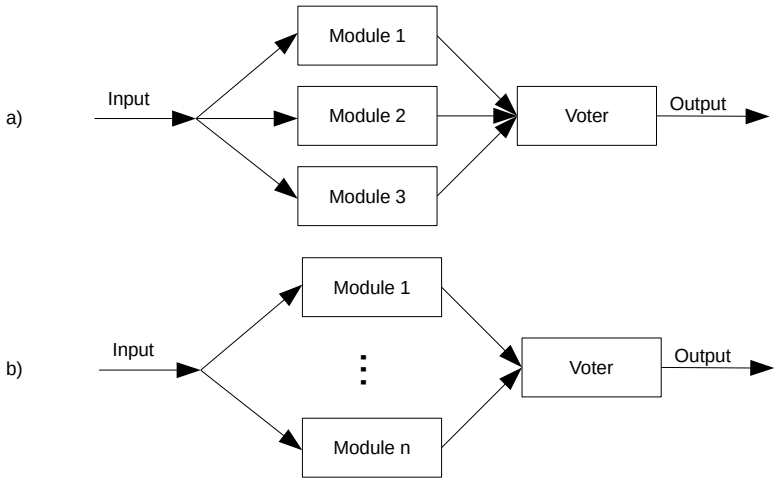


Figure 3.1: Common static redundant architectures, a) TMR, b) NMR

Figure 3.1 shows the most common static (or passive) redundant architectures applied for systems and HW:

- *TMR*, using three modules in parallel and a single voter. The outputs of the three modules are arbitrated by a majority voter. The TMR system can obviously tolerate the failure of one module. Since the voter is not redundant in this basic architecture, it is prone to single point failures. But since its complexity is usually significantly lower than that of the three modules, this safety gap is tolerated.
- *NMR* uses the same approach as TMR but with a higher (preferably odd) number of units. The advantage of NMR compared to TMR is the higher degree of fault tolerance. In case of $N = 2k + 1$ modules, k failures can be tolerated.

The main challenges of static redundant architectures are the management of timing jitter and the selection of the appropriate voting principle. The former is caused by the diverse implementation of the N modules. Since their implementation (or even their algorithms) are not identical, their temporal behaviour is different as well. Therefore, the voter has to be prepared to handle these timing discrepancies. Due to the same reason (i.e. diversity), in some cases it cannot be expected that all N modules produce exactly the same outputs. In that case a simple majority voting would not be sufficient (since all N outputs could be different). To handle this issue, various voting principles have been developed in the past. According to [Kum13] there are exact and inexact voters. *Exact voters* need absolute agreement, meaning that two inputs agree only if they are exactly the same. This voting principle has two major issues: noisy environments and design diversity, since both lead to inputs that probably will not be exactly the same. Contrary to that in case of *inexact voters* two inputs are in agreement already if the difference between them is below a certain threshold. This threshold is selected considering safety and robustness aspects, and is therefore always a compromise. Due to this reason the DC of inexact voters is lower than that of their exact counterparts. Another classification of voters is based on how they produce their outputs. According to [Sha09] there are following main types:

- *Majority voters* select an agreed value based on an odd number of inputs. If in case of N inputs $\frac{N+1}{2}$ inputs agree, the voter can produce an output. Note, that if this condition is not fulfilled, the voter is not able to produce an output. Some designs raise an error flag in this case. Obviously, in case of inexact voters, a rule has to be defined, how the output is generated from agreed but still different inputs.
- *Plurality voters* can produce an output if certain number of inputs agree. This number shall be ≥ 2 but can be lower than the majority number. As the name suggests, the input with the most votes is chosen.
- *Median voters* select the middle value of its inputs. This type of voter does seldom fit to safety related systems.
- *Average* and *weighted average voters* calculate their outputs as the average or weighted average of their inputs. The main disadvantage of this voter type is that one single wrong input will lead to a certain error in the output. The weighted average voting principle is meant to compensate this effect to some extent by applying higher weights to the inputs that are suspected to be more reliable.

The fault detection and correction capabilities of the different voter types may vary.

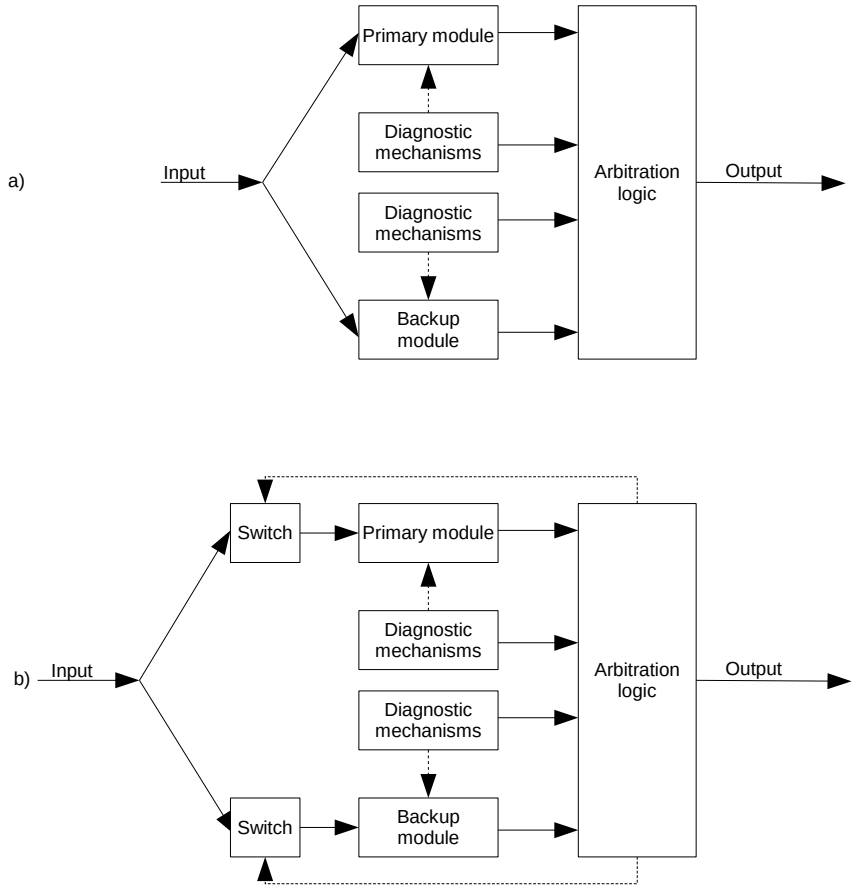


Figure 3.2: Common dynamic redundant architectures, a) Hot standby, b) Cold standby

Figure 3.2 shows the most common dynamic (or active) redundant architecture applied for systems and HW: *standby redundancy*. This solution consists of a primary module and n-1 backups. These architectures are always based on dedicated diagnostic mechanisms that can detect failures of the modules. If a failure is detected in the primary module, it is reported to the arbitration logic, that selects the next backup. The backup modules can be operational in parallel to the primary one (*hot standby*) or

passive (*cold standby*). The former has a faster response time, but the spare units consume power and are exposed to ageing. The latter is slower in response, but minimizes power consumption. Standby redundancy with N modules can tolerate $N-1$ failures. Compared to the TMR it is noticeable that for the same degree of fault tolerance, less modules are required. The drawback of standby redundancy is the necessity of the diagnostic mechanisms, that can be more complicated than a straightforward voter.

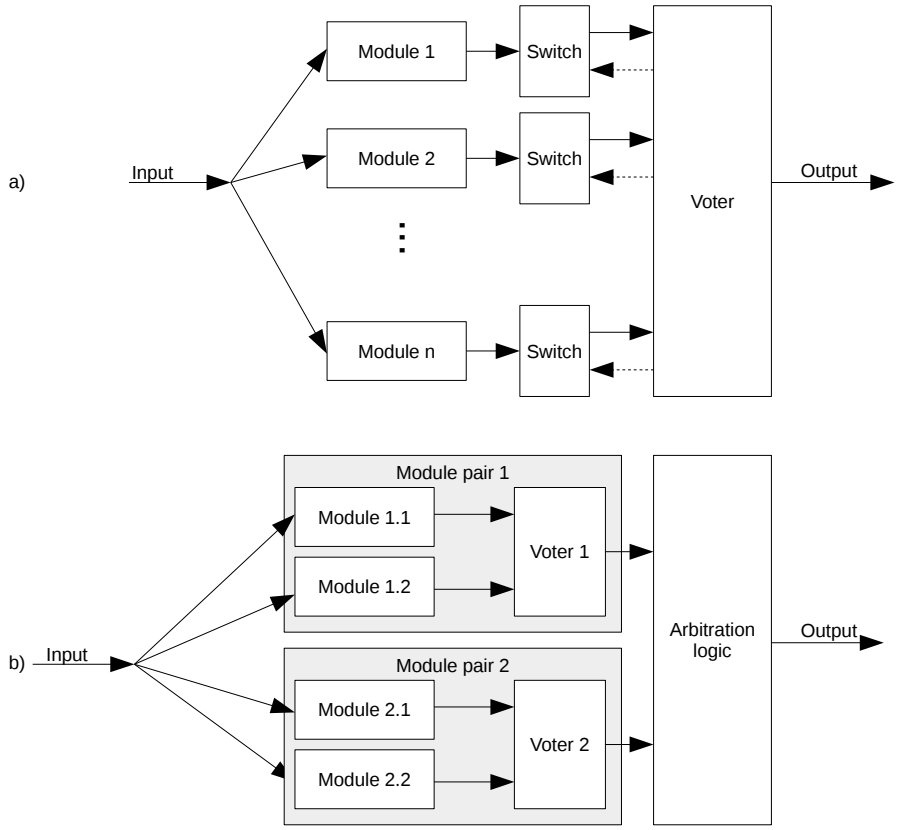


Figure 3.3: Common hybrid redundant architectures, a) Self-purging, b) Duo-duplex

Hybrid redundancy (see figure 3.3) intends to combine the two approaches above ([Dub13], [Neno7]):

- *Self-purging redundancy* is based on NMR by extending it with switches on the outputs of the redundant modules. Based on the result of the voting, the voter deactivates the faulty module(s) and reconfigures itself to cope with the decreased number of inputs. Self-purging redundancy with N modules can mask $N-2$ failures.
- *Duo-duplex redundancy* is basically a hot standby system, where the diagnostic mechanism is based on double redundancy with voting. The architecture consists of two duplex systems with separate voters. If a voter detects a failure of the primary module pair, the arbitration logic selects the backup one. Advantage of this architecture, compared to simple standby redundancy, is the simple detection mechanism. Disadvantage is the high number of modules (i.e. 4). A duo-duplex system can tolerate 1 fault. Note that its fault tolerance capability is equal to that of the TMR architecture, while using one module more.

In case of redundant HW and systems, it is important to understand, that homogeneous redundant architectures are sufficient against random HW faults and some local environmental disturbances (e.g. alpha particle hitting a microcontroller). If tolerance against systematic faults and broad environmental

disturbances (e.g. high temperature) is necessary, diverse redundant architectures are required. Section 2.2 explains the difference between the two types in detail.

Information redundancy

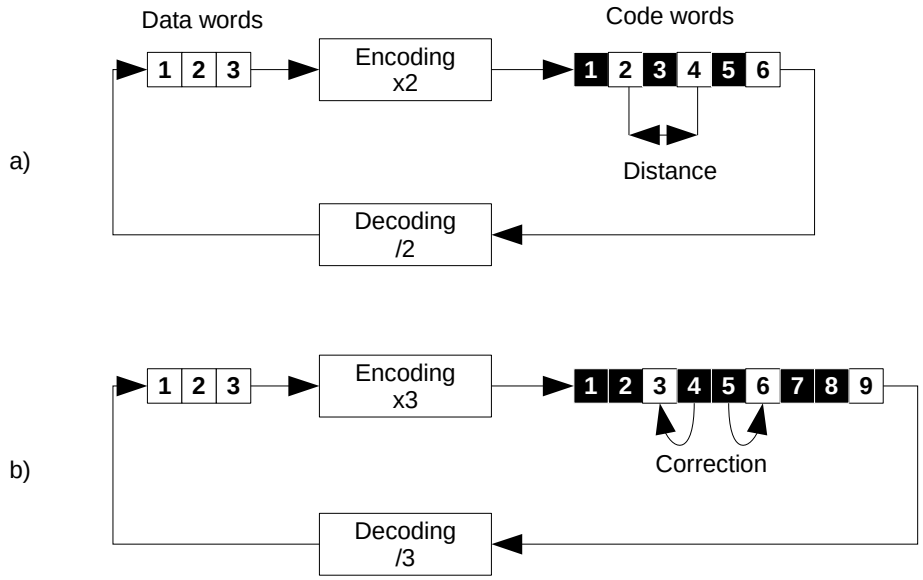


Figure 3.4: Basic principle of coding, a) coding with fault detection capability, b) coding with fault correction capability

Information redundancy is the commonly used mean to achieve fault tolerance of stored or transmitted data. Information redundancy often relies on *coding* [Dub13] [Ulb14]. Coding provides fault detection and (in some cases) fault correction capability for transient data failures. The basic workflow of coding-based information redundancy is shown in figure 3.4: n bit *data words* are encoded based on the *coding scheme* into $n + k$ bit *codewords*. In our example, the encoding scheme is simple: data words are multiplied by 2 and 3 (example a) and b), respectively). During *decoding* the data words are extracted from the codewords.

The fault detection and correction capability of coding is based on the fact that assuming n bit data words and k code bits, there are 2^{n+k} codewords but only 2^n of them are valid. These are written black with white background in figure 3.4. According to this, if the stored or transmitted codeword is corrupted, this fault can be detected. Faults cannot be detected if the corrupted codeword is valid, though. The distance between valid code words depends on the coding scheme. This distance (i.e. the number of bits that have to be changed to get the next valid codeword) is called the *code distance* [Dub13]. In example a) only fault detection is possible, since the distance of an invalid codeword is equal to the two adjacent valid ones. On the other hand, in example b) an invalid codeword is always closer to one of the adjacent ones. A basic assumption in coding theory is that faults affecting fewer bits have higher probability than faults affecting many. Using this assumption, if the codeword is invalid, it can be assumed that the original codeword was the nearest one. In figure 3.4 example b), the codeword 4 is corrected to 3, but 5 is corrected to 6. A coding scheme with a code distance of C can detect $C - 1$ bit and correct $\frac{C-1}{2}$ bit faults.

Coding schemes can be categorised in many ways. One important distinction is, if the code is separable. In case of a separable codeword with the length $n + k$ the first n bits are the data word and the last k bits are the code bits (see figure 3.5. The advantage of separable codes is, that the data word (i.e. the information the recipient is interested in) can be obtained without a complicated decoding procedure. In time-critical applications, or if there are recipients in the network, which are immune to corrupted data, this property is valuable.

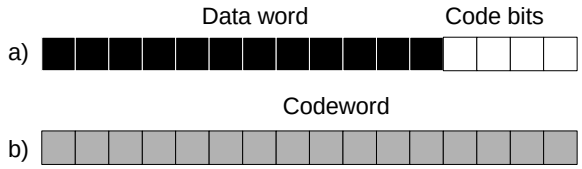


Figure 3.5: Structure of codewords, a) in case of a separable code, b) in case of an unordered code

A few examples of the most common coding schemes are the following ([Dub13])

- *Parity codes* extend the data word by one bit determining the parity of the data word. For example the data word $[0\,1\,0\,1]$ is extended by a 0 to yield the codeword $[0\,1\,0\,1\,0]$. This is called even parity, since the codeword contains an even number of 1's. If we encode the data word $[0\,0\,0\,1]$ using the same encoding scheme the resulting codeword would be $[0\,0\,0\,1\,1]$. During decoding and fault checking, the number of 1's in the codeword is checked: it shall be even in case of an even encoding scheme and odd in case of an odd encoding scheme. Due to obvious reasons, the parity codes have a code distance of 2, making them able to detect single bit faults and multiple bit faults affecting an odd number of bits. Since it cannot be identified which bits are affected, no faults can be corrected. Parity codes – thanks to their simplicity – can be found in various applications: processor registers, communication protocols, etc. Parity code is a separable code.
- *Double-inverted coding*, as the name suggests extends the data word by its complementary. In case of a data word of length n the codeword has the length of $2n$. For example the 8-bit data word $[0\,1\,1\,1\,1\,1\,1\,0]$ would be extended by the code bits $[1\,0\,0\,0\,0\,0\,0\,1]$. The drawbacks of double-inverted coding are the length of the codeword and the lack of fault correction capability.
- *CRCs* are generated by linear operations performed using binary generator polynomials (e.g. CRC-12: $1 + x + x^2 + x^3 + x^{11} + x^{12}$). The generated CRC is attached to the data word, hence the CRC is a separable code. CRCs are often used due to their great fault detection and correction capabilities. The generator polynomial $p_g(x)$ can detect $deg(p_g(x))$ bit failures, where $deg()$ denotes the highest exponent of the polynomial. The application examples of the CRC include not only the volatile and non-volatile memory of modern microcontrollers, but also CAN and FlexRay communication protocols. For common generator polynomials (e.g. the CRC-12 in the example above), there are off-the-shelf SW-libraries and even integrated HW-solutions available, making them very attractive.
- *AN-codes* are the most simple arithmetic codes. The coding scheme corresponds to the one shown in figure 3.4: the data words are multiplied by an integer C. To decode the codeword, it is simply divided by C. The constant C determines the code distance and therefore the fault detection and correction capabilities.

A very interesting application of coding is *coded processing*, shown in figure 3.6. SW using this feature performs arithmetic operations twice: one instance is calculated normally, while the second instance uses coded operands and arithmetic operations adapted to the coding scheme. The two calculation results are then compared. The architecture shown in figure 3.6 can detect but cannot correct faults. For fault correction purposes, a higher degree of redundancy is required.

Time redundancy

Time redundancy is based on repetition of computation or data transmission [Dub13]. The n instances of the computation results or the transmitted data are then compared. The advantage of time redundancy over structural or information redundancy is that no extra (physical or logical) elements are needed. The obvious draw-back is the elongated procession time.

Typical time redundancy architectures are shown in figure 3.7 [Dub13]. These architectures are practically identical to the static redundant HW and system architectures. The only difference is that data obtained at different time needs to be buffered before comparing or voting.

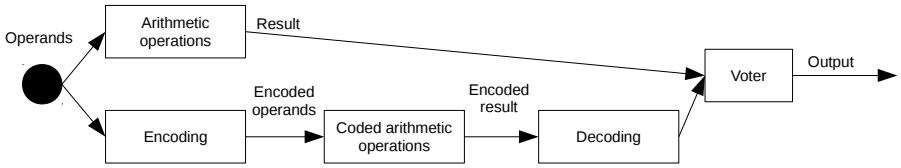


Figure 3.6: Principle of coded processing

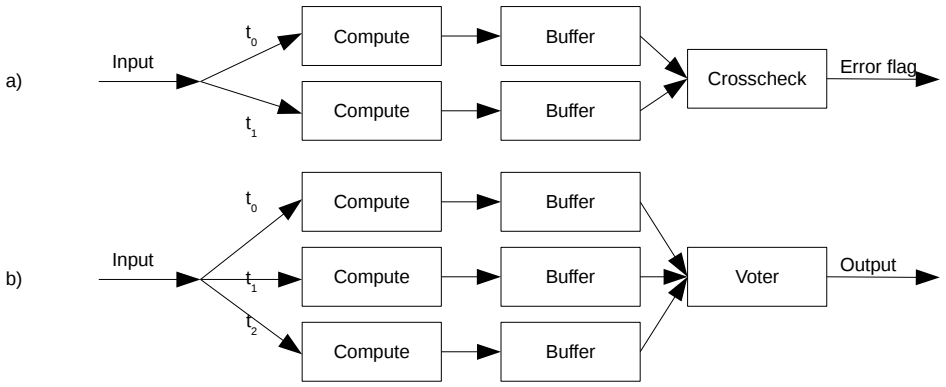


Figure 3.7: Common time redundant architectures, a) with fault detection capability, b) with fault correction capability

Time redundancy alone can cover transient faults only. If a fault is permanent or even systematic, the repeated data will be equal to the original one, leading to a false positive comparison result. By extending time redundancy with information redundancy, permanent faults can be detected as well. Typical solutions are:

- *Alternating logic:* this logic practically applies the double-inverted coding. The second transmitted instance is the inverted version of the first one. By comparing them, stuck-at faults of the transmission lines or logic circuits can be detected.
- *Recomputing with modified operands:* To detect permanent failures in computation logic, the computation is performed with modified operands for the second time. This modification can be shifting, or swapping the upper and lower part of the data, for example.

SW redundancy

Since SW is also prone to (systematic) failures, its fault tolerance shall be handled as well. [Dub13] explains the most common SW fault tolerance techniques. What is important to understand, is the completely different nature of HW and SW fault tolerance. HW fault tolerance is mainly applied to cope with random HW faults or faults caused by environmental factors. If transient faults in the HW can be tolerated, the device can operate without any functional or performance degradation after the fault has diminished. In case of permanent HW faults, the device can continue operation until the unit is repaired or replaced. SW, on the other hand, has only systematic faults. Due to sophisticated processes and extensive SW verification in safety related domains, the number of residual faults in the SW is minimal. These faults, as it is assumed, are caused by an unfortunate and unusual set and sequence of inputs, caused by an unusual use-case. Based on this assumption, these faults can be regarded and handled as transients, and are therefore typical subjects for fail-operational behaviour.

There are two main fault tolerant SW architectures: *multi version* and *single version*. Single version solutions are based on fault detection and subsequent fault containment. Typical fault detection

techniques are [Dub13]:

- *Acceptance tests* check the result of the SW function in focus and report any found error. They are only attractive if their computing time and data storage needs are significantly lower than those of the checked function. If not, multi-version techniques may be more simple.
- *Timing checks* do not focus on the results of a function but on its execution timing. Typical solutions are checkpoint based program flow monitoring or watchdog functions. These monitor the correct timing and execution order of the SW units.
- *Coding checks* can be used for SW using coded processing (see section 3.1.2), to detect systematic faults in the primary calculation path.
- *Plausibility checks* focus on basic data properties, as range or rate to detect severe calculation faults. The effectiveness and whether these checks make sense, are questionable. Severe systematic SW fault as overflows, range or rate violations can be detected with a very high probability during the development phase using static and dynamic verification methods.
- *Structural checks* are based on known structural properties of data. For example, the intended length of a list of an analogue-digital interface buffer over a specified time can be monitored to detect timing or program execution faults.

But in case of single version techniques, detecting the fault is only the first phase. The more challenging part is dealing with this detected fault. There are two main types of methods: *fault containment* and *fault recovery*.

Fault containment is based on encapsulation of SW units. Only by identifying and isolating faulty units is it possible to prevent further fault propagation. Hence, these containment measures are mainly based on architectural means and/or architectural control mechanisms:

- The most basic technique is *modularization*, decomposing the SW into modules, eliminating shared resources (e.g. RAM) and limiting or monitoring the communication between these modules.
- A more stringent measure is SW partitioning, a technique commonly used in mixed-criticality SW, to prevent SW modules with lower integrity impairing safety related ones. This technique is based on the independence of functionally independent modules, relying on control modules supervising the function execution and communication.
- The most stringent technique is *system closure*, where every action in the SW is authorized and monitored by a master supervisor function. This leads to a highly regulated, strict environment, where all intended and unintended interactions between the modules are managed and monitored.

Fault recovery can be a standalone technique (to react on detected faults), or can be combined with fault containment. Fault recovery aims the return to normal operation after a fault has been detected and optionally contained. The prerequisite of fault recovery is that the fault or abnormal condition is detected, the program execution is interrupted, and a special reaction is triggered. This task is done by *exception handling* [Dub13]. After exception handling halted the SW execution, it triggers a recovery mechanism. The most common single-version recovery mechanism is *checkpoint and restart*. This technique is based on the assumption mentioned at the start of this section: SW faults are mainly caused by abnormal system conditions, and are therefore to be treated as transients. The program execution state (e.g. register, memory values) is stored in the *checkpoint memory* at given *checkpoints* during the SW execution. If a fault detection mechanism detects an anomaly, the execution is interrupted and returns to the previous checkpoint, by reloading the state information from the checkpoint memory. Then the SW is executed again. Since SW faults are usually caused by unusual input value combinations or sequence, it is beneficial to slightly change the data reloaded from the checkpoint memory.

Multi-version techniques [Dub13] are very similar to the architectures used for HW and systems (see section 3.1.2). The most widespread solutions are *recovery blocks*, *N-version programming* and *N self-checking programming* (figure 3.8).

Recovery block is practically a multi-version checkpoint and restart technique. During normal, fault-free operation, the primary module produces the output. If a fault is detected by the fault detection block at the output of the switch, the content of the checkpoint memory is reloaded, and the next module is executed. The advantage of this technique is that the parallel modules are diverse redundant. Hence it is very unlikely that the next trial will fail as well. This architecture can tolerate $n - 1$ faults. A

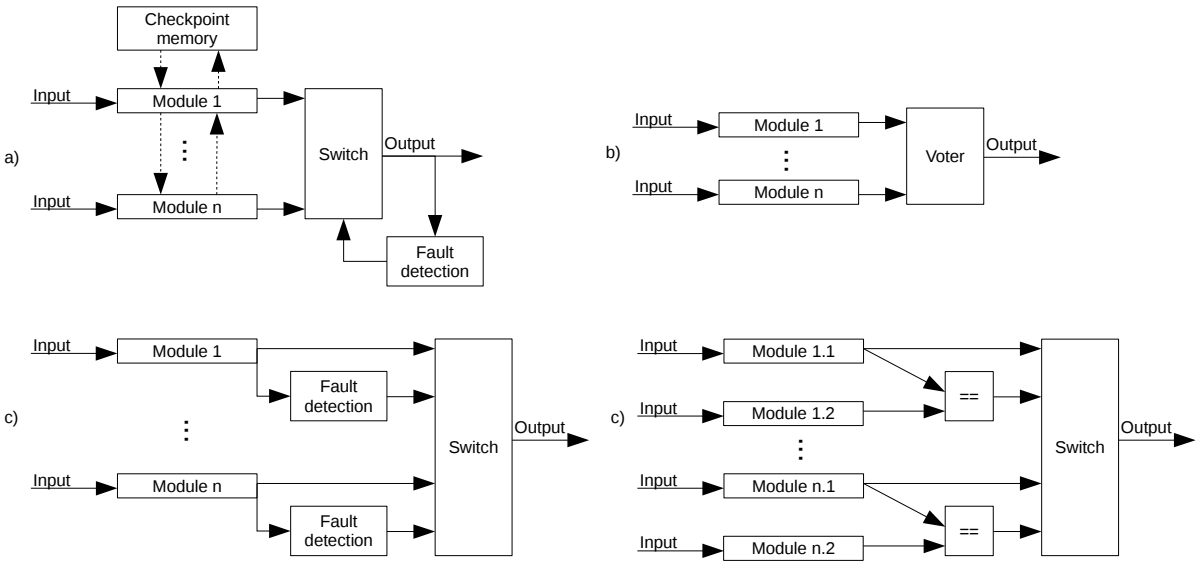


Figure 3.8: Multi-version SW redundancy, a) Recovery blocks, b) N-version programming, c) N self-checking programming using fault detection, d) N self-checking programming using cross-check functions

disadvantage of this solution (compared to n-version programming) is its reliance on the fault detection mechanism. These fault detection functions may be very complex in some cases.

N-version programming is the SW equivalent to NMR. N diverse implementations of the same function are in parallel. Their inputs are routed by a voter to the output. The voter is more simple than the fault detection mechanism of the recovery block architecture. A further advantage of this technique is that no recovery time is needed in case of a fault. A disadvantage is, that in case of n modules, only $n-2$ faults can be tolerated.

N-self-checking programming is a combination of recovery blocks and N-version programming. This technique is either based on fault detection mechanisms or by cross-checking pairs of modules. Different module versions are executed in parallel. The outputs of these modules are checked (either by a fault detection function or by cross-checks). The switch routes the result of the highest ranking correctly working module to the output.

It is important to emphasize, that design diversity and freedom from interference are of paramount importance for multi-version SW redundancy techniques. Only by this diversity can it be ensured that common mode faults of the n modules can be ruled out. Design diversity in case of SW means at least different implementation of the same specification. Different implementation is a combination of different programming languages, different independent teams and different tool-chains. In some cases it is even beneficial to use different specifications for the parallel modules. Freedom from interference on the other hand is necessary to prevent that parallel modules impair each other.

Analysis of dependent failures in the domain independent literature

[Jur12] presents a generic analysis technique including common cause failure analysis. This paper identifies the following potential causes of common cause failures:

- Physical source (internal or external)
- Functional (internal or external)
- Process

Interesting and agreeable aspects of this classification include that it is admitted that common cause failures can be *both* internal and external. Some papers state the contrary (e.g. [Cha09]), leading to inconsistencies in the analyses (see 3.5.2 for further details). This paper also states that the quantification

of common cause failures (i.e. their failure rates and diagnostic coverages) is very inaccurate. Therefore qualitative arguments have to be used, based on technical and organisational measures.

[Ern15] introduces an innovative, yet very complex method for the dependency analysis of safety related systems. This analysis is performed in the following three steps:

1. Identify dependencies
2. Quantify dependencies
3. Mitigate dependencies

The quantification in the second step is meant to support the evaluation of the importance of dependencies. This paper offers a multi-layered architecture description of HW and SW. Based on this architecture, possible dependencies can be identified. The dependency types provided by this paper are the following:

- Environmental
- Functional data
- Functional timing
- Unintended data
- Unintended timing
- Thermal dependencies
- Electromagnetic
- Mechanical

One very important statement of this paper is that there is one type of dependency that cannot be found by analysing architectural design descriptions: unintended data and timing dependencies. This statement is also supported by [Gan74]. This latter paper offers a method for the common mode analysis of nuclear facilities. In this paper it is pointed out that coupling factors and common causes are things that the designer didn't think of. Therefore they cannot be identified within the limits of design drawings.

3.2 Fail-operational aspects in avionics

3.2.1 Fail-operational aspects in avionics standards and guidelines

Fail-operational aspects in the main avionics safety standards

According to [SAE10] the structure of avionics safety standards is the following:

- *SAE ARP 4761* covers the safety assessment process, defining functions, failure and safety information of avionic systems, including specification and analysis techniques.
- *SAE ARP 4754/A* focuses on the development on system level
- *DO-254* is concerned with the development on HW level
- *DO-178C / ED-12C* covers the development on SW level

The [SAE96] addresses some topics related to fail-operational behaviour, although not explicitly. Fault tolerance as such is mentioned several times, but no requirements on fail-operational architectures or technical solutions are defined. The proposed architecture for a brake system controller in the Annex L, in the safety assessment example, is a duo-duplex architecture, though. This shows that fault tolerance is a common attribute of avionic systems. Although only implicitly related to fault tolerance, the [SAE96] needs to be pointed out. Contrary to other standards, this one provides a relatively accurate and specific guidance to common cause analyses. The objective is to prove independence where it is required by the system architecture, with special focus on common cause faults. It is basically subdivided into *zonal safety analysis*, *particular risk analysis* and *common mode analysis*.

Zonal safety analysis ensures that equipment installation fulfils the safety requirements. For this purpose each zone of the aircraft is analysed with respect to basic installation, interference between systems and maintenance errors.

Particular risk analysis focuses on external impact factors that could invalidate the claimed independence, including:

- Fire
- Fluid leakage
- Snow
- Bird strike
- Tread separation from tire
- Lightning

Common mode analysis is performed to verify that failures connected (i.e. combined) by AND-gates in the FTA are actually independent. For this purpose, design, manufacturing and maintenance errors are analysed ([SAE96]):

- HW or SW failure
- Production or maintenance failure
- Situation related stress
- Installation failure
- Requirements error
- Environmental factors (e.g. temperature, vibration)
- Failures of common external sources

Additionally, Annex K defines a common mode analysis process, consisting of the following steps:

1. Establish specific checklists
2. Identify requirements for common mode analysis
3. Analyse the design to ensure it meets the requirements identified above
4. Document these results

The proposed method is based solely on checklists and is performed on the aircraft level. These checklists are based on previous experience and example data. Focus is on areas where redundancy or independence is required.

Surprisingly, the [SAE10] does not contain any references, guidelines or requirements related to fail-operational behaviour or fault tolerance.

The [EUR99] addresses fault tolerance in the SW implicitly. (Note, that for this thesis the [EUR99] with the extending paper [Potwn] was used to yield the updated ED-12C standard). The only fault tolerance related content is to be found in [EUR99]: for SW architectures this standard considers *partitioning*, *multi-version programming* and *safety monitoring* as safety measures. For partitioning even potential coupling mechanisms (e.g. data coupling, HW resources, control coupling) are listed.

Fail-operational aspects in the NASA Software Safety Guidebook

The [NAS04] addresses fail-operational behaviour, fault and failure tolerance explicitly. It defines fault tolerance as the ability of a system to remain safe and operational even in the presence of faults. Interesting aspect of this guideline is the distinction between fault and failure tolerance. This distinction follows the fault and failure definition of the [ISO11]. Hence the main difference lies in the level of the mechanisms realising the fault/failure tolerant behaviour [NAS04]:

- Fault tolerance is based on low-level mechanisms that prevent faults to lead to failures, e.g. an error detection-correction code repairing corrupted cells in the memory of a microcontroller.
- Failure tolerance, on the other hand, is based on high-level safety mechanisms that prevent failures to lead to an unsafe state, e.g. three controllers in a TMR architecture.

The guideline emphasizes this difference in various occasions, pointing out that it has to be defined whether fault- or failure tolerance is targeted, or both. From the system perspective, these are architectural level decisions. A system that is rather trimmed to fault tolerance (in this sense), is equipped with various low-level safety mechanisms, intended to control specific faults. When designed to be failure tolerant, the tolerance mechanisms are rather to be found on the system architectural level. These mechanisms are designed to handle high level failure modes, independently of their causes (i.e. the faults behind them). The downside of failure tolerance is the higher cost due to the system level

redundancy. It is important to emphasize, that due to the recursive definition of faults and failures in the [ISO11], the distinction between the terms fault and failure tolerance will not be used in this thesis.

In both cases [NAS04] defines the degree of fault/failure tolerance as the number of faults/failures that can be present without the system getting unsafe or losing the function. The guidebook even defines the required degree of failure tolerance, depending on the severity of the related hazard:

- In case of *catastrophic hazards* (i.e. hazards leading to loss of human life), two failures have to be tolerated.
- In case of *critical hazards* (i.e. hazards leading to severe injury), one failure has to be tolerated

As all other standards and guidebooks investigated in this thesis, the [NAS04] also requires and describes various fault tolerance techniques, including built-in self tests, voting, N-version programming, recovery blocks and redundant architectures.

3.2.2 Fail-operational design in state-of-the-art avionics systems

Air- and spacecraft control functions shall remain operational even in the presence of faults, fail-safe behaviour is seldom an option. The NASA Space Shuttle control system has been developed in the '70-s, and already incorporated a highly sophisticated diverse redundant fail-operational architecture to meet stringent safety requirements [Bla09]. Therefore, the aerospace industry gained vast experience on this matter, that is for sure a great starting point for the automotive domain. This chapter investigates typical technical solutions and transfers their applicability to passenger cars. [Too09] and [Flü10] describe basic avionic (i.e. electronic systems used in aircraft, artificial satellites, and spacecraft) concepts and typical architectures. [Yeh96] and [Bla09] give insight into the avionic architecture of two prominent fly-by-wire airborne vehicles: the Boeing 777 and the NASA Space Shuttle, respectively.

There are three main types of avionics architectures [Flü10]:

- *Centralised* architectures are based on one redundant central controller being in charge for all the peripherals. The advantage of this solution is that only one redundant controller has to be developed. But due to the lack of reconfiguration capabilities, the reliability and safety requirements on this central controller are high.
- *Distributed* architectures do the exact opposite: each subsystem, actuator and sensor has its own control unit. These subsystems are interconnected by a redundant communication bus system. If one subsystem fails, its tasks can be taken over by another one. The disadvantage of this architecture is its higher cost and weight. Note, that an automotive electronics architecture is a textbook example for a distributed solution.
- *Hybrid* architectures combine the centralised and distributed approach. The avionics systems are partitioned into functional groups. Each of these functional groups has a central controller. These functional groups are connected by a redundant communication bus.

In the past decades various redundancy concepts have been evolved in avionics. Figures 3.3 b), 3.9 and 3.10 show three typical solutions [Flü10] [Yeh96] [Bla09]:

- *Duo-duplex* architecture has been already explained in the section 3.1. This architecture is used by Airbus.
- *Quadruplex* architecture has been used in the NASA Space Shuttle for critical units, along with its related data buses and even the hydraulics. Quadruplex redundancy is based on a 3 out of 4 voting. It degrades to a TMR when a failure occurs and then the third failure leads to a fail-safe shut down. The voting is also redundant, making these systems very robust against single point faults. Faulty units are isolated by switches minimising the probability of cascading failures.
- *Triple-Triple Redundancy (TTR)* architecture is used by the Boeing 777, and can be regarded as a further enhancement of the TMR. Here, there are three homogeneous redundant flight controllers (left, center, right) consisting of three diverse redundant lanes inside. In addition to that, one flight controller is spatially separated from the other two. In the Boeing the three lanes in the flight controllers had the same SW but compiled with different compilers. This TTR approach was applied for the flight controllers, to the electrical power supply, communication and the hydraulics (like in the Space Shuttle).

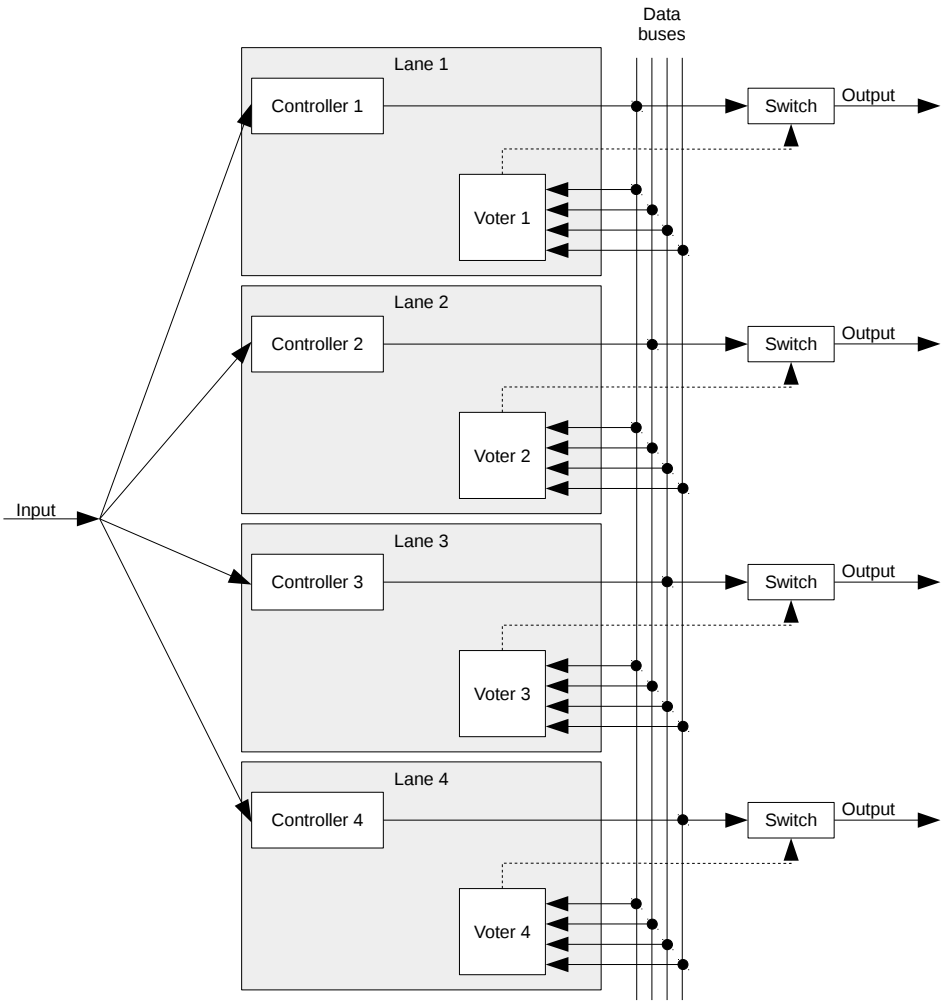


Figure 3.9: Quadruplex architecture

The two latter solutions rely on calculation synchronisation in order to prevent false alarms by the voters. Only by synchronising the program execution (which can be tricky for diverse redundant SW) it can be ensured that the values compared by the voters belong to the same time stamp.

In the following sections, different application areas of redundancy will be investigated.

Actuators

The intrinsic redundancy of the control surfaces of an aircraft is a big advantage over passenger vehicles. To design redundant sensor and controller systems is easy compared to the challenges of developing diverse redundant actuators. In most cases, packaging, control strategies and the human-machine interface are limiting factors. In case of aircraft, for example, the large surface of the wings allows the designers to include primary and secondary control surfaces. In this case, faulty actuators can be decoupled from the system. The remaining system has lower control performance but is still operational and safe.

Communication bus systems

Avionic bus systems connect aircraft systems (i.e. controllers, actuators and sensors) and are therefore integral parts of the architecture. In the past decades various bus systems (e.g. ARINC 429, ARINC

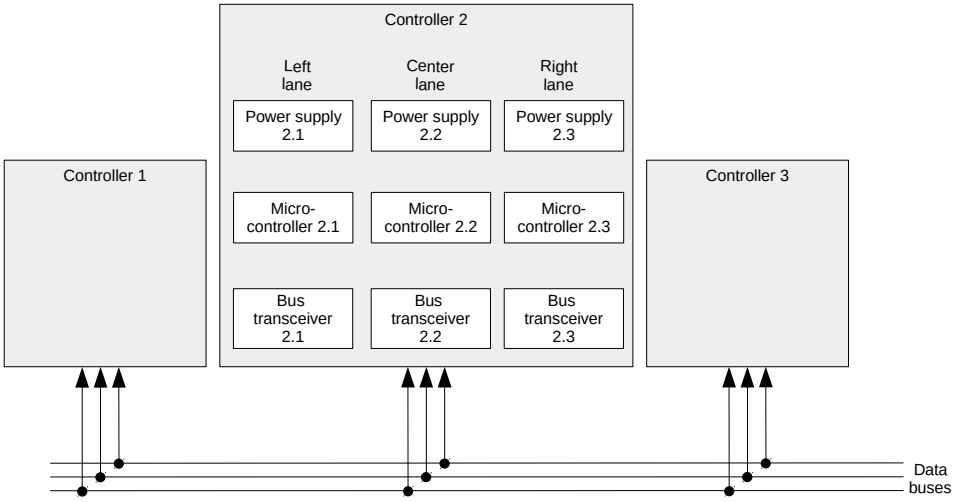


Figure 3.10: TTR architecture

629, Mil-STD-1553, TTP [Flü10]) have been developed and used in the field. The trend was similar to that of passenger vehicles in the '90-s: The increasing number of controllers and of their functions demanded a different communication approach. [Flü10] introduces the most important avionic bus systems thoroughly. There are some important characteristics of avionic bus systems that need to be pointed out for the subject of this thesis.

For safety critical redundant systems *deterministic timing* is of paramount importance. Protocols, like Time Triggered Protocol (TTP) and ARINC 664-7 solve this issue by introducing bus synchronisation and pre-determined time-slots for the bus messages. This way each system connected to the bus knows when to send and when to receive signals. This way it can be prevented that signals go missing due to high bus load, a failure mode that is difficult to handle in fault-tolerant systems. An additional advantage is that the "age" of signals used for voting is well-known, making voter designs much easier. Note, that the automotive FlexRay protocol is also a time-triggered protocol [Boro8].

Another interesting aspect of the fault-tolerance is the concept of *bus guard's*. All bus systems with a single physical layer (e.g. one twisted shielded pair of cable) are prone to the "babbling idiot" problem. "Babbling idiot" denotes the failure mode where one communication node blocks the bus access by continuously sending messages. To handle this issue, the TTP bus uses bus guards to decouple nodes that would disturb the communication on the bus they are connected to. Note, that this is a classic fail-silent strategy for bus communication.

Pilot warning

There are several differences between pilots and drivers, but the most important maybe is their training. Due to this reason, their role in the failure handling concept is designed completely differently. Where 99% of all drivers could possible not control an oversteering vehicle, pilots are trained to handle failures of the flight control systems and to bring aircraft back safely. The so called computer annunciation matrix of the Space Shuttle is a great example [Bla09]: the display shows the results of the voting in the redundant voters. The crew than can decide which controllers to deactivate to continue the mission. In later mission phases the crew is allowed to, and able to, re-activate controllers that failed before.

Qualitative and quantitative fault tolerance targets

One of the first questions that arise when discussing fail-operational automotive systems is: how many failures shall be tolerated and which safety integrity is to be expected from the remaining system after a failure?

In the Space Shuttle, a fail-operational-fail-operational-fail-safe strategy has been implemented [Bla09]. This means that two faults can be tolerated. After that, if a third failure occurs, a safe state is entered. This architecture lead to the quadruplex architecture shown in figure 3.9.

For the Boeing 777, qualitative and quantitative fault tolerance targets have been defined [Yeh96]:

- *Quantitative* probability requirements are 10^{-10} per flight hour for cases where all controllers are operational and even for scenarios where one controller fails.
- *Qualitative* requirements define that no single point faults (regardless of their failure rate) are allowed that would lead to the malfunction of a controller.

Power supply

One of the most intriguing features of aircraft is the sophisticated, but still very straightforward, power supply architecture. Its fault tolerance concept is based on the following three pillars: *multi-level redundancy*, *prioritisation of consumer loads* and the *fault isolation* capability. Small aeroplanes of the general aviation use simple 28V DC bordnets. For large aircraft AC bordnets offer a better volume to power ratio. These AC generators have either a fixed (400Hz) or a variable frequency (Airbus 380: 360 - 800Hz).

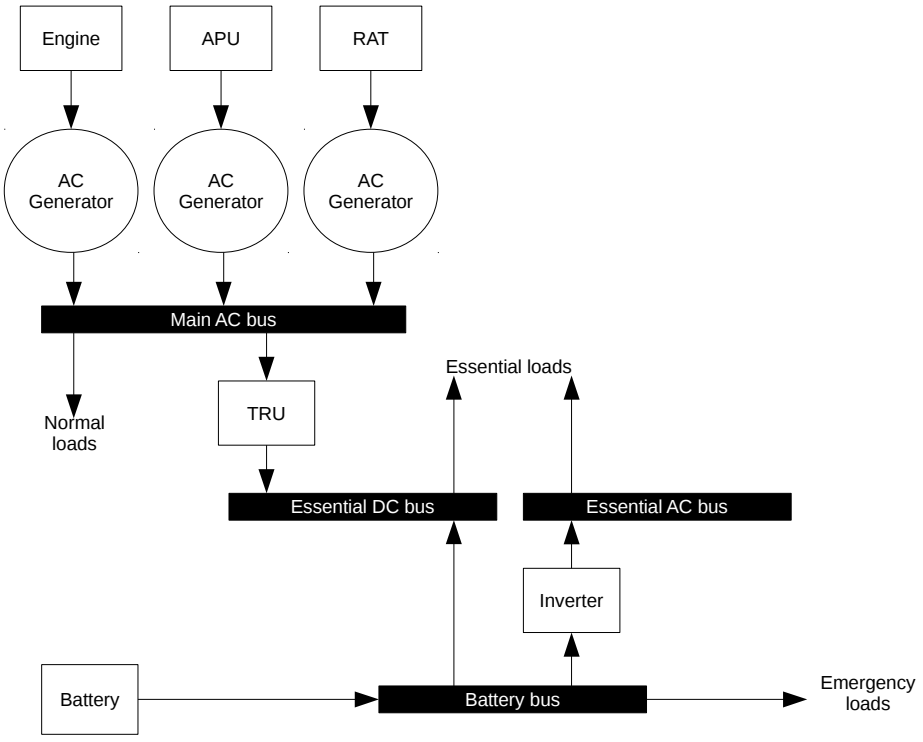


Figure 3.11: Power supply architecture of a large aircraft [To009] [Flü10]

Figure 3.11 shows the basic elements of the power supply architecture of a large aircraft. The key for the fail-operational power supply architecture is the categorisation of the consumer loads and the allocation of these to the busbars (i.e. wiring distributing electric power to consumer loads):

- *Emergency loads* are connected to the last backup: the battery bus. These loads are devices required for safe landing.

- *Essential loads* are connected to the essential AC and DC busbars. These loads are systems required for the safe operation of the aircraft.
- *Normal loads* are connected to the main AC bus. These loads are non-critical loads, like cabin lights.

Based on this categorisation, a multi-level redundancy concept is implemented. Both the DC and the AC busbars can be supplied from various sources. If one of the related devices fails, it can be disconnected and isolated from the rest of the power supply net by switches, relays and circuit breakers. These devices are not shown in the figure. Each architecture is different, but follows the same principles:

- The primary sources of AC power supply are engine driven *generators*. Obviously, if more engines are available, then each engine drives its own generator.
- The primary sources of DC power supply are devices called Transformer Rectifier Unit (TRU). These convert AC to DC and reduce their voltage levels.
- In case of a generator failure the AC busbar can be supplied from various sources:
  - *Inverters* converting DC power from the DC busbars to AC power
  - An *Auxiliary Power Unit (APU)*, an extra turbine dedicated to drive a generator if the engine-driven generators or the engines would fail can produce AC power.
  - A *Ram Air Turbine (RAT)*, also known as air-driven generator can step in if neither the engine-driven nor the APU-driven generators work. The RAT is an air-driven device that can generate power from the airflow when deployed. Figure 3.12 shows an example.
- If a primary source of DC power fails, backup can be guaranteed either by a parallel generator and TRU, or as emergency backup, from the battery.

Figure 3.12: Ram air turbine of the Boeing 757 (source: [img16])

Related to the power supply concept is the redundant fuel pump systems applied on general aviation aircraft. In these aeroplanes the primary fuel pump is driven by the engine, while backup is provided by electrical boost pumps [Too09].

Analysis of dependent failures in avionics systems

As already mentioned in section 3.2.1, the [SAE96] requires common cause analyses with special focus to common cause faults. [Bal07] provides a very thorough description of a common mode analysis method. First, a list of possible common mode faults is defined:

- Requirement specification errors
- HW or HW development error
- Random HW failures
- Production errors

- Repair errors
- Operational errors
- Installation errors
- Environmental factors
- Common external source faults

The proposed analysis itself is based on the evaluation of each AND-gate of the FTA, and is performed in the following steps:

1. Select all AND-gates from the FTA related to catastrophic events.
2. Classify these AND-gates.
3. Perform the detailed analysis for category 1 AND-gates, including the definition of a defense mechanism.
4. Provide independence justification for the other three categories.

The essential part of this analysis technique is the categorisation of the AND-gates. This allows the analyst to focus on the most important multiple point faults and to reduce the effort to a reasonable level. The following categories have been defined by [Balo7]:

- *Category 1* AND-gates are used to model redundant channels or the command/monitor of the ECU architecture, involving elements with high complexity. This category is in the foremost focus of the analysis. For this category, a full-scale common mode analysis shall be performed.
- *Category 2* AND-gates are used to model faults of mechanical HW and the loss of SW-or programmable electronics based diagnostics. In this case, the independence is obvious due to the completely different technologies used in the two branches of the tree. Therefore, the analysis effort can be limited here.
- *Category 3* AND-gates are used to model redundant HW designs. For this category, a limited common mode analysis is sufficient, focusing on concept and design similarities; manufacturing; installation; maintenance and environmental influences.
- *Category 4* AND-gates are used to model a sequence of events, where the faults do not occur simultaneously but in a certain sequence. This category can be covered by simple qualitative risk mitigation, as long as the two events are of different nature (e.g. random HW faults and fire) or if the two systems are different conceptually.

[Mos95] presents the results of the dependent failure analysis of the Space Shuttle. The presented method redefines the root clause classification strategies and defines a list of defense mechanisms. These root cause classes are very spacecraft-oriented, therefore their applicability to automotive systems is very limited. The interesting part of this paper is that the authors had the opportunity to use real-life in-flight accident data. Based on this data, the following leading root causes have been found:

- Design flaws (17% of the cases)
- Contamination (11%)
- Defective calibration or test procedures (9%)
- Moisture (6%)

These issues were responsible for 43% of all common cause failures. The accident data provided also information about the coupling factors. The leading ones were the following:

- Same part design (27%)
- Same system design (9%)
- Same component location or environment (11%)
- Same supporting systems (9%)
- Same component calibration characteristics (8%)
- Same system interconnections (6%)

These two sets of data underline the importance of *design diversity*.

[Pen12] presents a reliability analysis technique focusing on common cause failures, applied on satellite systems. The authors of this paper propose similarly to [Balo7] an FTA-based analysis. Here,

the emphasis is on a checklist- or analysis-based identification of common-cause failure component groups. Based on these groups, the common cause base events are identified using the FTA. The key point is, according to this paper, to identify the coupling factors, which can be categorised as follows:

- Engineering factors
  - Same or faulty design
  - Same hardware
  - Same manufacturing process
- Usage factors (focusing mainly on installation and operation)
- Environmental factors

Comparison of automotive and aerospace systems

[Neno7] provides an interesting comparison of the automotive and aerospace domains. It highlights the following main differences:

- Differences in the *severities* of accidents due to the probably higher number of affected persons,
- Shorter *necessary time to reach the safe state* in case of automotive systems: a passenger vehicle can be parked alongside the road, while an aircraft has to finish the flight and land safely.
- Higher *production volume* of passenger cars,
- Higher *financial pressure* in case of passenger vehicles due to the higher volumes,
- The *mission time* for aircraft is twice as long as that of passenger vehicles (two hours versus one).
- Better and more thorough *maintenance* of aircraft,
- Higher *complexity* of the road traffic,
- *Training* of drivers compared to that of pilots.

All these differences shall be taken into account of when transferring aircraft architectures into passenger cars.

3.3 Fail-operational aspects in agricultural applications

3.3.1 Fail-operational aspects in the [ASI14]

The [ASI14] is the german version of the ISO 25119. Considering the features of modern agricultural vehicles (see 3.3.2), it is not surprising that fail-operational behaviour is addressed explicitly. Part 1 of this standard defines the term *safe state* the same way as the [IEC10]. The note below this requirement even states explicitly that this safe state can be the normal intended operational mode, a backup operational mode, or a passive state. Note, that the former two operational modes are fail-operational states. In [ASI14] it is even required that the system designer shall consider if the safe state is reached by shutting down or by maintaining the function.

The AgPL (i.e. the safety integrity level of this standard) is defined by four parameters: the HW category, the Mean Time to Failure (MTTF), the DC and the Software Reliability Level (SRL). The [ASI14] follows a different principle than the standards presented in the previous sections: for a certain AgPL the focus can be shifted to the HW or the SW. If the designer chooses a higher HW category, a lower SRL and/or MTTF is allowed.

The HW categories are shown in figure 3.13. The [ASI14] even defines the fault tolerance capability of each category:

- *Category 1* consists of an input-, a logic- and an output module. Due to the single channel architecture, fail-operational behaviour is not possible.
- *Category 2* consists, additionally to the elements of category 1, of Test Equipment and a related output module. The Test Equipment monitors the Logic and controls its own output to reach a safe state if a failure is detected. This category is also not suitable for fail-operational systems.

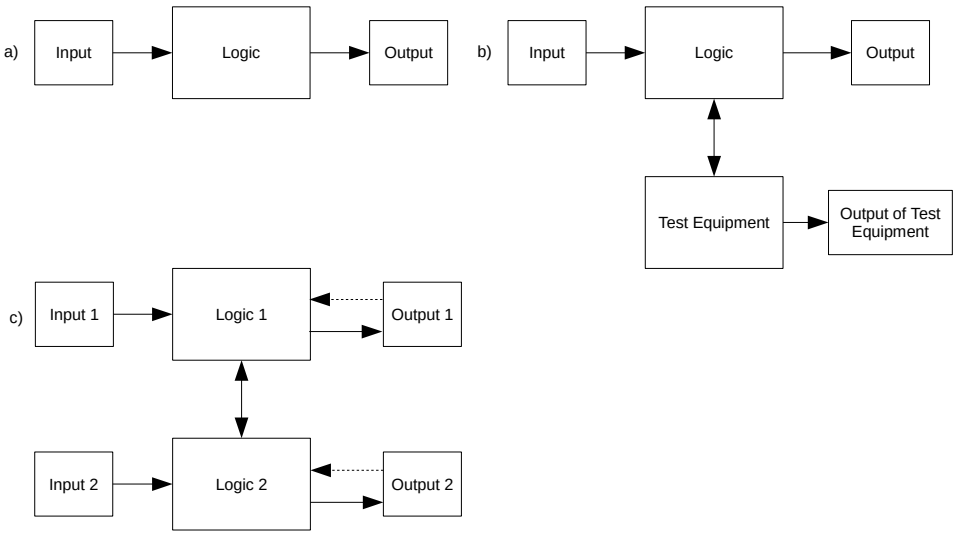


Figure 3.13: HW categories of the EN 16590, a) Category 1, b) Category 2, c) Category 3-4

- *Category 3* and *category 4* consist of two parallel paths, monitoring each other. The only difference between these two categories is the higher DC of category 4. Since this architecture has redundant input-, logic- and output modules, it can tolerate a single point failure. The standard even mentions explicitly that in order to do so, a redundant power supply is needed.

Note, that although this standard defines system architectures capable of tolerating one single point failure, these architectures are not mandatory. The main motivation of picking a higher HW category is to develop the SW in accordance with less string requirements.

Annex C of the standard lists typical diagnostic functions and their DC-s, among them some which are closely or remotely connected to fault tolerance. Examples include majority voter, tests by redundant HW, block replication, modified checksum for non-volatile memory. These diagnostic measures are very similar to the ones listed in the [IEC10] or the [ISO11].

An interesting point of the [ASI14] is its annex D, providing a very basic guideline for analysing common cause failures. It is based on a very basic checklist, evaluating typical common cause initiators. This analysis is required for categories 3 and 4, which is reasonable, considering their diverse redundant architectures.

3.3.2 Fail-operational design in state-of-the-art agricultural systems

Some modern agricultural vehicles are actually robots, performing their tasks autonomously [Han11]. Since robots have to outperform humans, their safety is of paramount importance. Due to these autonomous functions, fail-operational behaviour is a must. Unfortunately, the related literature gives very little insight into the fail-operational architecture of these vehicles, and their fault tolerant strategies and mechanisms. [Han11] shows the architecture of a typical agricultural robot, consisting of:

- Thermo camera
- Machine vision unit using stereographic cameras
- Laser-based radar
- GPS unit
- Speed radar
- Inertial unit (measuring lateral and longitudinal accelerations, yaw-rate, etc.)

This architecture is very similar to that of aircraft equipped with autopilot, which is completely comprehensible considering the similarity of their functionalities.

[Mal14] introduces a general purpose ECU for agricultural vehicles, compliant with the ISO 25119. This paper also identifies the necessity of fail-operational behaviour based on the safe state, and not based on the criticality, like many other publications. Its proposed ECU design is an enhanced category 2 HW-architecture (see section 3.3.1). Unfortunately it is not clear from this paper, how fault tolerance is achieved.

3.4 Fail-operational aspects in railway applications

3.4.1 Fail-operational aspects in railway standards

The two standards that are relevant for railway cars are the [ÖVE99] and the [ÖVE12]. The former gives requirements and recommendations on the reliability, availability, maintenance and safety engineering of railway applications (infrastructure and vehicles likewise). The latter focuses on safety related SW for railway infrastructure and vehicles.

The [ÖVE99][4.8] states that fail-safe behaviour (and a related fail-safe concept) has been always applied for railway systems. The main rationale according to the standard, is the clearly defined safe state and the predictable behaviour by switching to it.

The [ÖVE12] does not contain any explicit information or requirement, for or against a fail-operational behaviour. Interestingly, though, [ÖVE12][table A.3] lists various SW safety mechanisms that are related to fail-operational behaviour and fault tolerance. These mechanisms are in many cases either not recommended or are rated neutral. Examples include error correction codes (not recommended), recovery blocks (recommended at higher SIL-s), forwards- or backwards recovery (not recommended), artificial intelligence - fault correction (not recommended), graceful degradation (highly recommended at higher SIL-s) and dynamic SW reconfiguration (not recommended).

3.4.2 Fail-operational design in state-of-the-art railway systems

The only modern book focusing on modern electric railway cars, [Filo4], does not include any information about the functional safety aspects of these vehicles.

In accordance with the fail-safe requirement of the [ÖVE99], both [Has93] and [Gui93] state that train-borne systems show fail-safe behaviour. The main advantage of trains compared to passenger vehicles is that they can be brought into their ultimate safe state by emergency braking. Therefore if a safety critical failure occurs, the main goal is to apply the emergency brakes. In case of the Shinkansen redundancy was applied for the train-borne systems to support detection mechanisms [Has93]. For the train control systems, fault tolerant architecture was chosen, in order to increase their reliability. For the various computers this train control system consists of, dual or triple redundant topologies are used. The dual redundant controllers are arranged in a dynamic redundant architecture with a cold standby. The triple redundant ones are arranged either as a classic TMR, or a dual redundant topology with a third standby unit. On the SW architectural level, the Shinkansen utilises graceful degradation to deactivate not safety related tasks in case of a failure. The Systeme d'aide a la conduite, a l'exploitation et a la maintenance (SACEM) system uses fault tolerance measures to improve its availability [Gui93]. A one microcontroller architecture is used extended by coded processing. The HW on the train is not duplicated but the computers of the infrastructure are arranged in a dynamic redundant topology with cold standby. Cold standby systems consist of n parallel channels ($n \geq 2$), where the backup ones are passive as long as the primary one is fault-free. In case of a fault in the primary channel, the next backup steps in.

[Joho9] presents the results of a research project at the Chalmers Lindholmen University College. This project focused on the development of a fail-operational brake-by-wire architecture for railway cars. The starting point of their work was a usual electromechanical brake of SAB WABCO. This system is a common fail-safe system with an automatically activated passive safe state. If the electromechanical system fails, a previously pre-loaded spring applies maximum braking force on all axles of the car.

Figure 3.14 shows the new fail-operational brake architecture, proposed by [Joho9]. This is based on a distributed architecture, where each axle has its own brake unit consisting of an ECU, redundant sensors, a fail-safe actuator and a voting unit. The brake units are connected by a communication bus.

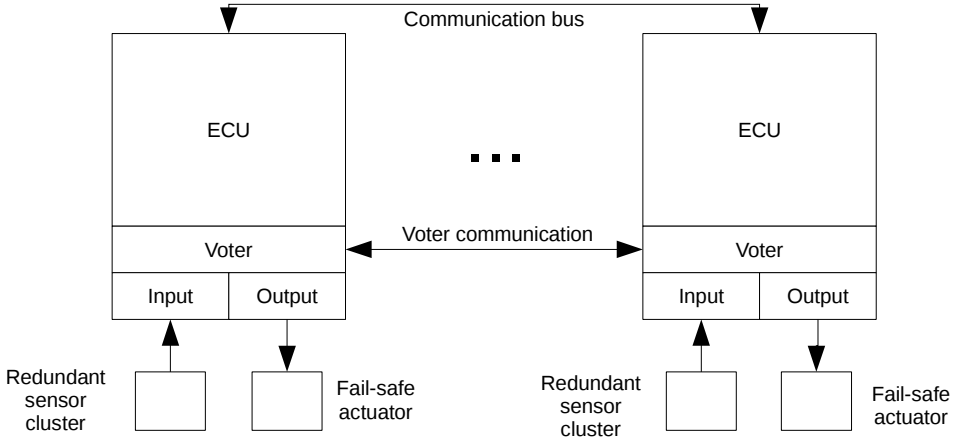


Figure 3.14: Proposed fail operational brake system architecture [Joho9]

The result is practically a distributed NMR architecture. But since this is the result of a research project, it gives us only a hint on the functional safety features of state-of-the-art railway vehicles.

3.5 Fail-operational aspects in automotive applications

3.5.1 Fail-operational aspects in the [ISO11]

For this thesis, the main safety standard in focus is the [ISO11]. Therefore, it is of paramount importance to investigate, how fail-operational behaviour is addressed in it. In advance: there are some explicit clues, but the [ISO11] is still a standard mainly meant for fail-safe systems. Exactly due to this reason, [Ede15] even asks if a new standard is necessary. The following sections give brief overview of the points in the standard, where fail-operational behaviour or related issues are addressed. In chapter 4, a detailed investigation is performed, coupled with the evaluation and comparison of typical fail-operational HW and SW architectures from the viewpoint of the [ISO11][part 5] and [part 6]. In this chapter, necessary extensions will be given, to cope with fail-operational systems.

Fail-operational aspects in the definitions of the [ISO11]

Since the [ISO11] is based on the [IEC10], it is no surprise that their *safe state* definitions are identical. As already mentioned in section 2.2, this definition doesn't rule out fail-operational behaviour. The note related to the definition even explicitly states that it can be the intended or degraded operational mode as well.

The [ISO11][part 1: 1.22, 1.13 and 1.14] define the terms *dependent failure*, *cascading failure* and *common cause failure*, respectively. As already shown in section 2.2, these are important fundamentals to the dependent failure analysis of independent redundant functions.

The terms *emergency operation* and *emergency operation interval* [ISO11][part 1: 1.34 and 1.35] are very useful for fail-operational systems. The former denotes a degraded functionality between the occurrence of a fault and the transition into the safe state. The latter designates the interval where emergency operation is needed. Both terms are related to the warning and degradation concept, defined in the concept phase.

Fail-operational in the concept phase [ISO11][part 3]

The [ISO11][part 3: 7.4.4.5] requires that a safe state shall be defined for each derived safety goal. In accordance with the definition, this can be also an active state. The standard even gives the example,

that the safe state can be the functionality maintained over a defined time.

Continuing that, in [ISO11][part 3: 8.2] the standard states that the functional safety concept addresses fault tolerance mechanisms and the warning and degradation concept. Later in [ISO11][part 3: 8.4.2.3] it is required that "functional redundancies (e.g. fault tolerance)" are considered when defining the functional safety concept. Afterwards, [ISO11][part 3: 8.4.2.3] requests that an emergency operation shall be specified if the safe state cannot be reached by immediately switching off the system.

Fail-operational in system development [ISO11][part 4]

It is obviously of interest, how the requirements related to the technical safety concept consider fail-operational systems. The [ISO11][part 4: 6.4.2.3] clearly addresses both fail-safe and fail-operational systems. It requires, that for each safety mechanism that supports the system to reach or maintain a safe state, the following shall be defined:

- the transition into the safe state (i.e. how the actuators shall be controlled to do so)
- the FTTI
- the emergency operation interval, if the safe state cannot be reached immediately
- means to maintain a safe state

With the exception of the second point, these apply to fail-operational systems as well, the latter two even exclusively.

Fail-operational in hardware development [ISO11][part 5]

Regarding the HW development processes, [ISO11][part 5] is also applicable for fail-operational HW. Hence, focus is directed rather on specific technical concepts of this part. Although it's not explicitly and exclusively meant for fail-operational systems, the importance of the LFM (see section 2.4.4 for details) needs to be underlined. This metric can reflect weaknesses of redundant architectures (which are a must for fail-operational behaviour) related to latent faults.

The [ISO11][part 5: 9.4.2.3] requires that in case of dual point faults, the exposure duration is considered. The related note even defines what shall be included in this exposure duration:

- multiple point fault detection interval (if there is a safety mechanism that can detect the first fault) or the vehicle lifetime (in case of latent faults)
- maximum duration of an ignition cycle (if the driver is requested to stop the vehicle)
- average time interval until the vehicle is brought to a workshop for repair (if the driver is requested to visit a workshop)

Obviously, this requirement already considers the interaction between the exposure duration and the warning and degradation concept.

Annex D of [ISO11][part 5] contains the usual tables (as in other standards), containing typical detection mechanisms and their estimated DC. These tables contain numerous detection principles that are related to fault tolerance techniques. Examples include majority voting, SW diversified redundancy, HW redundancy, parity bit, checksum, block replication, CRC, information redundancy. However, as also in other standards, these are merely meant as detection mechanisms and not as means of fault tolerance.

Fail-operational in software development ([ISO11][part 6])

Since fault tolerance of the SW is typically an architectural design issue, it is no surprise that there are typical measures in the [ISO11][part 6: clause 7]. Tables 4 and 5 contain detection and reaction mechanisms, that are typically meant to improve the fault tolerance of SW.

[ISO11][part 6: table 4] requires certain detection mechanisms on the SW architectural level based on the ASIL. For ASIL D (and only for that), diverse SW design is required.

[ISO11][part 6: table 5] subsequently requires certain reaction mechanisms on the SW architectural level based on the ASIL. The interesting point is that three out of four required mechanisms are typical fault tolerance measures: static recovery mechanisms, graceful degradation and independent

parallel redundancy. There are several curiosities of this requirement. The first one is that these are required or recommended depending solely on the ASIL. The second one is that these measures are completely pointless for fail-safe systems, since a complete shut down is sufficient. In the practice this is exactly what's done, instead of complicated tolerance mechanisms. But for fail-operational SW these mechanisms (or at least one of them) are a must, independently of the ASIL

[ISO11][part 6: Annex D] is concerned with the freedom from interference between SW components. If independent redundant paths are implemented to enhance the fault tolerance of the SW, this becomes an issue. The standard presents three main areas (timing and execution, memory and exchange of information) and various sub-points to support this analysis.

Fail-operational in the safety analyses [ISO11][part 9]

[ISO11][part 9, chapter 7] defines generic requirements on the analysis of dependent failures. As already mentioned, this analysis is essential in case of fail-operational systems. Since these are always based on several sorts of redundancy, the independence of the redundant functions is a key characteristic. Only a thorough, systematic analysis can prove that there are no uncontrolled coupling factors that could invalidate this claimed independence.

The standard defines some requirements and gives brief guidance by requiring the following topics to be addressed:

- random HW failures
- development faults
- manufacturing faults
- installation faults
- repair faults
- environmental factors
- failures of common external resources
- stress due to specific situations

The problem with this part of the standard is that it gives neither guidance which parts of the design need to be analysed using this criteria, nor are these criteria complete and applicable in general.

3.5.2 Fail-operational design in state-of-the-art automotive systems

Since fail-operational architectures are not common in state-of-the-art automotive products, the related literature is mainly concerned with future products where fail-operational behaviour is, or can be, required. Currently, fault tolerance is only considered for systems, where the necessity of this feature is obvious: X-by-wire systems and autonomous or automated vehicles. But fortunately, there are numerous papers and books covering this topic. To structure the results of the literature research, this section will address the following issues:

- Necessity of fault tolerance
- Attributes of fail-operational architectures
- Architectures
- Power supply architectures
- Communication bus systems

Necessity of fault tolerance

As mentioned before, fail-operational behaviour is almost virgin soil for the automotive industry. Therefore even the necessary criteria, for fail-operational architectures are debated and unclear. [Iseo8] and [Stöo2] simply state that fault tolerance is inevitable in case of a high required safety integrity. Contrary to that [Vil14] states that fail-operational behaviour is required if a simple shut down of the system is not safe. This vague formulation is then further detailed: a passive state is unacceptable as safe state if there is any operational phase where a simple deactivation of the system can lead to a safety goal violation. In that case, [Vil14] requires that at least a degraded function shall be provided. A

similar aspect is also discussed by [Ber15], presenting how the rating of the loss of steering support hazard has changed over the past decade. Although the deactivation of EPS systems has been regarded as their safe state, currently some car manufacturers rate this hazard even ASIL C. [Joro0] specifies the lack of a mechanical backup as main motivation for a fail-operational electromechanical brake system. [Cos01], [Ede15] and [Ber15] derive the necessity of fail-operational behaviour from the trend towards autonomous and automated driving. Since in that case, the driver is not in the control loop, the deactivation of the brake-by-wire or steer-by-wire systems (i.e. fail-safe behaviour) would not be safe. [Agr11] identifies regenerative energy storage, park-by-wire and brake-by-wire systems as fail-operational systems, based on the committee draft version of the [ISO11]. The reasons remain unclear. (Remark: The same paper rates propulsion systems as fail-safe).

Attributes of fail-operational architectures

Probably the most important characteristic of fault tolerance is its *degree*. Unfortunately not many sources state any precise figures. Contrary to the vehicles of other domains discussed in the chapters before, passenger vehicles are produced in very high numbers. Due to this, the cost increase caused by redundant elements in the architecture shall be kept to a minimum. [Boro8] even states that redundancy itself has to be kept to a minimum to avoid high costs. [Vil14] is more pragmatic and willing to compromise: for cost efficiency reasons they propose single fault tolerance. The degree of required fault tolerance has definitely an impact on cost, weight and required packaging space. But currently there is no consensus which degree is required based on which criteria. [Sin11] for example states that a brake-by-wire system has to tolerate 2 faults without loss of braking, because the occurrence of a 3^{rd} fault is very unlikely. [Cos01] also requires that one *major critical fault* shall be tolerated, without naming the exact rationale. [Sin11] states that it is sufficient to tolerate 1 critical fault, because the vehicle can be brought into its safe state before a second independent fault would occur. [Joro0] also requires that one fault can be tolerated without loosing the basic brake functionality, but requires only a fail-safe behaviour for the ABS. [Wys03] states that redundancy of X-by-wire systems shall be able to cope with single- and dual point faults, without explaining the rationale.

Another crucial aspect is the *allowed functional and performance degradation* of fail-operational systems, after the first failure. It has to be decided, which functions remain active and which performance they have to provide in order to stay safe. [Sin11] defines three operation modes for the brake-by-wire system: full functional, partial and emergency. Note, that this degradation concept is based on the assumption that 2 faults shall be tolerated. The emergency mode in this paper means that after the 2^{nd} fault the brake-by-wire system applies a constant braking force leading to a pre-defined deceleration to stop the vehicle completely. [Cos01] requires that the residual performance is sufficient if the safety of the system is still acceptable. [Agr11] proposes an electric drive architecture with low performance backup motors to provide a limp-home functionality.

This leads to the next point: which *safety integrity* is required for the *degraded system*, assuming different driver warning scenarios and fault tolerance capabilities? [Ede15] states that the ASIL of autonomous driving vehicle's systems shall not be increased, and shows also an example where a TMR system sustains its ASIL even in the presence of a fault. But there is no clear statement in this study, whether the original ASIL shall be maintained, and if yes, then why. [Ber15] avoids a clear statement, but suggests that probably the same ASIL applies for the degraded mode. The rationale given in that paper is that the parameter exposure in the hazard analysis and risk assessment reflects the probability of an *operational situation* and has nothing to do with the occurrence of a failure.

After clarifying functional and performance requirements of the limp-home mode, there is still one question open: *how long* shall this degraded operation mode last. [Sin11] assumes and requires that safe vehicle operation shall be possible for a certain time period, in order that the driver can seek assistance or continue the journey. [Ber15] also poses the question: how long shall the degraded mode be provided? Related to this aspect, [Ber15] also doubts the applicability of the FTTI as concept.

Architectures

Since fail-operational systems are not established in the automotive industry, architectures proposed by books and papers are either in a research state or are simply carried over from generic literature. [Rei14]

shows for example a basic triple redundant architecture with individual monitoring units for each path. This architecture is only meant as an example, no specific serial product is referred to. [Wys03] and [Agr11] use duo-duplex architectures for their controllers. [Vil14] presents a standard TMR, a duo-duplex and a very interesting hybrid architecture (see figure 3.15) architecture as example. The simplicity of the latter is fascinating. Its second backup path provides only a limp-home functionality, and it is activated if the monitoring of the first path detects a failure. The advantage of this asymmetrical solution is that it is based on a simple monitored primary path (i.e. a standard fail-safe architecture) and a stripped-down backup system.

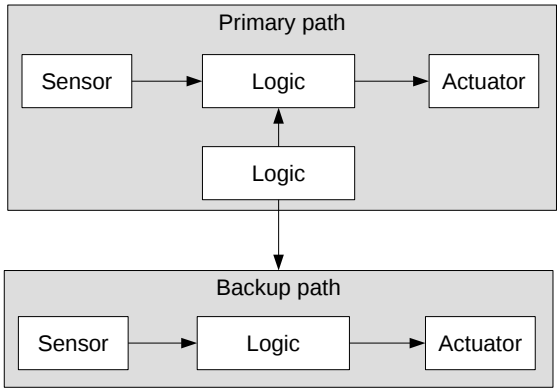


Figure 3.15: Hybrid fail-operational architecture [Vil14]

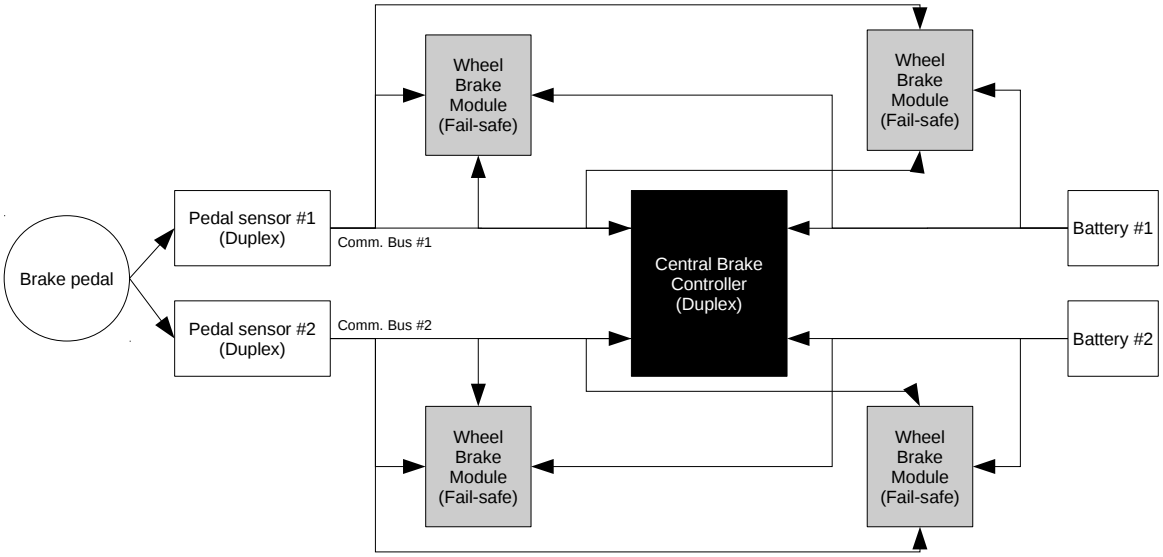


Figure 3.16: Architecture of a brake-by-wire system [Stö02]

[Stö02] gives an insight into the architectures and technical solutions of advanced drive-by-wire systems. This paper shows an electromechanical brake-by-wire system as example (see figure 3.16), with the following architectural features:

- Inherent redundancy by the four wheel brake modules. The same approach is used for conventional hydraulic brakes where diagonal pairs are connected to one hydraulic circuit.

- Duo-duplex brake pedal module. This module has higher fault tolerance than the rest of the system, due to its central role in the architecture.
- Duplex central brake controller
- Duplex power supply, where each battery supplies two of the four wheel brake modules.
- Duplex communication bus, where two wheel brake modules are connected to each other.

[Sin11] and [Joroo] show very similar brake-by-wire architectures. There are only some minor differences:

- all four wheel brake modules are connected to both communication bus lines
- in [Sin11] all four brake modules are connected to both batteries
- all four wheel brake modules have internal voters to compare incoming signals. [Joroo] presents a very sophisticated voter system to compensate faults of one unit by a dynamic brake force distribution.
- the pedal module is TMR in both.

In these examples, the architectural variants are infinite. The challenge is to pick the right combination of fault tolerance measures, and combine them into a financially and technically reasonable product.

An interesting and often overseen aspect of SW redundancy is handled in [Armar]: *reconfiguration* after the failure of a SW component. The authors of this paper developed a formal method to describe and calculate optimal SW configurations. The deployment is based on the basic principle that each SW component is hosted by more than one controller. Each SW component has a master instance and one or more slave instance(s) as hot or cold standby, depending on the required reconfiguration time. Basically, if a controller fails, others have to step in, taking over the execution of the affected SW components. Obviously, the degree of fault tolerance is simply dependent on the number of instances. The one topology, investigated by [Wys03] is also based on SW redundancy and reconfiguration. The SW functions of steering, propulsion and braking are executed by two controllers. These two separate controllers are part of a duo-duplex architecture. Unfortunately, the reconfiguration is not discussed further in that paper.

Many sources refer (e.g. [Rei14], [Ede15]) to the HFT concept of the [IEC10] as a topic related to fail-operational behaviour. In section 3.1.2 it was already pointed out that this is only partially true. $HFT > 0$ basically allows the designer to use components with lower robustness against single point faults in the parallel paths, but this is not meant as fault tolerance measure.

For sensors, *analytical redundancy* is a very attractive, since cost efficient solution. This term denotes topologies where a second instance for a sensor signal is calculated using a model and already measured other physical values. A good example is the redundant steering wheel angle signal of ESP systems calculated using vehicle dynamics data, like yaw-rate and wheel speed signals [Iseo8]. Another application of analytical redundancy is explained in detail by [Iseo8], where the throttle valve angle of an internal combustion engine is estimated by using the measured current and voltage of its actuating electric motor.

Redundant actuator architecture variants can be realised in various ways. [Iseo8] shows parallel and serial layouts for electric motors, with hot or cold standby. More cost-efficient is the smart utilisation of the multiple windings of a permanent magnet synchronous motors, like shown in [Neh10]. Another even more simple solution includes the isolation of a phase of a three-phase electric motor in case of its failure, and controlling the motor by the remaining two phases. In each of these cases, the performance of the system after the first fault is degraded but still acceptable for the particular application. This is a typical limp-home or emergency operation strategy.

[Neh10] shows redundant powerstage architectures for X-by-drive systems, although the presented solutions can be used for any inverter or ECU driving an electric motor via a 2- or 3-phase H-bridge. There are various interesting aspects of these topologies. One key aspect that is showcased by the figures 3.17 and 3.18: the *sphere of replication*. In the former one, not the entire H-bridge is duplicated. Only one leg is added that can step in if one of the legs U, V or W is affected by a fault. Fuses and switches in each leg ensure that the faulty one is separated. In the latter topology, almost all components are duplicated: the DC-link capacitors, the entire inverter and even the motor (using dual winding). Obviously, the broader the sphere of replication, the more faults can be tolerated. For instance, a winding short circuit of a motor winding cannot be isolated and controlled by the additional leg solution. But the question

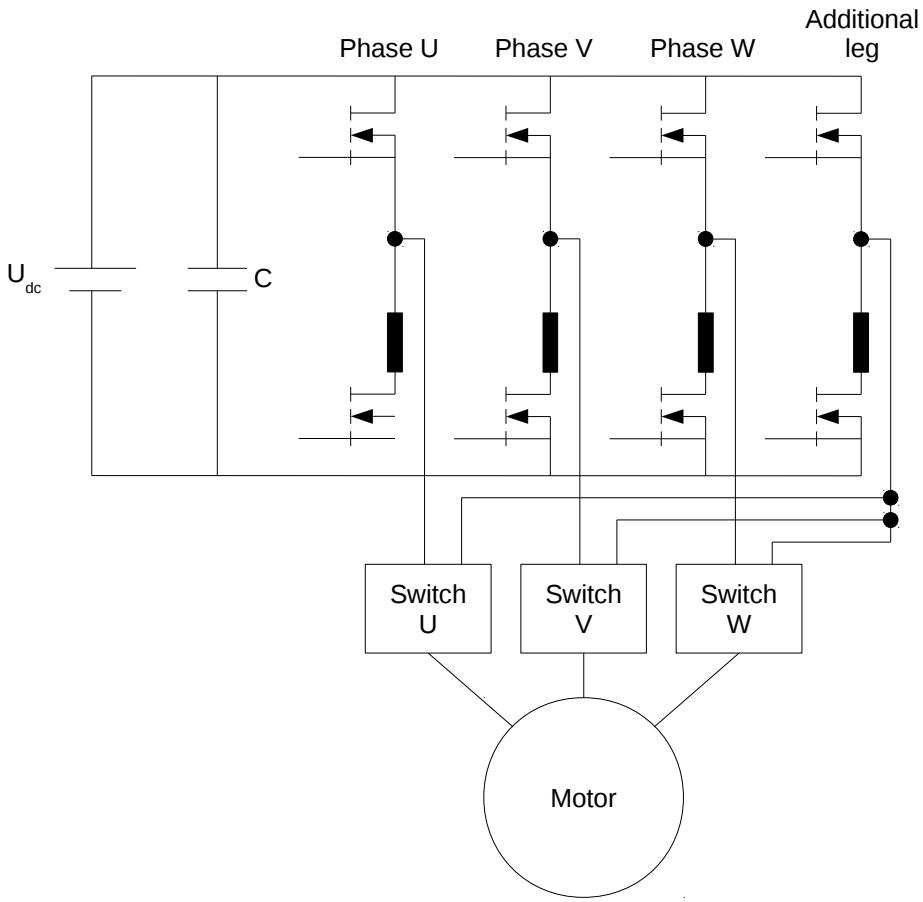


Figure 3.17: Fault tolerant H-bridge topology with an additional leg [Neh10]

to be answered is: do we need to replicate *all* components, or only those with the *highest probability of failure occurrence*?

The two solutions shown in the figures 3.17 and 3.18 also show that fault tolerance can be achieved on a high or low architectural level. Avionics for example tends to apply redundancy on a high level (see 3.2 for further details). Due to cost and packaging space efficiency reasons, low level redundancy is more attractive for automotive solutions. [Neh10] also introduces a very simple and straightforward formula to rate different topologies. The two factors this evaluation uses are cost increase and performance degradation in case of a failure. Curiously, [Wan12] introduces very similar and even identical architectures for hybrid electric vehicle inverters. In both papers (i.e. [Neh10] and [Wan12]), the isolation of the H-bridge leg affected by a fault is done by fuses. Hence, a reconfiguration in case of a transient fault is not possible. By using MOSFET-s (metal–oxide–semiconductor field-effect transistor) instead of fuses, this disadvantage can be eliminated.

[Agr11] shows two architecture alternatives for electric vehicles. The interesting thing in the propulsion system architectures is how redundant architectures are handled. In the first architecture only one electric motor is used with double winding, driven by the same duo-duplex inverter. Therefore, both the inverter and the motor are prone to single point faults. To eliminate this design weakness the second architecture consists of two additional, low performance motors on the rear wheels, with two separate inverters, to provide a limp-home functionality.

[Wys03] introduces an interesting concept: *functional redundancy*. The example in this paper used to illustrate this approach is a steering function. If the steering system fails, it can be replaced by ”steering by braking” using the ESP actuators. This solution is also mentioned by [Ber15], as a concept already

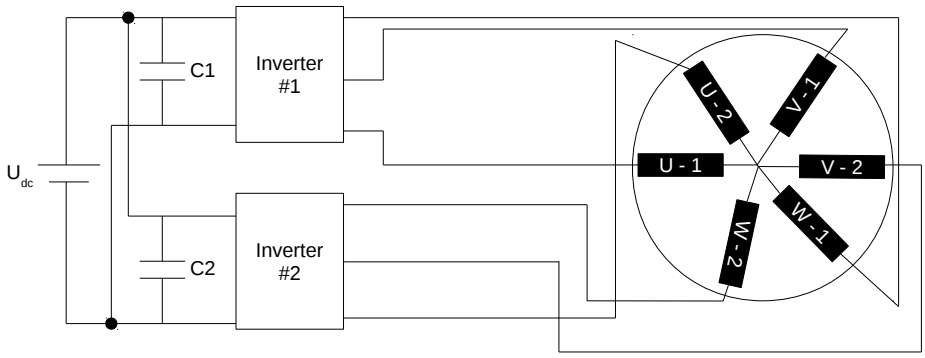


Figure 3.18: Fault tolerant inverter topology [Neh10]

being investigated in the automotive industry.

[Ede15] investigates typical architectures from the [IEC10]: 1001, 1001d, 1002 and 2003 topologies. The most interesting result of his study is a TMR architecture incorporating a symmetrical ASIL-decomposition scheme (see figure 3.19). In order to support this architecture, [Ede15] even proposes the extension of the [ISO11] by this ASIL-decomposition scheme. The fascinatingly elegant feature of this solution is that the commonly used TMR architecture is extended by ASIL-decomposition, yielding a very lean architecture that maintains its safety integrity even after the first fault occurs. There is one question that remains unanswered, though. It is unclear, why the diagnosis blocks are used for a TMR architecture, since its main advantage is not needing any of those.

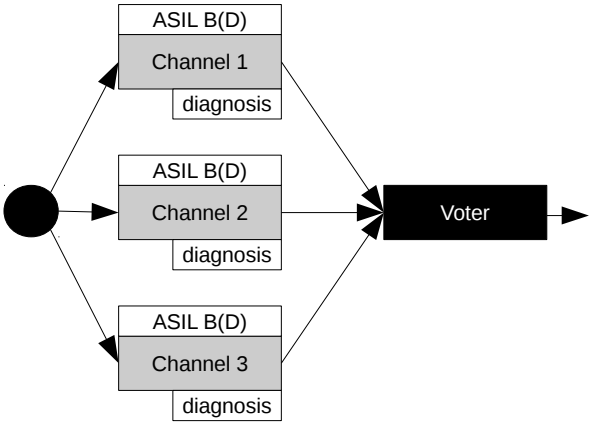


Figure 3.19: ASIL D TMR architecture [Ede15]

[Ber15] addresses briefly the issue of the HW metrics of the [ISO11], pointing to the phenomena that failure modes that were undoubtedly rated as safe will turn into potential SPF-s or MPF-s.

Power supply architectures

Several sources (e.g. [Sin11], [Stö02], [Wys03]) address the issue of the single channel power supply of passenger vehicles. Most sources (e.g. [Stö02], [Wys03]) only mention that a redundant power supply is necessary, but they fall short of improvement proposals. [Bor08] proposes to use the high voltage battery of hybrid electric vehicles as second power source. For a possible solution see figure 3.20.

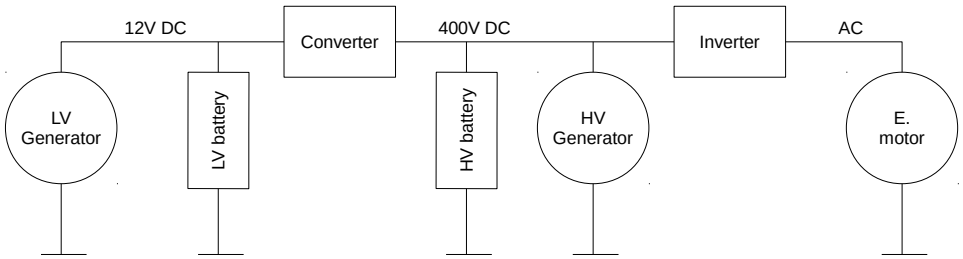


Figure 3.20: Redundant power supply architecture for hybrid electric vehicles

[Agr11] identifies the loss of power supply as safety critical hazard and also proposes a duplex power supply with two energy storage systems. Unfortunately, the presented concept does not go into details. So it remains unclear how the electrical power is distributed and how different consumers are isolated from the grid in case of a failure.

Communication bus systems

For fault tolerant systems that rely on incoming and/or outgoing communication signals, fail-operational communication bus systems are a must, as stated by e.g. [Stö02], [Sin11] and [Wys03]. [Bor08], [Sin11] and [Ech07] mention that FlexRay basically supports redundant bus structures, and that two channels can be even handled by one FlexRay controller. Another interesting, and for fail-operational purposes helpful feature of FlexRay are the bus guardians, that can shut down and isolate faulty communication transceivers [Bor08]. Besides that FlexRay is partially time-triggered which ensures that safety related signals cannot be lost due to bus arbitration.

Of course, all these properties are worthless without an appropriate architecture. [Ech07] states that FlexRay can support various architectures: TMR, duo-duplex, etc. The architecture shown in 3.16 is for example a duo-duplex one. [Ech07] defines also the design aspects that need to be addressed when developing a FlexRay-based fault tolerant communication bus:

- Degree of fault tolerance,
- Time- or event-triggered processing,
- Fault tolerance protocols,
- Worst case execution time,
- Error handling,
- Topology.

A fault tolerant communication protocol can be designed based on already established automotive bus protocols. CAN or FlexRay are both appropriate if integrated into the right architecture.

[Nen07], on the other hand, proposes a completely different communication topology. Instead of a bus topology, where all communication partners are connected to the same physical medium, the proposed SafeNET architecture is based on a network of individual nodes. By using an appropriate topology, the failure of one node does not affect the communication of the others.

[Wys03] even requires diverse physical media for the two communication channels. This could be realised by optical and conductive media.

Driver warning

Not many sources address the topic of driver warning, although it is one of the most challenging questions for fail-operational automotive systems. As already presented in section 3.2.2, in air- and spacecraft the pilot is integral part of the fault handling concept, making difficult decisions about de- and reactivating failed units. This is definitely not an option for passenger vehicles, but the question still remains open: how much can be expected from the driver?

[Sin11] and [Cos01] assume that already after the first fault, the driver has to be warned. It is then his or her decision to seek assistance in a service or to continue the driving cycle. A similar concept is presented by [Ber15]: in case of autonomous driving, the driver is warned after the first failure and is obliged to regain control in approximately 10 seconds.

Analysis of dependent failures in automotive systems

[Cha09] provides guidance on the main terms related to dependent faults in automotive systems. The paper defines the terms common cause failure, common mode failure and cascading failure, in a similar way as the [ISO11] does. There are some inconsistencies, though. These inconsistencies are important, because there are common misunderstandings and misconceptions within the automotive functional safety community. Some argue that the main difference between cascading failures and common cause failures is that the latter has always an *external* root cause. This paper highlights the fact that there are cases (e.g. design faults) where this is not true, but it still sticks to this distinction. Besides that, this paper states another misconception that random hardware faults do not cause common cause failures.

[Cha09] presents two types of dependent failure analyses: one for common cause failures and another for cascading failures. The first analysis, focusing on common cause faults, is based on a checklist, identifying potential root causes, e.g.:

- Mechanical aggression resulting from misuse or crash
- Extreme environmental conditions
- New technology
- Wrong maintenance
- Production failure
- Design fault
- Specification fault
- Physical routing of signals

The second analysis is intended to detect cascading failures by re-analysing the effects of failures, looking for potential coupling mechanisms. Unfortunately, the two methods are not detailed further. Hence it remains unclear, how these root causes have been derived, or if this model is complete and consistent. Besides that, it is unsettled how to identify the points to look at with the checklist, and how this checklist can be derived.

3.6 Summary of the literature research

The literature study has shown that the generic aspects of fail-operational architectures are well known and thoroughly documented. In particular in avionics, several solutions for fault tolerant systems, HW and SW have been developed and proven over the last four decades. Air- and spacecraft architectures, including power supply and communication networks, evolved over this time period to highly sophisticated topologies accepted as benchmark in various industrial domains. These various architectures have been analysed regarding their availability and safety in terms of their overall availability and MTTF. All these topologies are based on some kind of redundancy, applied for physical components, information or SW supporting the tolerance of faults in the respective elements. The study of other domain safety standards and guidelines has shown that there is only very little that can be carried over into automotive products or into the [ISO11]. These safety standards either do not address the topic specifically or require techniques and mechanisms that can be transferred into road vehicles only with great care taken. Nonetheless, topologies and technical solutions applied in safety related high availability avionics systems can be used as starting point for the development of automated and autonomous driving vehicles.

For automotive systems, though, the existing studies and related literature can focus on future challenges and research projects only. Most automotive products show fail-safe behaviour; fault-tolerance techniques are applied only on the low HW or SW level. Currently there is no clear consensus why and when fail-operational behaviour is needed at all in the first place. The existing literature focusing on fail-operational automotive systems is either addressing particular technical solutions (e.g.

[Neno7], [Neh10], [Sin11], [Agr11], [Wys03] or [Echo7]), or is rather focusing on the challenges that lie ahead, without going too much into the proposed solutions (e.g. [Ber15], [Ede15] or [Vil14]).

Although various papers (automotive and non-automotive alike) state that independence is the key for redundant architectures, the literature on dependent failure analyses is rather scarce. Some structured methods are available to identify common cause failures in avionics, but those are only remotely applicable for automotive systems. The reasons for this are manifold. First of all, the [ISO11] addresses another type of dependent fault: cascading faults. Besides that, the models and methods applied for avionics systems are based on FTA-s and checklists tailored for the analysis of air- and spacecraft. For automotive systems, there is no structured method defined that is based on a generic model and can be integrated into a typical automotive analysis landscape.

The [ISO11] itself contains, as the present study has shown, some hints related to fault tolerance but is predominantly meant to be applied for fail-safe architectures. This is completely understandable given the fact that state-of-the-art automotive systems are fail-safe. Some papers (e.g. [Ede15]) even doubt that the [ISO11] can be applied for fail-operational systems at all, without going further into details.

Based on the results of the literature research, this thesis wants to fill the following gaps:

- Derive a structured method and clear criteria to identify the necessity of fail-operational behaviour due to safety reasons.
- Identify generic attributes of fail-operational behaviour to support the high level requirement definition. These attributes define basic behaviour patterns and high level requirements on fail-operational systems, and include criteria on the necessity of fail-operational behaviour; types and number of failures that need to be tolerated; required system performance after the first fault.
- Investigate the applicability of the [ISO11] in detail for fail-operational sytems, HW and SW starting from the concept phase through the entire safety lifecycle. Derive requirement extensions and improvement proposals wherever necessary.
- Investigate typical fault tolerant HW architectures from the viewpoint of the [ISO11][part 5, clauses 8 and 9] (i.e. with regards to their SPFM, LFM and SPFM) instead of their MTTF.
- Investigate typical fault tolerant SW topologies regarding their applicability in an automotive environment. Evaluate their suitability based on criteria relevant for state-of-the-art automotive systems.
- Derive a generic dependent failure analysis technique that can support the analysis of claimed diversity on the system, HW and SW levels, to extend and complete the requirements of the [ISO11][part 9].
- Develop a method to optimise redundant automotive architectures.

4 Aspects of fail-operational systems conforming to the [ISO11]

4.1 Structure of this chapter

How the aspects of fail-operational systems are addressed in the [ISO11] have been already examined in section 3.5.1. As next logical step in this chapter, the applicability of the [ISO11] is investigated in detail, identifying potential gaps and deriving improvement proposals. The following key topics of the development of fail-operational systems will be investigated from the viewpoint of the standard:

- necessity of fail-operational behaviour,
- basic attributes of fail-operational behaviour,
- development of fail-operational systems, HW and SW,
- technical independence and analysis of dependent failures.

These items and the didactic structure of this chapter follow the safety lifecycle of the [ISO11], starting with the concept level considerations and then moving on to the three areas of the product development.

4.2 Method to determine the necessity of fail-operational behaviour [Sch15]

As already shown in chapter 3, there is currently no clear criteria defined, to identify the necessity of fail-operational behaviour. Some sources refer to the high required safety integrity, some to autonomous and X-by-wire systems. In this section, this criteria is derived from basic definitions and axioms of functional safety. Then, a suitable method is developed based on the previously derived criteria.

The basic principle is that fault tolerance is required if the passive state (i.e. where the element ceases to perform at least one of its intended functions) or the transition into it is unsafe. Based on the definition of safety, this would mean that the associated risk is unreasonable. In order to get to clear criteria, the hazard analysis and risk assessment method (see section 2.4.2) can be applied. By analysing the loss of particular functions (or the entire set of functions) using the three parameters (i.e. severity, exposure and controllability), the risk of a passive state and of the transition into it can be assessed. The proposed method can be illustrated using the following examples:

- *EPS system*, providing steering support for the driver. Failure mode under investigation: Loss of steering support
  - Operational situation: Cornering on a country road → Exposure: E4 (high probability)
  - Effect on the vehicle: Steering effort significantly and suddenly increased → Controllability: C1 (simply controllable)
  - Worst case accident: collision with other traffic participants or road-side objects → Severity: S3 (severe injuries, survival uncertain)
  - Resulting ASIL = B
- *Brake-by-wire system* without mechanical backup. Failure mode under investigation: Loss of primary braking function
  - Operational situation: Braking before cornering on a country road → Exposure: E4 (high probability)
  - Effect on the vehicle: Vehicle speed cannot be decreased with the right deceleration → Controllability: C3 (difficult to control or uncontrollable)

  - Worst case accident: collision with other traffic participants or road-side objects → Severity: S3 (severe injuries, survival uncertain)
  - Resulting ASIL = D

- *Double-speed transfer case*, providing a low range gear for better off-road performance. Failure mode under investigation: Low range cannot be engaged
  - Operational situation: Off-road, steep down-hill descent → Exposure: E1 (very low probability)
  - Effect on the vehicle: Engine brake insufficient, vehicle accelerates → Controllability: $C0$ (controllable in general)
  - Worst case accident: Low to medium speed collision with object → Severity: S2 (severe injuries, survival probable)
  - Resulting ASIL: no rating necessary, since controllability is $C0$

As shown in the example, the risk related to the loss of function hazards could be assesed using the HARA. The resulting ASIL gives guidance whether fault tolerance capabilities are required or not. If the $ASIL \geq A$, the situation is clear: fail-operational behaviour is required and all mechanisms related to it have to be developed in accordance with the [ISO11]. But to cope with the QM cases is less obvious. For this, the definition of QM has to be understood. As the [ISO11][part 3, 7.4.4.1, NOTE 2] says that QM "denotes no requirement to comply with the ISO 26262". Therefore, QM per se doesn't mean that the failure mode in focus is not a hazard. QM means only, that the related risk is so low that even a state-of-the-art quality management can reduce it to an acceptable level. So, a QM rating is no excuse from implementing fault tolerance measures. The only case, where no fail-operational behaviour is required, is where one of the parameters (i.e. S, C or E) is rated 0 (i.e. $S0$, $C0$ or $E0$, respectively). In that case the failure mode is not a hazard, since it doesn't lead to harm, is always controllable or can lead to harm only in incredible operational situations.

Another intriguing aspect is that all three example systems have other safety goals as well, with some times different ASIL-s than that of their loss of function hazards:

- EPS → Safety Goal: Prevent self-steering (ASIL D)
- Brake-by-wire-system → Safety Goal: Prevent unintended braking (ASIL C)
- Double-speed transfer case → Safety Goal: Prevent unintended range change to neutral whilst parking (ASIL B)

This shows that the safety goals related directly to the fail-operational behaviour may have completely different ratings than other safety goals of the product. This aspect will be further investigated in section 4.4.

As shown in these examples, the HARA method can be utilised to identify the necessity of fail-operational behaviour. By deriving appropriate safety goals, fault tolerance shall be required explicitly, to support later phases of the safety lifecycle (for further details see section 4.4).

4.3 Definition of the attributes of fail-operational behaviour [Sch15]

As starting point of any fail-operational design, certain attributes of fail-operational behaviour need to be specified up front. These attributes have been derived based on the literature research (section 3.5.2), but were not described before as basic and structured concept:

- Fault tolerance targets
- Allowed functional and performance degradation
- Time to remain operational after the first fault occurred
- Required safety integrity for the system after the first fault

As it can be seen from the list above, these attributes are important inputs for the system, HW and SW development (see sections 4.4.2, 4.4.3 and 4.4.4).

4.3.1 Fault tolerance targets

Fault tolerance is not an absolute term. The factors, that need to be defined regarding this matter are:

- qualitative targets (i.e. the degree of fault tolerance and the type of faults that need to be tolerated),
- quantitative targets (i.e. reliability-related factors).

Both the qualitative and quantitative targets are related to the basic concept of the *sphere of replication*. Depending on these targets, it can be determined, which portion of the product needs to be replicated. Since redundant elements mean always higher cost, weight, complexity and bigger packaging space, it is the key interest of the designer to minimize the sphere of replication. Obviously, the smaller the replicated portion, the higher the amount of single point faults. Previously, in section 3.5.2 was already mentioned that various sources argue that only the components with the highest failure rates should be redundant. This already implies that some sort of quantitative criteria shall be applied.

First of all, it has to be emphasized again that the risk related to fail-operational systems is not higher per se than those of fail-safe ones. Therefore basic principles of the [ISO11] can be carried over to fail-operational architectures. Applying more stringent requirements is not necessary.

Quantitative fault tolerance targets

For discussing this attribute, the example from the section 4.2 can be re-used: EPS. For this aspect, the following two safety goals can be investigated:

- Safety goal #1 (SG1): Prevent self-steering, ASIL D
- Safety goal #2 (SG2): Maintain steering support, ASIL B

It is obvious, that the SG2 represents the fail-operational requirement. Quantitative requirements can be easily defined by applying the [ISO11][part 5, 8 and 9.4.2] (i.e. the concept of the SPFM, LFM and PMHF) for this safety goal. In order to understand, why these metrics fit perfectly for this purpose, the meaning of these metrics in this context shall be investigated:

- The *SPFM* denotes the robustness against single point- and residual faults, leading to the violation of SG2, i.e. to the loss of the function. The higher its value, the lower the amount of single point faults. By widening the sphere of replication, the SPFM can be improved, due to obvious reasons.
- the *LFM* reflects the robustness against latent faults, leading to the violation of SG2, i.e. to the loss of the function. The higher its value, the lower the amount of latent multiple point faults. The LFM can be increased by improving the diagnostics related to the redundant elements and by informing the driver if a fault occurs and needs to be tolerated.
- the *PMHF* indicates the probability of the violation of SG2, i.e. the residual failure rate of the loss of function hazard. This value can be decreased by expanding the sphere of replication, by improving the failure mode coverage by diagnostic mechanisms and by improving the component failure rates.

The three metrics of the [ISO11], applied for the fail-operational safety goal (SG2 in the example) perfectly support the definition of quantitative fault tolerance targets.

By applying quantitative targets for fault tolerance, different architectures can be compared in a very comprehensible and structured way. And this leads to the problem faced in case of fail-safe systems as well: what to do with ASIL A systems, where no quantitative targets are available? Even if it appears to be a problem common to fail-safe and fail-operational systems, the former case is far more straightforward. Fail-safe safety related E/E systems have been being developed for decades, yielding several safety architectures and generally applicable safety concepts. Due to this reason, there is commonly agreed state-of-the-art for fail-safe systems of all ASIL-s. But when it comes to fail-operational automotive systems (as investigated in 3.5.2), there is no state-of-the-art out there. Serial production fail-operational topologies are rare, and pre-development architectures are kept under wraps. Therefore, as discussed in the next section, it is very difficult to use qualitative targets for ASIL A applications, where quantitative targets cannot be derived directly from the [ISO11].

Qualitative fault tolerance targets

Quantitative targets offer many advantages, but have one major weakness: they can be applied only for random HW faults. If systematic faults should be addressed as well, qualitative targets are needed. In order to derive qualitative fault tolerance targets from the [ISO11], the following questions have to be answered:

- Are SPF-s allowed at all? If yes, then for which cases?
- MPF-s of which order are allowed?
- How to cope with systematic faults?

The [ISO11][parts 4 and 6] (i.e. system and SW, respectively) do not contain any requirements defining if SPF-s are acceptable. Part 5, concerned with the HW, on the other hand, is unambiguous on this matter. The related requirements are in clause 9 (see 2.4.1), therefore the evaluation of the PMHF (in accordance with the [ISO11][part 5, clause 9.4.2] and the "Evaluation of each cause of safety goal violation" method [ISO11][part 5, clause 9.4.3] have to be distinguished:

- In case of applying the PMHF, there is no related requirement for ASIL A and B. For ASIL C and D SPF-s are only acceptable if the related parts are treated with dedicated measures (e.g. burn-in test) (see [ISO11][part 5, 9.4.2.4]).
- In case of applying the "Evaluation of each cause of safety goal violation" method, there is no requirement for ASIL A. For ASIL B, SPF-s are acceptable if the related failure rate is sufficiently low. For ASIL C and D, the failure rates shall be sufficiently low and dedicated measures shall be applied.

Based on this, it can be stated that in case of both methods, for lower ASIL-s, SPF-s are acceptable. The same applies for higher ASIL-s, if the related parts are treated with dedicated measures. This means that based on the [ISO11], no required degree of fault tolerance can be derived. Since SPF-s are acceptable even in case of ASIL C and D (under defined circumstances), no complete fault tolerance related to random HW faults is required.

This leads to the next question raised at the start of this section: To which extent are MPF-s acceptable? The answer is, that there are no related requirements in the standard.

After having clarified which qualitative targets apply for random HW faults, it needs to be investigated how systematic faults need to be dealt with. As already explained in section 2.2, the nature of random and systematic faults is completely different. While random HW faults cannot be prevented and avoided, systematic ones can, at least to some extent. Hence, when it comes to fail-operational behaviour, tolerance of random HW faults is a must. On the other hand, whether systematic faults need to be coped with by applying fault tolerance, can be debated. Classical fail-operational topologies, applied on the system level, can tolerate them to a certain extent. The key here is the level of independence. As already shown in section 3.2.2, even the Boeing 777 used the same SW with different compilers [Yeh96].

Going back to the [ISO11], it can be seen that safety mechanisms controlling random HW faults are also capable of doing so with systematic failures of systems and HW. For SW, the [ISO11][part 6] (especially [tables 4 and 5]) are very process-oriented for ASIL A and B, adding almost no runtime detection mechanisms and no recovery mechanisms at all. For ASIL C and D some basic fault tolerance mechanisms are required, but not consequently.

Therefore it can be stated that in the [ISO11], there is no specific guidance related to the tolerance of systematic faults.

4.3.2 Allowed functional and performance degradation

As specified in the definition in section 2.2, fail-operational behaviour does not mean that the system shall remain fully functional after a fault occurs. Depending on the component affected by the fault, the architecture and the degradation strategy, the system performance may be degraded.

Performance of safety related fail-operational systems can be described in many ways, depending on the characteristics of the system in focus. Depending on the function of the product, various characteristic physical parameters can be regarded as performance. This performance indicator may be a force, torque, electric current, deceleration, etc. Decisive is the safety relevance of these performance

indicators. For example in case of a brake-by-wire system, the achievable maximal deceleration is definitely a performance indicator, whilst frequency of the noise generated by the brake disks isn't. Using the same selection criteria, performance can be also regarded as a set of functions that is provided by the system. From this point of view, higher performance means higher amount of functions still active.

Using this approach, there are mainly two types of performance degradation:

- deactivation of functions that do not require fail-operational behaviour,
- degrading the output performance of the fail-operational functions.

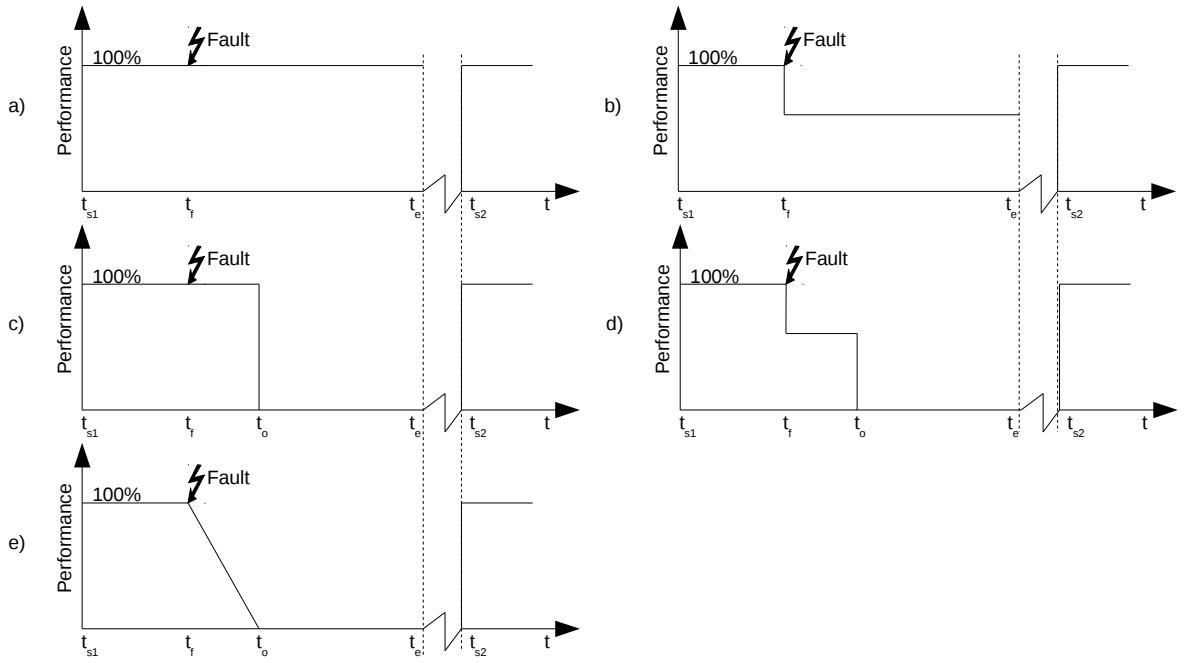


Figure 4.1: Performance degradation - Different types of fail-operational behaviour

Different types of this behaviour are shown in figure 4.1. In each case the time-stamps are the following:

- The first driving cycle starts at t_{s1}
- The fault occurs at t_f
- The first driving cycle ends at t_e
- The next driving cycle starts at t_{s2}
- In case of some architectures, the system enters a passive safe state at t_o

Note, that these diagrams show two parameters of fail-operational behaviour: the allowed performance degradation, and the time to remain operational after the first fault occurs. The list of variants may be infinite, 4.1 shows the most basic behavioural models:

a) This case shows the high-end solution: after the fault, there is no degradation at all, the system provides its full performance until the end of the driving cycle. In the next driving cycle, after passing a start-up test, the system is fully operational again. This model may be appropriate for steer-by-wire-systems, for example.

b) This diagram shows a behaviour, where the fault leads to a certain performance degradation, but the system remains operational (to some extent) until the end of the driving cycle. An electric propulsion system, where a limp-home function is still provided after a fault occurs, could be an example.

c) This time-chart shows a temporary fail-operational behaviour: After the fault, the system provides its full performance but only for a limited time. Obviously, the driver has to be warned at t_f. The lane keeping assist system of an automated driving vehicle could be an example, where the driver can regain control within a certain time.
d) This one is a mixture of b) and c): The performance is degraded after the fault occurred, and the system goes into a passive safe state after a certain time. For this case the same is valid as above: the driver has to be warned at t_f.
e) The last chart shows a typical measure to increase the controllability of the transition into a passive safe state: the system remains operational for a certain time but its performance is degraded continuously to zero. This also requires driver warning at t_f. A EPS system, decreasing its steering support continuously, to minimize its vehicle dynamics impact, can be mentioned as example.

Factors that need to be considered when deciding which behaviour suits best for the system in focus include cost, functional requirements and *functional safety*. This thesis is focusing on the latter. To identify which level of performance degradation is allowed, the method as described in section 4.2 has to be used: apply the HARA for the following two potential hazards:

- Loss of a certain, specific function. This gives guidance, which functions can be simply switched off and which shall remain active.
- Degradation of a specific function to a particular extent. This gives guidance, which degradation is still safe.

The evaluation criteria is the same as identified in section 4.2.

Already in figure 4.1 it gets clear that the chosen fail-operational behaviour cannot be separated from the chosen warning concept. The [ISO11][part 3] calls it even a *warning and degradation concept*, which makes absolutely sense, since the two interact in various ways. Let's take the following example: the steering system of an autonomously driving vehicle. The controllability of the loss of steering hazard would be probably *C*3. But if the driver is warned and the function is maintained for the time required for the driver to regain control, the controllability may be *C*0 (note that this behaviour is depicted in figure 4.1, point c)).

The [ISO11][part 3, clauses 7 and 8] offer appropriate methods to derive an appropriate warning and degradation concept for a fail-operational system. As already said in section 4.2, the loss of function hazard needs to be analysed, leading to a safety goal with a certain ASIL. In the example above, this would be "Steering function shall be maintained", with an ASIL B. Based on this, the functional safety concept can define a functional safety requirement that the "steering function shall be maintained for 20*s* after a fault potentially leading to loss of steering function occurred". Besides that, the warning and degradation concept (also included in the functional safety concept) would describe a driver warning strategy, to notify the driver that he or she has to regain control within 10*s*.

The warning and degradation concept (including the allowed performance degradation) has to be a subject of safety validation, with special focus on the controllability.

4.3.3 Time to remain operational after the first fault occurred

As already shown in figure 4.1, there are various temporal behavioural models, that can be applied for fail-operational systems. Basically, the system shall remain operational at least until the driver or another vehicle system takes over. For the time to remain operational, the same applies as for the allowed performance degradation (see section 4.3.2): it is closely related to, and in interaction with, the warning and degradation concept.

To investigate this fail-operational attribute, the definition of the emergency operation interval ([ISO11][part 1, 1.35]) shall be revisited: *"specified time-span that emergency operation is needed to support the warning and degradation concept"*. Where the term *emergency operation* denotes *"degraded functionality from the state in which a fault occurred until the transition to a safe state is achieved [...]"* ([ISO11][part 1, 1.34]). As it can be seen, both terms are tailored to fail-safe systems. Looking back at figure 4.1, the following can be stated:

- In case of *a)*, the emergency operation interval does not exist. In these cases, the system is reconfigured almost instantly and a fully operational safe state is reached.

- *b)* is a difficult question. Although an operational safe state is reached, due to the degraded performance, this could be regarded as emergency operation. But on the other hand this is not only required "until the transition to a safe state is achieved" as the definition says. Therefore, the definition shall be adapted.
- *c)*, *d)* and *e)* are clear: the emergency operation interval lasts from t_f to t_o.

Extension and optimisation proposal: Emergency operation

An optimised wording for [ISO11][part 1, 1.34] could be the following: degraded functionality from the state in which a fault occurred until the transition to a safe state is achieved or until the system is deactivated at the end of the ignition cycle.

Therefore, this duration shall be determined together with the definition of the warning and degradation concept.

Note, that in case of permanent faults, there is another question related to the time to remain operational: what to do at the start of the next ignition cycle? Considering that fail-operational automotive systems are most likely capable of tolerating only one fault (see section 4.3.1), these faults being tolerated are latent dual point faults. The [ISO11][part 4, 6.4.4.1-6.4.4.3] requires that safety mechanisms to detect latent faults are defined and that an appropriate multiple-point fault detection interval is specified. Since this interval is usually set to one driving cycle, these safety mechanisms are performed typically either at start-up or at shut-down. Here, again, the warning and degradation concept shall specify whether the system remains passive (since this is allowed in this case, right after the start of the ignition cycle) or goes into a degraded operation and the driver is warned.

4.3.4 Required system safety integrity of system after the first fault

As already shown in section 3.5.2, it is not clarified yet which safety integrity is required for the system after the first, tolerated fault occurred. As already presented in section 2.4.2 and shown in figure 2.8, the HARA analyses *failure modes* in various *operational situations* and evaluates their related risks using three parameters: *exposure* to the operational situation, *controllability* of the hazard in that operational situation and the *severity* of the harm if the hazard couldn't be controlled.

In accordance with this, the safety goals and/or their ASIL of the system after the first fault can differ from the original ones, only if one of the following conditions is met:

- There are former failure modes that are impossible with the reconfigured system.
- There are new failure modes that were impossible with the original fault-free system.
- The extent of the hazards has changed, leading to a different controllability and/or severity in the rating.
- The driver is warned so that he or she can avoid certain operational situations, leading to a different exposure rating.

Impossible failure modes

In some architectures the reconfigured system may have other or less failure modes than the original one. This can only happen if certain sub-functions are completely deactivated or if the backup system has different components with different failure modes.

Newly introduced failure modes

In case of some vehicle architectures it may occur that the role and function of a system will be taken over by a complex interaction of systems. In this case the complexity of the topology behind the very same functions may be higher and therefore completely new failure modes are introduced.

An example could be a steering by braking function in case of a steer-by-wire equipped vehicle.

Changed hazard extent

If the system performance is lower after the reconfiguration, the following factors may be profited from:

- the same hazard can be *more easily controlled* (leading to a better controllability),
- the same hazard can lead to a hazardous event *only in certain operational situations* (leading to a lower exposure),
- the same hazard can cause *less harm* (leading to a lower severity).

Take as example a rear axle electric drive with a dual winding motor (see section 3.5.2). If an unintended acceleration torque is produced by the system during cornering, it can lead to oversteer. The extent of this oversteer depends on the grip level (e.g. high grip on dry tarmac, low grip on icy roads) and on the produced torque. The controllability is obviously better if the oversteer is less severe. If due to a fault, one set of windings is deactivated, the residual performance is approximately 50%. Therefore, the vehicle destabilisation on dry tarmac is much less severe, hence possibly simply controllable ($= C1$). On low-grip surface (e.g. snowy roads), the effect may be still uncontrollable oversteer but here the exposure is lower than that of a high-grip surface (usually $E2$ versus $E4$).

Driver warning leading to lower exposure to certain operational situations

The intention of warning and degradation concepts is to influence driver behaviour. This expected behaviour can be a certain action (e.g. applying the parking brake when the parking lock of the automatic transmission is out of order) or just increased alertness (e.g. decreasing the speed and avoiding on-limit manoeuvres if the ESP control lights are on). The assumption is that if the driver knows that a particular system is affected by a fault, certain operational situations will be most probably avoided. Therefore, the hazard does not become a hazardous event or at least only one with a lower probability.

This assumption is valid only if both of the following criteria is fulfilled:

- driver is warned,
- it can be assumed that the unfavourable operational situation is avoided, because either the driver knows explicitly what to avoid, or the warning looks severe enough to yield increased alertness.

The latter point is heavily disputed. Although this approach reflects the current state-of-the-art in the automotive industry, there are opponents. The strategy of the car manufacturers is to design an appropriate warning concept backed up by an exhaustive user manual. Based on that it is assumed that the driver is aware of the seriousness of the warning and he or she will react properly. The main point of criticism is that due to the sometimes overwhelming amount of information (on the display and in the user manual) and due to the foreseeable ignorance of the drivers, this expectation is false.

It is important to emphasise again that the ASIL of the reconfigured system cannot be decreased by arguing that the exposure is lower to "situations" where the system is affected by a fault. The parameter *exposure* (as already mentioned by [Ber15]) depends solely on the *operational situation* and not on the fault probability. That the allowed probability of safety goal violations may be higher for the emergency operation interval may sound reasonable, but needs to be investigated separately. One of the key features of the [ISO11] is that the ASIL is not defined in terms of fault probabilities or failure rates. Therefore, the two issues can and shall be distinguished here.

4.4 Fail-operational aspects in product development

4.4.1 Fail-operational aspects in the concept phase

The two main work products of the concept phase of the [ISO11] are the HARA (including the safety goals) and the functional safety concept. With regards to fail-operational behaviour, the following can be addressed during this phase:

- necessity of fail-operational behaviour (section 4.2),
- allowed functional and performance degradation in case of reconfiguration (section 4.3.2),

- required time to remain operational (section 4.3.3),
- ASIL of the reconfigured system (section 4.3.4).

As visible this list, these are practically the fail-operational attributes identified before, with the exception of the required degree of fault tolerance.

Hazard analysis and risk assessment

As already mentioned in section 4.2 and 4.3, the HARA method is applicable for fail-operational systems. Here, no adaptations are necessary. The necessity of fail-operational behaviour and an appropriate operational safe state can be determined by a fail-operational safety goal. This safety goal is always related to a "loss of function" hazard.

Extension and optimisation proposal: Additional note to the HARA to address fail-operational systems explicitly

Provide guidance related to fail-operational systems with the following note to requirement [ISO11] [part 3, 7.4.2.2.2]: NOTE 3 In order to identify the necessity of fail-operational behaviour, the loss of the item's functions may be considered as hazards.

Functional safety concept

The requirements related to the functional safety concept and the related workflow fit perfectly for fail-operational systems as well. Only some minor extensions are required.

The [ISO11][part 3, 8.4.2.4 - 8.4.2.6], addressing the relation between an operational safe state and the warning and degradation concept, can be applied for fail-operational items. The following extension proposals can give much more accurate guidance for these systems.

First of all: to make the specification of later requirements easier, the following definitions may improve the applicability of the standard to fail-operational systems:

Extension and optimisation proposal: Definitions of different failure behaviour

Fail-safe behaviour: Elements showing *fail-safe* behaviour enter an active or passive *safe state* and cease to perform their functions in case of a certain amount of failures.

Fail-silent behaviour: Elements showing *fail-silent* behaviour enter a *safe state* with no interference with other elements and cease to perform their functions in case of a certain amount of failures.

Fail-operational behaviour: Elements showing *fail-operational* behaviour continue to perform a defined set of their intended functions to a defined extent, with a defined performance, for a defined time in case of a certain amount of failures.

Besides that, in case of fail-operational systems, there are some aspects of the warning and degradation concept that need special attention:

Extension and optimisation proposal: Aspects of fail-operational warning and degradation concepts

Additional note to the requirement [ISO11][part 3, 8.4.2.5]:

NOTE 2 In case of a fail-operational system, the warning and degradation concept may address the following:

- functions that shall remain active,
- required performance of these functions,
- required time for these functions to remain active,
- related driver warning, if applicable,
- expected benefit from the driver warning.

Additionally to these points, the [ISO11] shall define the criteria that can lead to a lower ASIL of the reconfigured system:

Extension and optimisation proposal: Criteria for repeating the HARA for the reconfigured system

8.4.2.8 If one of the following applies for the system after the first, tolerated fault, the HARA shall be repeated and the functional safety concept shall be updated accordingly:

- there are former failure modes that are impossible with the reconfigured system
- there are new failure modes introduced with the reconfigured system,
- the extent of the hazards has changed, leading to a different controllability and/or severity in the rating,
- the driver is warned so that she or he can avoid certain operational situations, leading to a different exposure rating.

4.4.2 Fail-operational aspects in the system development

On the system level, the main work products where aspects of fail-operational behaviour need to be addressed, are the technical safety requirements specification, the technical safety concept and the system design.

The main goal of this part of the standard is to derive technical safety requirements from the functional safety concept and to allocate these to the elements of the refined safety architecture. Therefore, this is the right step to define fault tolerance mechanisms and to refine the fault tolerant architecture on the system level. In addition to that, the Hardware-Software Interface (HSI) is defined in this step.

The [ISO11][part 4, 6.4.2] provides requirements to the safety mechanisms to be defined, by giving guidance which topics need to be addressed. The topics that are addressed here (in particular in the requirements 6.4.2.2 and 6.4.2.3) need no further extensions, and can be used for fail-operational systems as well.

The intention of the following extension proposal is to address fail-operational behaviour explicitly:

Extension and optimisation proposal: Requirement on safety mechanisms focusing on random HW failures

7.4.4.1 Measures for detection and control, or mitigation of, *or tolerance of* random hardware failures shall be specified with respect to the system design given in 7.4.1 (System design specification and technical safety concept).

Note: the requested change is marked italic.

It has to be emphasized that the aspects addressed in the concept phase (e.g. parameters of fail-operational behaviour) shall be detailed further during the system development phase. But still, the framework provided by the [ISO11][part 4] is sufficient for this purpose.

Nonetheless, it is important to emphasize the importance of the HSI for fail-operational systems. As already described by [Dub13], the physical architectures applied for fail-operational systems are identical to those on the HW level. These topologies are based on physical redundancies. These design patterns can cover random HW failures, and – depending on the level of diversity – systematic failures as well. Therefore on the purely static level of the system architecture practically only the HW redundancies are visible. But underneath, on the functional and logical level, the redundant HW and SW architectures interact.

A very simple example is shown in figure 4.2. As it can be seen, the sub-figure a) shows a static architecture, with one logic module, three sensors in presumably a TMR arrangement, and two actuators in seemingly a dynamic redundant setup. As already mentioned above, these topologies are actually fault tolerant HW topologies. Sub-figure b) reveals why the HW-SW interaction is so important in these redundant architectures. Colour codes show if a certain function is realised in HW, SW or both. Therefore, potential fail-operational aspects of the HSI need to be investigated further. The [ISO11] [part 4, clause 7.5.6] defines requirements on the HSI. The following aspects are of particular importance for fail-operational systems:

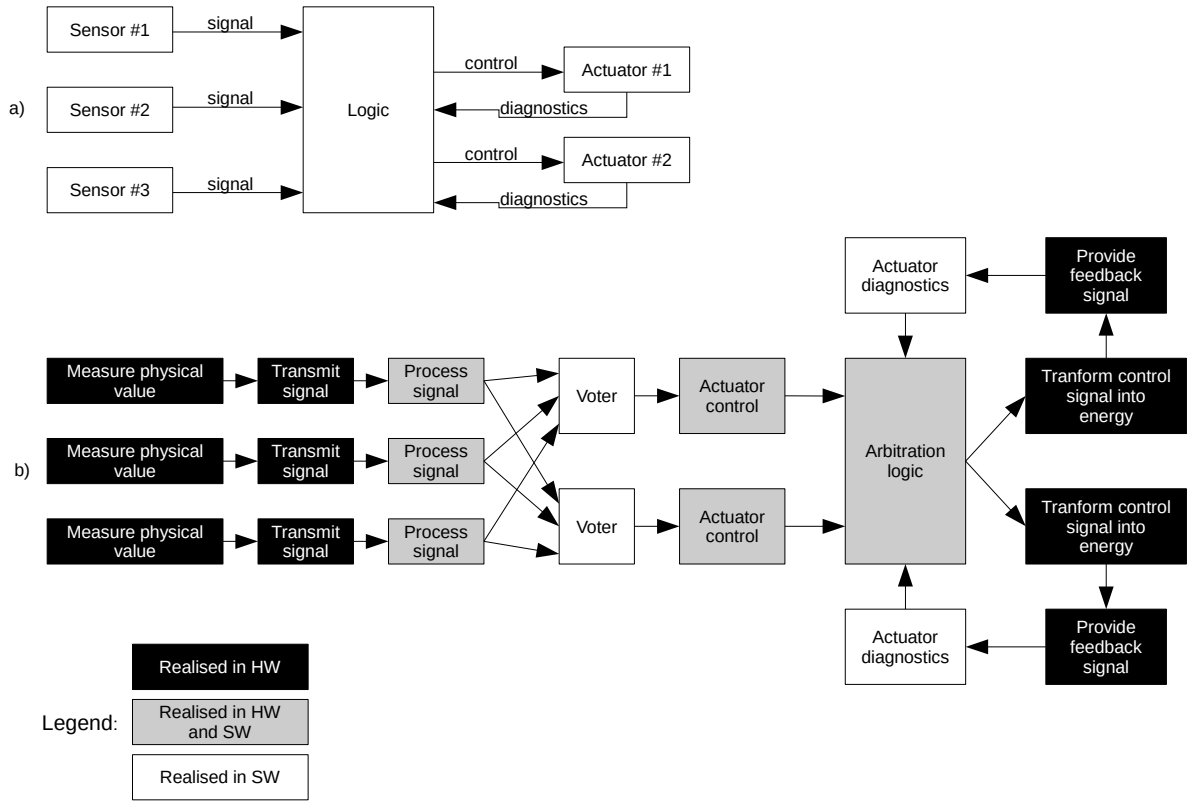


Figure 4.2: Layers of a redundant system architecture - a) Static view; b) Functional view

- *hardware features that support software partitioning*: In case of multi-version SW redundancy, independence of the parallel SW paths is essential. HW features that can support this represent a connection point between the HW and SW architectures. In figure 4.2 b) the parallel SW functions (e.g. signal processing, voter, actuator diagnostic) shall be independent from each other.
- *shared and exclusive use of hardware resources:* In case of multi-version SW redundancy, it has to be clarified which resources are used by both SW paths. Obviously, this represents a dependency (see section 4.5 for further details) that needs to be dealt with. In figure 4.2 b) the parallel SW functions (e.g. signal processing, voter, actuator diagnostic) may run on the same microcontroller, if it is not replicated (see section 4.4.3 for further details, why this may be appropriate). In that case the resource sharing concept is of paramount importance.
- *diagnostic features concerning the hardware, to be realised in software:* fault-tolerant HW redundancy can be supported by SW safety mechanisms. In figure 4.2 b) the actuator diagnostic functions, the arbitration logic, the voters, etc. are implemented purely or partly in SW. This means that the redundant HW architectures are completed by SW safety mechanisms.

As it can be seen by these examples, the SW and HW architectures and their redundancy concepts cannot be separated but need to be addressed on the system level as a whole, in order to fulfil the availability requirements defined by the safety goals and broken down further by the functional safety concept.

4.4.3 Fail-operational aspects in hardware development

When it comes to fail-operational properties of the HW, the left side of the V-model [VDI04] is what's interesting:

- specification of HW safety requirements, defining fault tolerance requirements and HW fault tolerance mechanisms,
- architectural design, specifying redundancy concepts on the HW level,
- calculation of the architectural metrics, focusing on the standard and fail-operational safety goals,
- evaluation of the safety goal violations due to random HW failures, focusing on the fail-operational safety goal.

So basically, for the HW it shall be answered, which mechanisms shall be implemented, by which architecture, and how good this resulting design is.

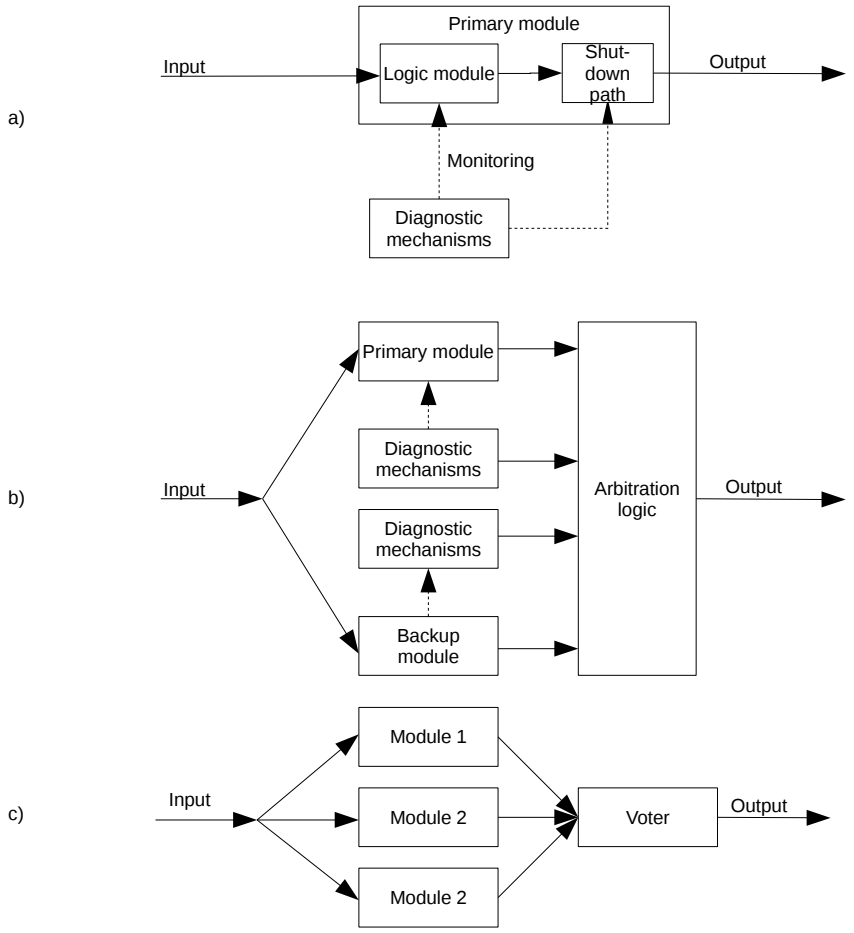


Figure 4.3: Redundant HW architectures - Comparison example

In order to gain a better understanding of the interaction between the points listed above, three typical topologies (figure 4.3) will be investigated using a simplified FMEDA and an FTA: a simple fail-safe architecture, a static redundant architecture with hot standby and a TMR architecture [Hireda]. This comparison will provide valuable information about the challenges of upgrading existing fail-safe architectures to fail-operational ones. The most important assumption of this example is that the logic, i.e. the function it implements will remain the same even if fail-operational behaviour is required. Therefore, the logic modules have nearly the same design. Due to the design diversity that is needed to achieve sufficient independence, completely identical design is not acceptable. It can be therefore assumed that the failure modes and failure rates of the redundant logic modules are similar or even identical. The dynamic redundant and the TMR architectures were the first choice due to the number of additional modules needed. Other architectures need even more additional HW, leading to an unacceptable increase of cost, weight and required packaging space. These architectures are clearly very

schematic and overly simplified but are still appropriate to model the challenges that designers will face when optimising the qualitative and quantitative failure behaviour of these elements.

The investigation of the three topologies will follow the same scheme:

1. Perform a simplified FMEDA and an FTA.
2. Interpret the qualitative analysis results by identifying which faults of these architectures are typically safe faults, SPF-s, RF-s and MPF-s.
3. Determine typical relative failure rates to these categories. These failure rates represent typical magnitudes of typical failures of these architectures. The basic analysis will then be performed in the subsequent step. The sensitivity of the results to changes in the assumed failure rates is then additionally investigated in the last step (section 4.4.3).
4. Derive the three metrics of the [ISO11] (SPFM, LFM and PMHF) from the FMEDA and the FTA.
5. Analyse the results' sensitivity to changes in the assumed failure rates.

In order to limit the complexity of the example and to stay focused on the comparison of the architectures, the applied analyses employ several simplifications:

- Conflicts between different safety goals of the system are not considered.
- The failure modes are kept as simple as possible. In fail-safe systems the 50% - 50% safe - dangerous failure model is often applied as simplification (e.g. [Lyo92]). In that case, it is assumed that the 50% dangerous portion leads to an active safety goal violation (e.g. self-steering in case of an EPS). The 50% safe fraction would either lead to a passive system (i.e. into the safe state) or would have no significant effect. In this comparison this 50-50% split is used. The conservative assumption is made that the 50% portion formerly regarded as safe leads to the loss of the function (i.e. to the violation of the fail-operational safety goal).
- The failure modes are considered independent.
- The failure rates represent magnitudes and not actual failure rates.
- In the FMEDA, all components are considered safety related.
- In the FTA, the probabilities are calculated as $P = \lambda t$ instead of $P = 1 - e^{-\lambda t}$. Assuming 10^4 hours lifetime and 2000 FIT failure rate, this simplification leads to 1% difference in the calculated probability.
- The FTA calculation is performed on the fault trees without any previous reorganisation. Since the trees are rather simple, due to the simplified fault model, the calculation accuracy is still sufficient.
- As lifetime, 10^4 hours have been assumed. The usual automotive lifetime for E/E components is 8×10^3 hours, the assumption is therefore conservative and simplifies the calculation.

Based on these analyses, the three architectures will be compared based on these qualitative and quantitative aspects. All points will be performed for two safety goals: for a standard safety goal (e.g. Prevent self-steering, for EPS systems) and the fail-operational safety goal (e.g. Prevent loss of steering support, for EPS systems).

The goals of this comparison are manifold:

- investigate the fault propagation paths in these architectures,
- evaluate the three metrics (i.e. the SPFM, LFM and PMHF) in these three architectures,
- identify main contributors and impact factors related to the three metrics.

Simple fail-safe architecture

First, the simple fail-safe architecture needs to be analysed. These architectures are common currently in the automotive industry, hence their fault propagation behaviour is well understood. In this case, there is no fail-operational safety goal, since without redundancy this architecture is not capable of fault tolerance. Nonetheless, the contributors to the loss of function failure mode will be investigated. The analysis of this architecture will be the reference for the other two.

Figures 4.4 and 4.5 show simplified fault trees of the fail-safe architecture, taking the violation of the standard safety goal and the loss of function hazard as top event, respectively. Based on these figures, a deeper look can be taken at the typical failure categories of the [ISO11][part 5] (see figure 2.9). Focusing on the standard safety goal, the faults can be categorised as follows:

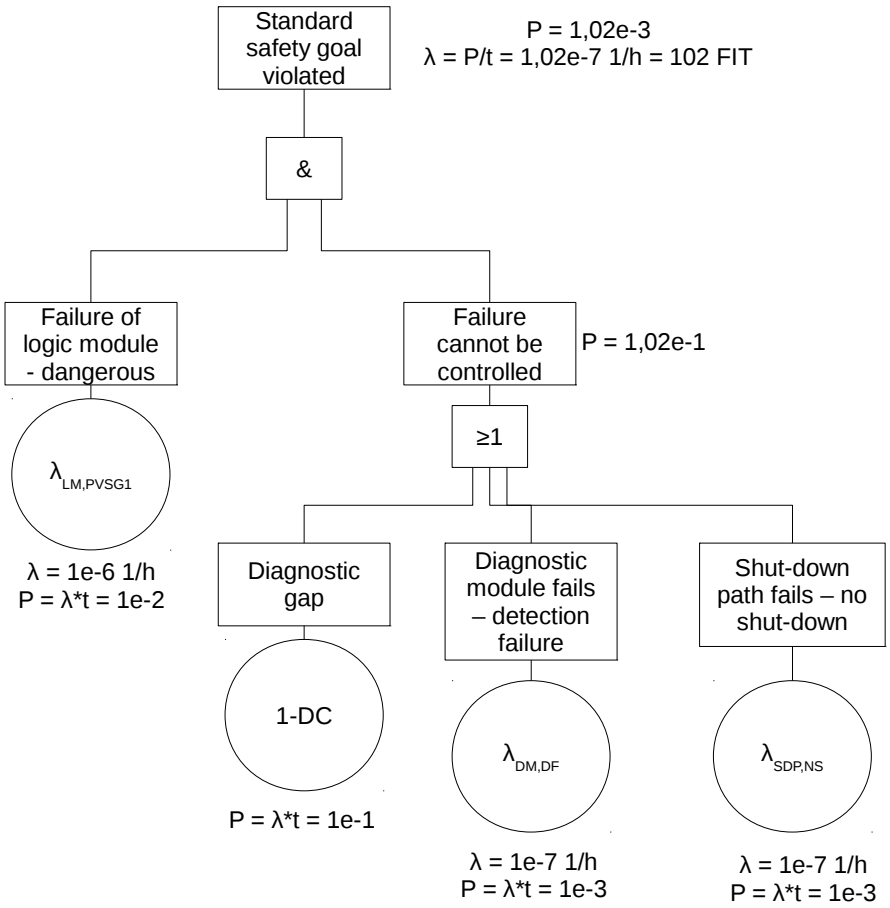


Figure 4.4: Simplified fault-tree of a fail-safe architecture - Violation of the standard safety goal

- *Safe faults* are in this architecture all faults that lead to the loss of the function (see figure 4.5): the safe faults of the logic module, leading to a deactivated output; the false alarm failures of the diagnostic module; and the faults of the shut-down path leading to an unnecessary shut-down.
- *Single point or residual faults* are the faults of the logic module that are not covered by safety mechanisms.
- *Detected multiple point faults* are the faults of the logic module covered by safety mechanisms, that would lead to the violation of the safety goal (in the absence of these safety mechanisms).
- *Perceived multiple point faults* are not shown in these figures.
- *Latent multiple point faults* are potentially the faults that lead to the corruption of the safety mechanisms: detection failures of the diagnostic mechanisms and faults of the shut-down path, leading to no shut-down. Obviously, these faults are only critical in the presence of a fault of the logic module. In usual single channel automotive architectures, like the one of the present example, the safety mechanisms are not all tested at start-up, and their failures are not reported to the driver. Therefore these faults remain latent.

For the quantitative analyses the following assumed failure rates are used.

- The logic module has a failure rate of $2000FIT$, distributed by using the common 50%-50% principle to safe and dangerous faults. In this case this means $1000FIT$ that leads to an active safety critical failure mode and $1000FIT$ leading to a passive module.
- The diagnostic module has a lower complexity than the logic itself, leading to a failure rate one

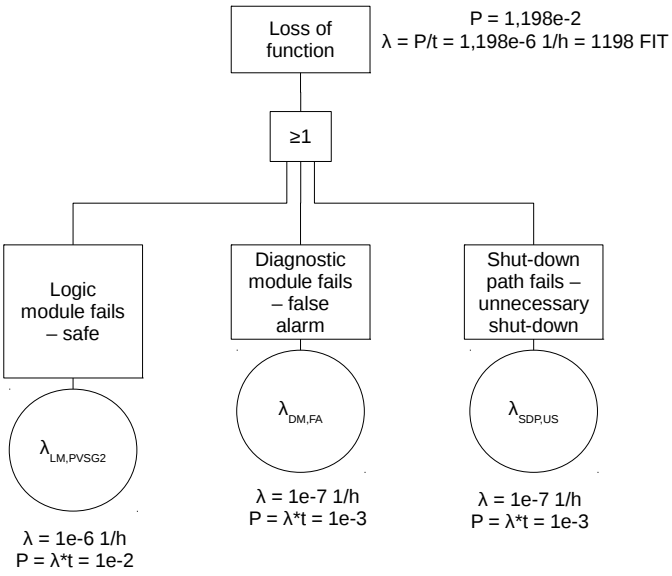


Figure 4.5: Simplified fault-tree of a fail-safe architecture - Loss of function hazard

magnitude lower (see [ISO11] [clause 5.4.7, NOTE]). Here, the same principle is applied, leading to $100FIT$ leading to a false alarm and $100FIT$ leading to a diagnostic module incapable of detecting failures of the logic module. The latter failure mode is called "detection failure" in this thesis. The rationale for this magnitude is that in common single channel systems, the number and complexity of HW parts dedicated to the monitoring of the logic module is significantly lower compared to the logic module. These HW components include read-back lines, watchdogs, etc.

- The shut-down path has a lower complexity than the logic itself, leading to a failure rate one magnitude lower than the logic module. By using the same principles as above, the resulting failure rate is $100FIT$ leading to an unnecessary shut-down and $100FIT$ leading to no shut-down. The rationale for this magnitude is that the number and complexity of the HW parts involved in the shut-down paths are significantly lower relatively to the logic module. These include relays, switches, solenoid valves, etc.

Besides that, a DC of 90 % is used as a conservative assumption. Note, that state-of-the-art single channel systems can reach much higher DC values.

Based on the results of the simplified FMEDA (see figure 4.6) and the FTA (see figure 4.4 and 4.5), the following can be stated:

- The *SPFM* depends mainly on the diagnostic coverage of the safety mechanisms (see 2.1) and on the portion of safe faults. Depending on the quality of the diagnostic mechanisms, the SPFM can be brought easily above 90%, even above 99% if necessary.
- As already shown in [Sch13], these topologies are not prone to *MPF,L-s*. The main reason is that the safety mechanisms of these systems are mainly based on detection and reaction. Besides that a good diagnostic coverage of the faults that would lead to a safety goal violation, also contributes to a low SPF/RF failure rate. These two factors lead to a high *LFM* even without dedicated safety mechanisms. In the example none of the MPF-s of the diagnostic module and the shut-down path are covered by safety mechanisms, still the $LFM = 91,3\%$. Therefore, start-up tests for these two elements are not necessary.
- The *PMHF* depends mainly on the initial safety related failure rate and on the SPFM. Even values below $10FIT$ are manageable for the standard safety goal, but they are not for the loss of function hazard.
- For single channel systems it can be stated that $PMHF \approx \lambda_{SPF/RF}$ since the diagnostic gap of the

		Single point fault evaluation				Multiple point fault evaluation			
Failure mode	Failure rate [FIT]	Safety goal violated?	Related safety mechanism	DC_{SPF}	$\lambda_{SPF/RF}$	Safety goal violated?	Related safety mechanism	DC_{MPF}	$\lambda_{MPF,L}$
Logic module fails – dangerous	1000	1	SM1	90%	100	0	Not necessary	0%	0
Logic module fails – safe	1000	0	Not necessary	0%	0	0	Not necessary	0%	0
Diagnostic module fails – detection failure	100	0	Not necessary	0%	0	1	None	0%	100
Diagnostic module fails – false alarm	100	0	Not necessary	0%	0	0	Not necessary	0%	0
Shut-down path fails – no shut-down	100	0	Not necessary	0%	0	1	None	0%	100
Shut-down path fails – unnecessary shut-down	100	0	Not necessary	0%	0	0	Not necessary	0%	0
	$\sum\lambda_{SR}$= 2400 FIT			$\sum\lambda_{SPF/RF}$= 100 FIT				$\sum\lambda_{MPF,L}$= 200 FIT	
	SPFM = 95,83%								
	LFM = 91,30%								

Figure 4.6: Simplified FMEDA of a fail-safe architecture - Standard safety goal

safety mechanism is dominating the related branches of the fault tree.

Dynamic redundant architecture with hot standby

Figures 4.7 and 4.8 show simplified fault trees of the dynamic redundant architecture. As it can be seen, the fault propagation in this architecture is very similar to that of the fail-safe architecture discussed in the section above, regarding the standard safety goal. The reason for this is that the basic principle of the implemented safety mechanisms is the same: detect failures by diagnostics and react on them. The main difference, obviously is, that whilst in case of the fail-safe topology the reaction is a simple shut-down. In the dynamic redundant architecture, the output of the backup module is routed through by the architecture logic. Another difference is that due to the redundant architecture, more MPF-s have been introduced.

Based on the fault trees and the FMEDA, the typical failure categories can be identified. For the standard safety goal:

- *Safe faults* are the ones leading to a passive system: passive faults of the primary and backup modules, passive faults of the arbitration logic and MPF-s of the third degree.
- *Single point or residual faults* are the active faults of the primary module that are not covered by the safety mechanisms and the active faults of the arbitration logic.
- *Detected multiple point faults* are the active faults of the primary and backup modules detected by the primary and backup diagnostic modules, respectively.
- *Perceived multiple point faults* are irrelevant for the investigation in this thesis.
- *Latent multiple point faults* are the faults of the diagnostic modules leading to undetected faults in the primary and backup modules, faults of the arbitration logic that would prohibit it to switch to the backup module. Besides that the undetected dangerous portion of the backup module is also latent. There is one type of fault that may remain latent depending on the warning and degradation concept: if the arbitration logic already switches to the backup module although the primary one is functioning well. If this circumstance is reported to the driver, the fault is detected. If not, it is latent.

In case of the fail-operational safety goal, the fault classification looks very similar, but still different:

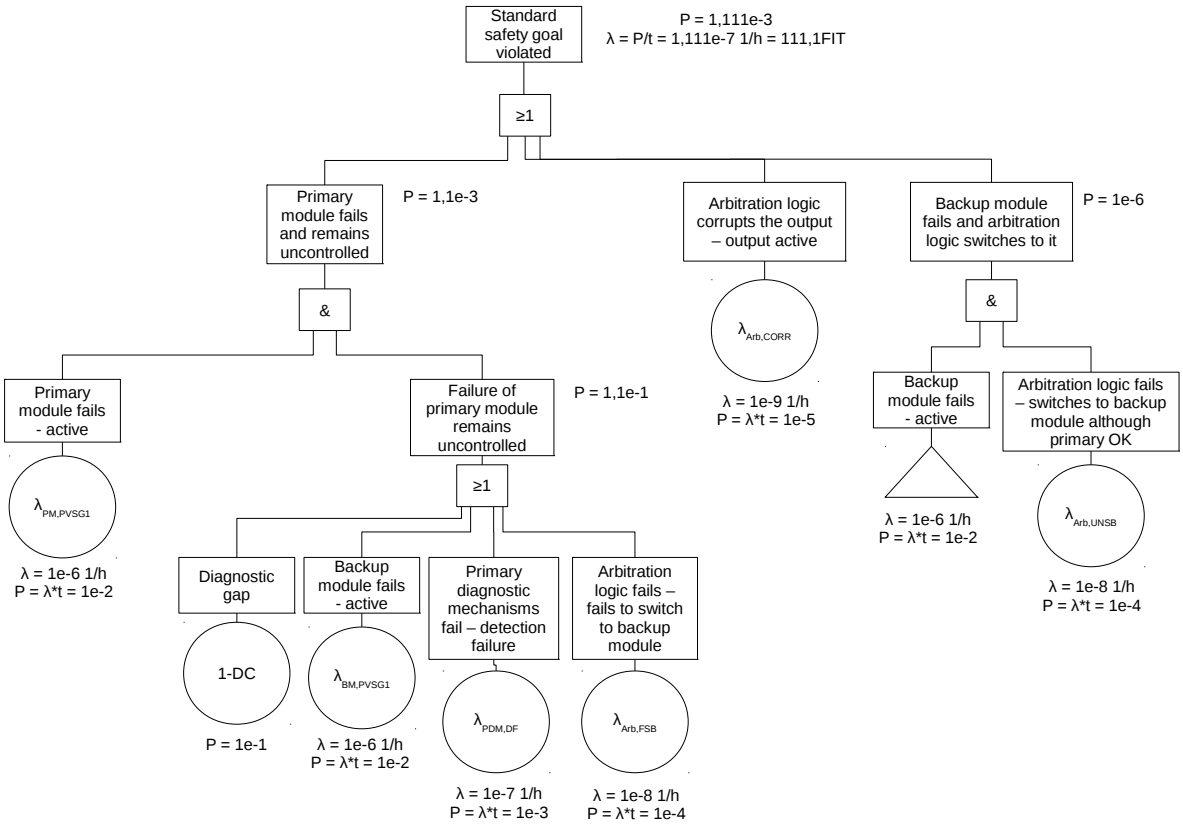


Figure 4.7: Simplified fault-tree of a dynamic redundant architecture - Standard safety goal

- *Safe faults* are the ones leading to an active, but malfunctioning system: active faults of the primary and backup modules, active faults of the arbitration logic and MPF-s of the third degree.
- *Single point or residual faults* are the passive faults of the primary module that are not covered by the safety mechanisms and the passive faults of the arbitration logic.
- *Detected multiple point faults* are the passive faults of the primary and backup modules detected by the primary and backup diagnostic modules, respectively.
- *Perceived multiple point faults* are irrelevant for the present investigation.
- *Latent multiple point faults* are exactly the same as in case of the standard safety goal.

Based on these two lists and on the figures 4.9 and 4.10, one thing is alarming: except for three fault types, that are MPF-s of third degree, all faults are dangerous either for the standard or for the fail-operational safety goal. The typical safe faults of fail-safe systems, leading to a passive system, are becoming dangerous for the fail-operational safety goal.

The assumed failure rates for the logic and diagnostic modules are the same as in the fail-safe case. What came in additionally, are the failure rates of the arbitration logic:

- The failure rate of the arbitration logic corrupting the output is estimated to $2 \times 1FIT$, based on the fact that this architectural element is very simple, consisting of a straightforward signal routing logic. This fault has been also split up to $1FIT$ leading to a corrupted but active output and $1FIT$ leading to a passive output.
- The failure rate of the arbitration logic failing to switch to the backup module or switching to it unnecessarily is estimated to be fairly low, being one magnitude lower than that of the diagnostic modules: $2 \times 10FIT$.

Based on the results of the simplified FMEDA (figures 4.9 and 4.10) and the FTA (see figures 4.7 and 4.8), the following can be stated:

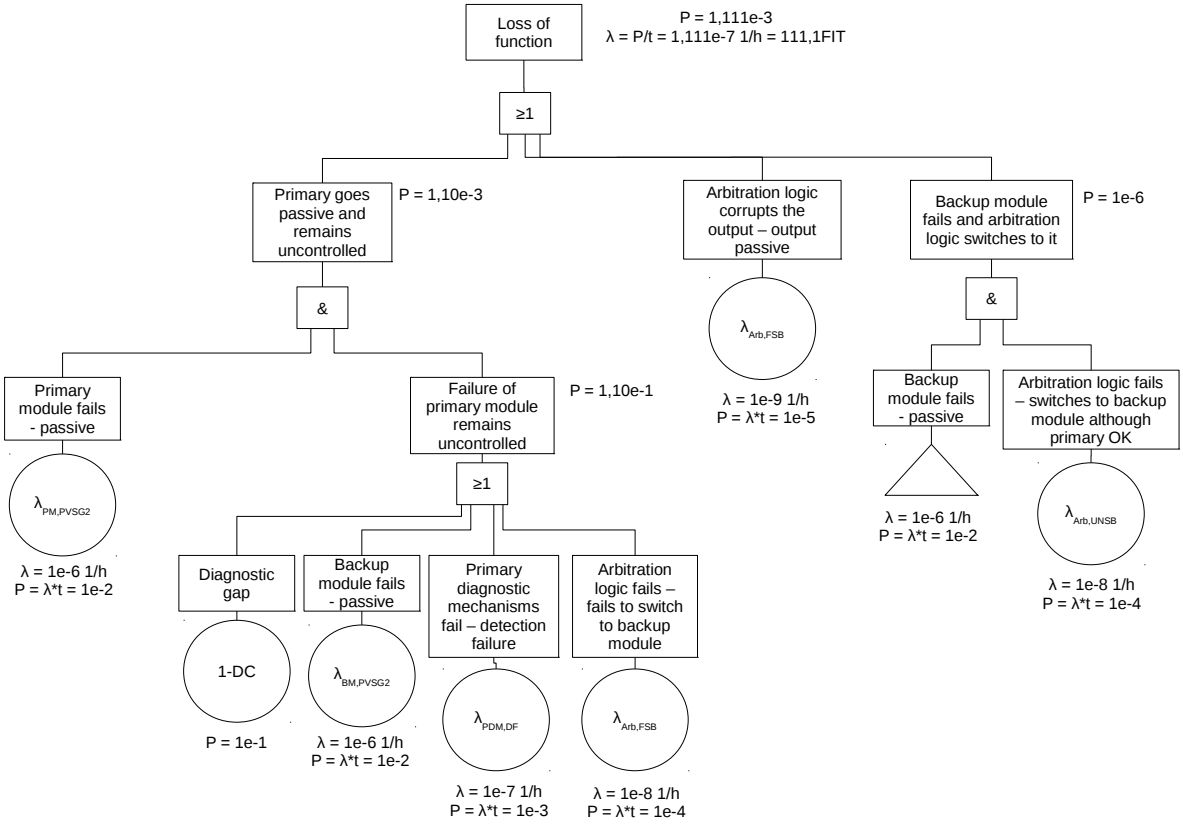


Figure 4.8: Simplified fault-tree of a dynamic redundant architecture - Fail-operational safety goal

- The *SPFM* of the dynamic redundant architecture is slightly above that of the fail-safe architecture, depending mainly on the diagnostic coverage of the safety mechanisms (see 2.1) and on the portion of safe faults. The reason for this improvement is the higher safe failure rate for both safety goals.
- Regarding the LFM the same phenomena can be seen as in the fail-safe case. The only difference is that some new latent fault sources are introduced by the arbitration logic, but the failure rate of those is relatively low. Since the safety mechanisms of this architecture are also based on detection and reaction, the LFM can be very high even without extra effort.
- The *PMHF* shows a bit different picture than in the case before. The not controlled failures of the primary module still dominate, but one level below the trend can be shifted. If the $DC \leq 90\%$ (like in figure 4.7), the diagnostic gap will still dominate. But as soon as the DC rises, probability of the dual-point fault of both the primary and the backup module failing simultaneously will be in the same magnitude. If assuming the same diagnostic coverage of the diagnostic modules, the PMHF of the dynamic redundant architecture is slightly worse than that of the fail-safe one: 111 vs 102. The reason is that although the diagnostic gap is still dominating, various sources of dual-point faults contribute to a higher overall safety goal violation probability.
- For this topology it can be already seen that $PMHF > \lambda_{SPF/RF}$ since the MPF-s contribute significantly to the PMHF.

TMR architecture

Figures 4.11 and 4.12 show the simplified fault trees of the TMR architecture, for both previously discussed safety goals. These fault trees show the main difference to the previously discussed fail-safe and dynamic redundant architectures: The TMR architecture is not based on detection and reaction but

		Single point fault evaluation				Multiple point fault evaluation			
Failure mode	**Failure rate [FIT]**	**Safety goal violated?**	**Related safety mechanism**	**DC_{SPF}**	**$\lambda_{SPF/RF}$**	**Safety goal violated?**	**Related safety mechanism**	**DC_{MPF}**	**$\lambda_{MPF,L}$**
Primary module failure – active	1000	1	SM1	90%	100	0	Not necessary	0%	0
Primary module failure – passive	1000	0	Not necessary	0%	0	0	Not necessary	0%	0
Primary diagnostic module – detection failure	100	0	Not necessary	0%	0	1	None	0%	100
Primary diagnostic module – false alarm	100	0	Not necessary	0%	0	0	Not necessary	0%	0
Backup module failure – active	1000	0	Not necessary	0%	0	1	SM2	90%	100
Backup module failure – passive	1000	0	Not necessary	0%	0	0	Not necessary	0%	0
Backup diagnostic module – detection failure	100	0	Not necessary	0%	0	0	Not necessary	0%	0
Backup diagnostic module – false alarm	100	0	Not necessary	0%	0	0	Not necessary	0%	0
Common module failure – active	0	1	None	0%	0	0	Not necessary	0%	0
Common module failure – passive	0	0	Not necessary	0%	0	0	Not necessary	0%	0
Arbitration fails to switch to backup module	10	0	Not necessary	0%	0	1	None	0%	10
Arbitration switches to backup module although primary OK	10	0	Not necessary	0%	0	1	None	0%	10
Arbitration corrupts the output – output is active	1	1	None	0%	1	0	Not necessary	0%	0
Arbitration corrupts the output – output is passive	1	0	Not necessary	0%	0	0	Not necessary	0%	0
	$\sum\lambda_{SR}$=	**4422 FIT**			**$\sum\lambda_{SPF/RF}$=**	**101 FIT**			**$\sum\lambda_{MPF,L}$=**	**220 FIT**
	SPFM =	**97,72%**								
	LFM =	**94,91%**								

Figure 4.9: Simplified FMEDA of a dynamic redundant architecture with hot stand-by - Standard safety goal

on fault tolerance by mitigation. Depending on the voter type (see section 3.1.2 for more details) the output of the voter can be one of the module outputs or a value generated based on all three of them.

Based on the fault trees and the FMEDA, the typical failure categories can be identified. For the standard safety goal:

- *Safe faults* are the ones leading to a passive system: passive faults of the three modules and faults of the voter leading directly to a passive output.
- *Single point or residual faults* are the active faults of the three modules that are not covered by the voter (due to diagnostic gap) and the active faults of the voter. It needs to be emphasized that voters may have a $DC < 100\%$. Especially inexact voters, where the agreement and disagreement of the voter inputs is depending on a pre-defined tolerance. This tolerance is not necessarily so low that any deviation within the limits is safe. The DC value can be even very low in case of voters that calculate their output from their inputs instead of routing one of them through.
- *Detected multiple point faults* are the active faults of the three modules detected by the voter if these faults are reported to the driver.
- *Perceived multiple point faults* are irrelevant for the present investigation.
- *Latent multiple point faults* are the controlled active faults of the three modules if they are not reported to the driver, and the faults causing the voter to transmit the wrong output (in case of some voter types).

The fault categories are almost identical in case of the fail-operational safety goal:

- *Safe faults* are the ones leading to active malfunctioning modules and faults of the voter leading directly to an actively malfunctioning output.
- *Single point or residual faults* are the passive faults of the three modules that are not covered by the voter (due to diagnostic gap) and the passive faults of the voter.
- *Detected multiple point faults* are the passive faults of the three modules detected by the voter if these faults are reported to the driver.
- *Perceived multiple point faults* are irrelevant for the present investigation.

		Single point fault evaluation				Multiple point fault evaluation			
Failure mode	Failure rate [FIT]	Safety goal violated?	Related safety mechanism	DC_{SPF}	$\lambda_{SPF/RF}$	Safety goal violated?	Related safety mechanism	DC_{MPF}	$\lambda_{MPF,L}$
Primary module failure – active	1000	0	Not necessary	0%	0	0	Not necessary	0%	0
Primary module failure – passive	1000	1	SM1	90%	100	0	Not necessary	0%	0
Primary diagnostic module – detection failure	100	0	Not necessary	0%	0	1	None	0%	100
Primary diagnostic module – false alarm	100	0	Not necessary	0%	0	0	Not necessary	0%	0
Backup module failure – active	1000	0	Not necessary	0%	0	0	Not necessary	0%	0
Backup module failure – passive	1000	0	Not necessary	0%	0	1	SM2	90%	100
Backup diagnostic module – detection failure	100	0	Not necessary	0%	0	0	Not necessary	0%	0
Backup diagnostic module – false alarm	100	0	Not necessary	0%	0	0	Not necessary	0%	0
Common module failure – active	0	0	Not necessary	0%	0	0	Not necessary	0%	0
Common module failure – passive	0	1	None	0%	0	0	Not necessary	0%	0
Arbitration fails to switch to backup module	10	0	Not necessary	0%	0	1	None	0%	10
Arbitration switches to backup module	10	0	Not necessary	0%	0	1	None	0%	10
Arbitration corrupts the output – output is	1	0	Not necessary	0%	0	0	Not necessary	0%	0
Arbitration corrupts the output – output is	1	1	None	0%	1	0	Not necessary	0%	0
$\sum\lambda_{SR}$=	4422 FIT			$\sum\lambda_{SPF/RF}$=	101 FIT			$\sum\lambda_{MPF,L}$=	220 FIT
SPFM =	97,72%								
LFM =	94,91%								

Figure 4.10: Simplified FMEDA of a dynamic redundant architecture with hot stand-by - Fail-operational safety goal

- *Latent multiple point faults* are the controlled passive faults of the three modules if they are not reported to the driver, and the faults causing the voter to transmit the wrong output (in case of some voter types).

Considering both the standard and the fail-operational safety goal, it is visible, that all failure modes of this simplified model are dangerous in some way, leading to the violation of one of the safety goals as SPF, RF of MPF.

The failure rates assumed for the modules are the same as in the two examples above. The failure rates of the voter are assumed to be the following:

- The failure rate of the voter picking the wrong input is assumed to be $10FIT$, since the complexity of this component shall be fairly low compared to the logic modules.
- The failure rate of the voter corrupting the output is assumed to be $1FIT$, supposing that the complexity of the signal routing logic is very low.
- The DC of the voter is assumed to be 99%, reflecting that voters tend to have high DC, and higher than the diagnostic modules of the fail-safe and dynamic redundant architectures.

Based on the simplified FMEDA (see figures 4.13 and 4.14) and the FTA (see figures 4.11 and 4.12), the following can be seen:

- The *SPFM* of the TMR topology is significantly better than that of the dynamic redundant architecture ($99,5\%$ versus $97,7\%$), mainly because of the higher diagnostic coverage of the voter. If the diagnostic module of the dynamic redundant variant is increased to 99%, the difference practically diminishes. Besides that, it has to be highlighted that the voting principle has major influence on the DC and on the SPFM and the PMHF. The voter has to be kept as simple as possible, since it is prone to SPF-s, decreasing the SPFM and increasing the PMHF.
- Regarding the LFM the behaviour of this architecture is different. As already predicted in [Sch13], this topology is prone to latent faults. Unless the faults of the modules are reported to the driver even if their effects are completely mitigated, the LFM will be very low. With the faults not reported, the $LFM = 50,2\%$ in this example, which is even far below the target for ASIL B. By applying an appropriate warning concept, the LFM can be easily increased above 90%

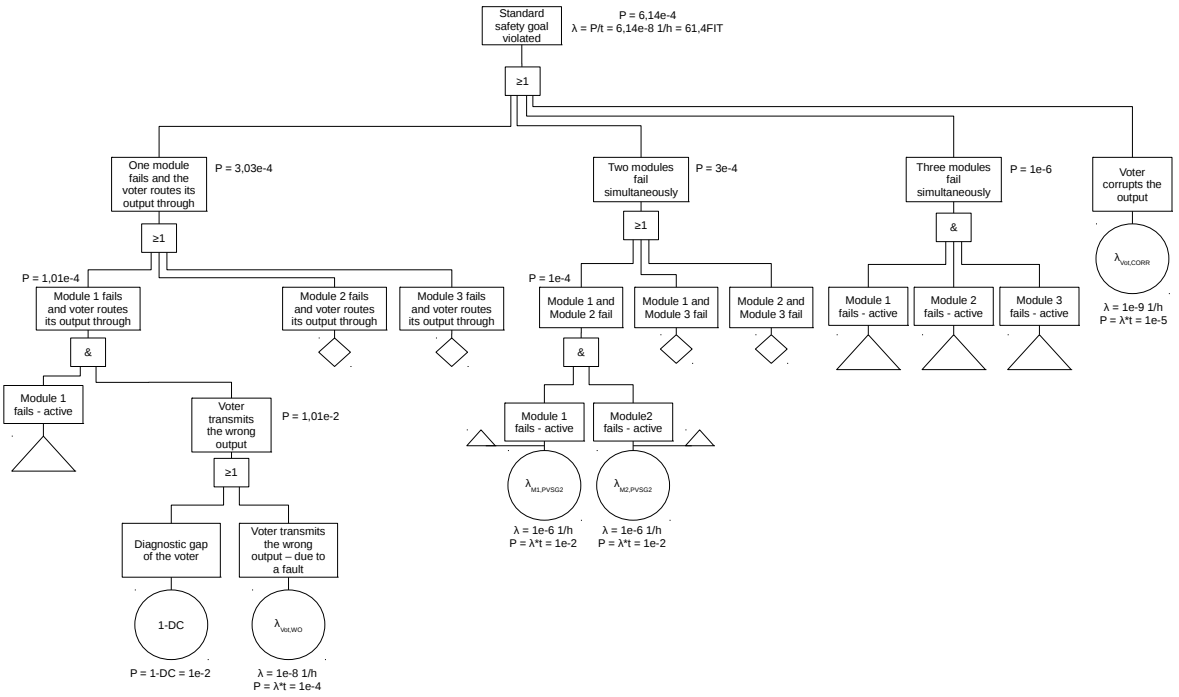


Figure 4.11: Simplified fault-tree of a TMR architecture - Standard safety goal

- The *PMHF* is not dominated by the undetected faults of the modules: the failure rate of the dual-point failures caused by the simultaneous faults of two modules are in the same magnitude. The PMHF is lower than that of the dynamic redundant variant (61FIT versus 111FIT), but mainly due to the higher DC. The case is very similar to the SPFM: if the DC of the diagnostic module of the dynamic redundant architecture is also set to 99%, its PMHF will be in the same magnitude and can be even lower than that of the TMR variant (22FIT versus 61FIT). The main reason for this is the lower overall failure rate of the dynamic redundant architecture.
- $PMHF > \lambda_{SPF/RF}$ is valid for this topology as well, since the MPF-s contribute significantly to the PMHF.

Extension - investigation of incompletely redundant architectures

As already shown in section 4.3.1, it is not required by the [ISO11] to reach absolute fault tolerance: SPF-s are allowed even for ASIL C and D. Therefore, it can be assumed that real-life architectures will not aim for complete replication (figure 4.15). It is of great interest, how the replication rate affects the three metrics investigated above.

Note, that only the effect on the fail-operational safety goal is investigated for incompletely redundant architectures, hence no diagnostic modules focusing on the common module are included. Those would be able to detect faults but no appropriate reaction would be possible.

In case of the dynamic redundant architecture, let λ_0, λ_1 and λ_2 denote the failure rates of the common module, of the primary module and of the backup module, respectively. Let p ($p = 0...1$) denote the *replication rate*, with regards to the entire failure rate of the logic. In this case the following equations reflect the failure rates of the modules:

$$\lambda = 2000FIT \tag{4.1}$$

$$\lambda_0 = (1 - p)\,\lambda \tag{4.2}$$

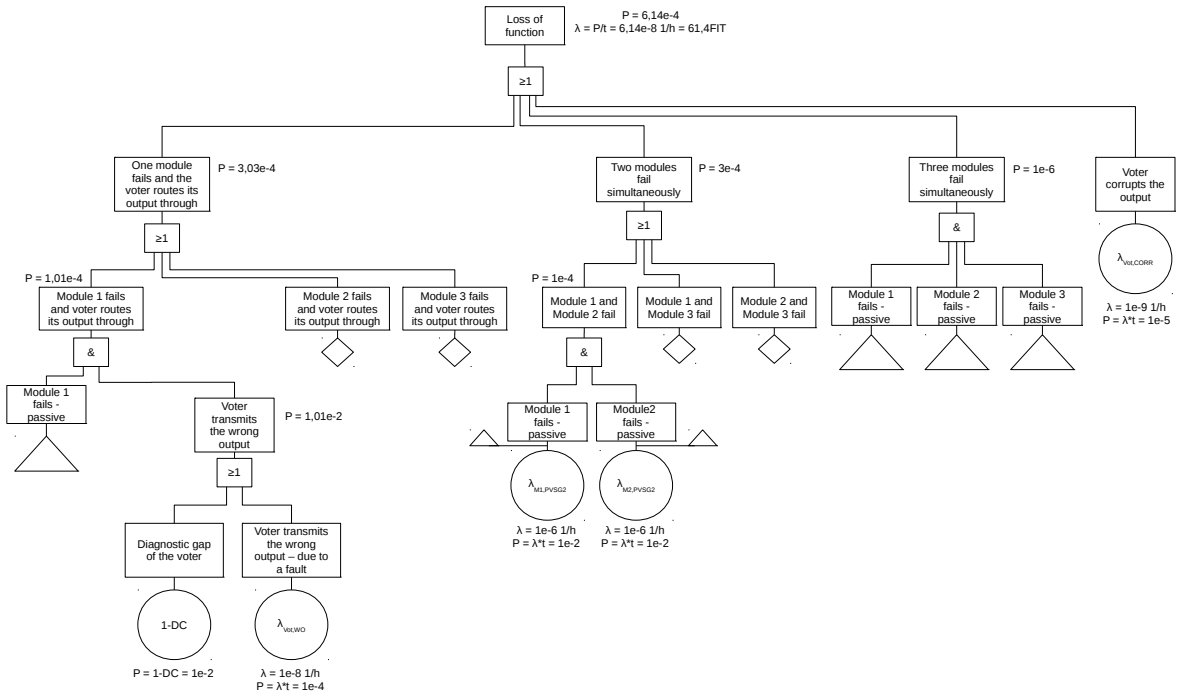


Figure 4.12: Simplified fault-tree of a TMR architecture- Fail-operational safety goal

$$\lambda_1 = \lambda_2 = p\lambda \tag{4.3}$$

Besides that the failure rate of the diagnostic modules can be calculated as:

$$\lambda_{diag0} = 200FIT \tag{4.4}$$

$$\lambda_{diag} = p\lambda_{diag0} \tag{4.5}$$

Note, that the failure rate of the arbitration logic is excluded from λ_0 to simplify the calculation.

In case of the TMR architecture, let λ_0, λ_1, λ_2 and λ_3 denote the failure rates of the common module and of the three parallel modules, respectively. Let p ($p = 0...1$) denote the replication rate, with regards to the entire failure rate of the logic. In this case the following equations reflect the failure rates of the modules:

$$\lambda = 2000FIT \tag{4.6}$$

$$\lambda_0 = (1 - p)\,\lambda \tag{4.7}$$

$$\lambda_1 = \lambda_2 = \lambda_3 = p\lambda \tag{4.8}$$

Note, that the failure rate of the voter is excluded from λ_0 to simplify the calculation.

The results for the dynamic redundant and for the TMR architecture can be seen in figures 4.16, 4.17, and 4.18. In both architectures the PMHF is decreasing and the SPFM is increasing at higher replication rates. The reason is obvious: faults of the common modules are SPF-s, therefore the λ_{SPF} will increase. The LFM is almost immune to the replication rate in both the dynamic redundant and in the TMR architecture.

In figures 4.16 and 4.17 the SPFM and LFM target values (see tables 2.2 and 2.3) are marked, to point out another interesting aspect. As it can be seen, the SPFM of the two fail-operational architectures are almost equal in the high replication range, but deviate significantly in the lower part of the diagram.

		Single point fault evaluation				Multiple point fault evaluation			
Failure mode	Failure rate [FIT]	Safety goal violated?	Related safety mechanism	DC_{SPF}	$\lambda_{SPF/RF}$	Safety goal violated?	Related safety mechanism	DC_{MPF}	$\lambda_{MPF,L}$
Module 1 fails – active	1000	1	SM1 (Voter)	99%	10	1	SM1 (Voter)	100%	0
Module 1 fails – passive	1000	0	Not necessary	0%	0	0	Not necessary	0%	0
Module 2 fails – active	1000	1	SM1 (Voter)	99%	10	1	SM1 (Voter)	100%	0
Module 2 fails – passive	1000	0	Not necessary	0%	0	0	Not necessary	0%	0
Module 3 fails – active	1000	1	SM1 (Voter)	99%	10	1	SM1 (Voter)	100%	0
Module 3 fails – passive	1000	0	Not necessary	0%	0	0	Not necessary	0%	0
Common module failure – active	0	1	None	0%	0	0	Not necessary	0%	0
Common module failure – passive	0	0	Not necessary	0%	0	0	Not necessary	0%	0
Voter transmits the wrong output	10	0	Not necessary	0%	0	1	None	0%	10
Voter corrupts the output – output is active	1	1	None	0%	1	0	Not necessary	0%	0
Voter corrupts the output – output is passive	1	0	Not necessary	0%	0	0	Not necessary	0%	0
$\sum\lambda_{SR}$=	6012 FIT			$\sum\lambda_{SPF/RF}$=	31 FIT			$\sum\lambda_{MPF,L}$=	10 FIT
SPFM =	99,48%								
LFM =	99,83%								

Figure 4.13: Simplified FMEDA of a TMR architecture with hot stand-by - Standard safety goal

The 60% that is often used for ASIL A (although not required by the [ISO11]) is already reached at about $p = 15\%$ in case of all three investigated variants. The 90% target for ASIL B is met at approximately $p = 70\%$ in both cases. For the very demanding 97% and 99% target for ASIL C and D, practically $p = 100\%$ is required. The LFM of all investigated variants is very high (assuming that the driver is warned in case of the TMR topology - see table 4.3), reaching even the 90 % target for ASIL D. The same evaluation would be pointless for the PMHF, since the assumed failure rates have no absolute relevance and are only intended to relatively compare the three architectures.

Sensitivity analysis focusing on the assumed failure rates and their distribution

As already stated before, the assumed failure rates (incl. their distribution) used for the FMEDA and the FTA are based on the industrial experience of the authors. But nonetheless, the sensitivity of the results to changes in these failure rates needs further investigation. The key assumption made in this section is, that the failure rates of the voter (in case of the TMR architecture), of the arbitration logic and of the diagnostic modules (in case of the dynamic redundant architecture) are lower or equal than that of the failure rate of the module in focus (i.e. the value λ in equations 4.1 and 4.6).

In general, parameters noted as f_x denote ratios between the failure rate λ (in equations 4.1 and 4.6) and failure rates of the elements of the investigated architecture. Parameters noted as $d_{S,x}$ symbolise distributions. In most cases the latter stands for safe-dangerous distributions, where the failure model consists of two (i.e. one safe and one dangerous) failure mode.

There are only two assumptions made at the start of this section related to the failure rates that are not investigated in the sensitivity analysis:

- The λ value remains untouched. The reason for this is that all failure rates represent magnitudes. Therefore their absolute values are not relevant only their proportion.
- The second one is the relation of the failure rate of the redundant modules. Since the functions of these modules are similar, it is highly unlikely that their failure rates will significantly deviate from each other.

Due to the number of the parameters investigated in this section, the sensitivity analysis is a multi-dimensional problem. In order to present the results in a comprehensible way, each of these parameters

		Single point fault evaluation				Multiple point fault evaluation			
Failure mode	Failure rate [FIT]	Safety goal violated?	Related safety mechanism	DC_{SPF}	$\lambda_{SPF/RF}$	Safety goal violated?	Related safety mechanism	DC_{MPF}	$\lambda_{MPF,L}$
Module 1 fails – active	1000	0	Not necessary	0%	0	0	Not necessary	0%	0
Module 1 fails – passive	1000	1	SM1 (Voter)	99%	10	1	SM1 (Voter)	100%	0
Module 2 fails – active	1000	0	Not necessary	0%	0	0	Not necessary	0%	0
Module 2 fails – passive	1000	1	SM1 (Voter)	99%	10	1	SM1 (Voter)	100%	0
Module 3 fails – active	1000	0	Not necessary	0%	0	0	Not necessary	0%	0
Module 3 fails – passive	1000	1	SM1 (Voter)	99%	10	1	SM1 (Voter)	100%	0
Common module failure – active	0	1	Not necessary	0%	0	0	Not necessary	0%	0
Common module failure – passive	0	1	None	0%	0	0	Not necessary	0%	0
Voter transmits the wrong output	10	0	Not necessary	0%	0	1	None	0%	10
Voter corrupts the output – output is active	1	1	None	0%	1	0	Not necessary	0%	0
Voter corrupts the output – output is passive	1	0	Not necessary	0%	0	0	Not necessary	0%	0
	$\sum\lambda_{SR}$= 6012 FIT				$\sum\lambda_{SPF/RF}$= 31 FIT				$\sum\lambda_{MPF,L}$= 10 FIT
	SPFM = 99,48%								
	LFM = 99,83%								

Figure 4.14: Simplified FMEDA of a TMR architecture with hot stand-by - Fail-operational safety goal

will be investigated separately, in combination with different replication rates. The other parameters will thereby remain unchanged, unless necessary to present a specific phenomenon. In those cases a clear indication will be given.

The parameters used for the sensitivity analysis (defined in detail above) are investigated in the following ranges:

- f_x values: $[x, \quad 1]$, where x represents their basic assumed values at the start of this section.
- $d_{S,x}$ values: $[0, \quad 1]$.
- DC values: the common 60%, 90% and 99% are used in the analysis.

For each parameter, a figure displays its effect on the SPFM, LFM and PMHF. Besides that, an indication is given how the level of necessary replication rate is affected.

To support the sensitivity analysis of the dynamic redundant architecture, the parameters $f_{diag,P}$, $f_{diag,B}$, f_{arb1}, f_{arb2}, DC_{PM}, DC_{BM}, $d_{S,CM}$, $d_{S,PM}$, $d_{S,BM}$, $d_{S,PDM}$, $d_{S,BDM}$, $d_{S,arb1}$ and $d_{S,arb2}$ are introduced, in accordance with the following equations (for the notations related to the failure rates see Figure 4.8):

$$\lambda_{PM,PVSG2} = \lambda \times p \times (1 - d_{S,PM}) \tag{4.9}$$

$$\lambda_{PDM,DF} = \lambda \times p \times f_{diag,P} \times (1 - d_{S,PDM}) \tag{4.10}$$

$$\lambda_{BM,PVSG2} = \lambda \times p \times (1 - d_{S,BM}) \tag{4.11}$$

$$\lambda_{BDM,DF} = \lambda \times p \times f_{diag,B} \times (1 - d_{S,BDM}) \tag{4.12}$$

$$\lambda_{CM} = \lambda \times (1 - p) \times (1 - d_{S,CM}) \tag{4.13}$$

$$\lambda_{Arb,UNSB} = \lambda \times f_{arb1} \times d_{S,arb1} \tag{4.14}$$

$$\lambda_{Arb,FSB} = \lambda \times f_{arb1} \times (1 - d_{S,arb1}) \tag{4.15}$$

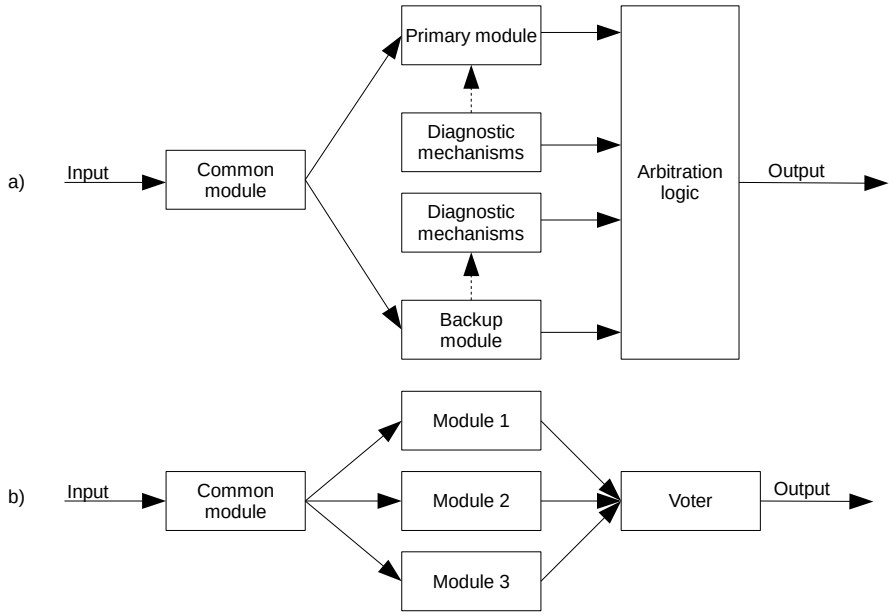


Figure 4.15: Incompletely redundant HW architectures a) incompletely redundant dynamic redundant architecture with hot standby, b) incompletely redundant TMR architecture

$$\lambda_{Arb,CORR} = \lambda \times f_{arb2} \times (1 - d_{S,arb2}) \tag{4.16}$$

To reconnect to section 4.4.3, the assumed values for the parameters defined above were:

- $f_{diag} = 0.1$
- $f_{arb1} = 0.01$
- $f_{arb2} = 0.001$
- For DC_{PM} and DC_{BM} both the values 0.99 and 0.9 were investigated, and were assumed as equal.
- $d_{S,CM} = d_{S,PM} = d_{S,BM} = d_{S,PDM} = d_{S,BDM} = d_{S,arb1} = d_{S,arb2} = 0.5$

Figures 4.19 - 4.31 show the results. In each figure, the SPFM, LFM and PMHF are shown depending on the parameter in focus. Since the dependency on the replication rate still applies, each figure contains the $p = 0\%$, 50% and 100% cases.

The sensitivity of the results to the parameters related to the dynamic redundant architecture, defined above, can be summarised as the following:

- With increasing $f_{diag,P}$ (figure 4.19), the SPFM and the PMHF are increasing, while the LFM is decreasing. The reason for this is that this increases the portion of the latent multiple point faults. Since this multiple point fault does not affect the PMHF as significantly as it does the SPFM and LFM, an increase of the $f_{diag,P}$ may affect the necessary replication rate positively or negatively, depending on the target ASIL.
- The sensitivity to $f_{diag,B}$ looks differently (Figure 4.20), since the failure rate of the backup diagnostic module affects the three metrics in an other way. In this case, the SPFM and LFM are increasing, while the PMHF remains untouched. The reason for this is that in accordance with the ISO 26262-5:2011, clause B.1, NOTE 1, multiple point faults of order higher than two can be regarded as safe (see also in figure 4.10). Therefore higher failure rates of the backup diagnostic module are only increasing the safe failure rate. In order to avoid biased results caused by artificially increasing the failure rate of the backup diagnostic module to seemingly improve the SPFM and LFM, a qualitative review of the analysis and of the design is necessary. Therefore, it can be stated, that the necessary replication rate is not affected, or lower in very rare cases, if the $f_{diag,B}$ increases.

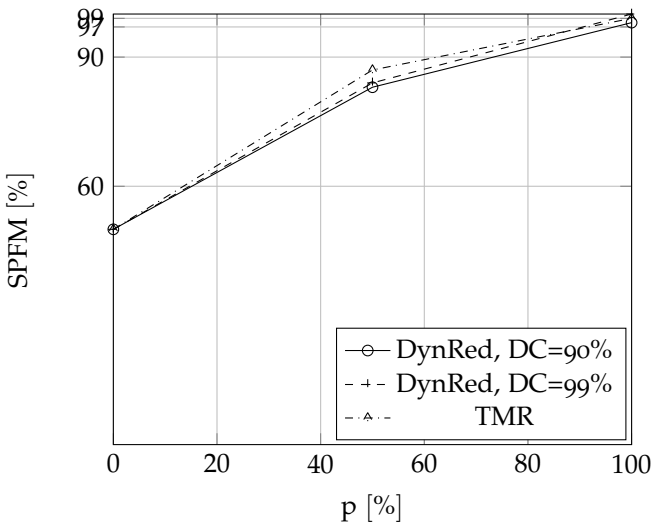


Figure 4.16: SPFM - Comparison of incompletely redundant architectures

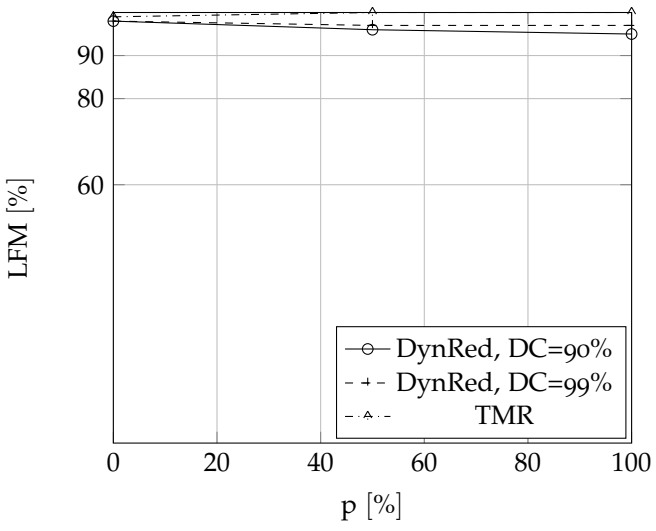


Figure 4.17: LFM - Comparison of incompletely redundant architectures

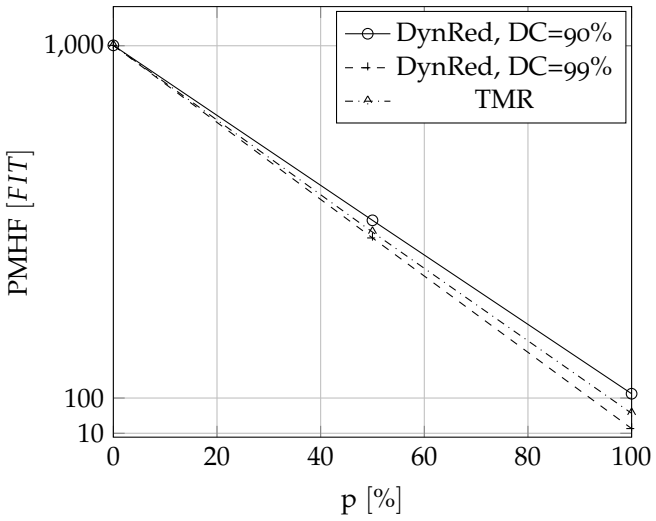


Figure 4.18: PMHF - Comparison of incompletely redundant architectures

- If the f_{arb1} increases (figure 4.21) the SPFM and the PMHF do as well. The LFM, on the other hand, is decreasing. The rationale is that the portion of latent multiple point faults is increasing. Since the PMHF is not affected as significantly as the two architectural metrics, the f_{diag} may affect the necessary replication rate both in a positive or a negative way.
- The sensitivity to the parameter f_{arb2} is different (figure 4.22): since this one affects the single point failure rate, both the SPFM and PMHF are worse if the f_{arb2} is higher. The LFM on the other hand is slightly improving. Therefore, if the arbitration logic is prone to failures corrupting its output, the necessary replication rate will be definitely higher.
- An increase of DC_{PM} (figure 4.23) affects all parameters positively. This is obvious, since this parameter describes the "quality" of the diagnostic algorithms of the primary diagnostic module. Due to this reason, the necessary replication rate decreases with increasing DC_{PM}.
- Since the quality of the safety mechanisms implemented in the backup diagnostic module only affect potentially latent faults, the parameter DC_{BM} (figure 4.24) can influence only the LFM. The SPFM and PMHF remain intact. Therefore, with higher DC_{BM}, the level of the necessary replication rate declines.
- The $d_{S,CM}$ affects architectures with low replication rates obviously harder (figure 4.25). In that case, lower $d_{S,CM}$ leads to worse SPFM, LFM and PMHF. Due to this reason, lower $d_{S,CM}$ value requires higher level of replication for a given ASIL.
- Since the primary module does not have potentially latent failures, the $d_{S,PM}$ does not impair the LFM. The SPFM and PMHF get worse if the $d_{S,PM}$ decreases. The magnitude of this effect is higher at higher replication rates, since in that case the overall failure rate of the primary module is higher (equation 4.3). Thus, higher replication rate is required if the $d_{S,PM}$ is lower.
- Contrary to the safe-dangerous distribution of the primary module, that of the backup module ($d_{S,BM}$) mainly affects the LFM, since the backup module has multiple point failures only (figure 4.27). Consequently, with increasing $d_{S,BM}$ the LFM and PMHF are improving. Which means that the necessary replication rate would be lower.
- Since the primary diagnostic module has multiple point faults only, an increase in the parameter $d_{S,PDM}$ leads to an increase in the LFM and in a very slight decrease in the PMHF (figure 4.28). The latter is more affected if the failure rate of the primary diagnostic module (i.e. the parameter $f_{diag,P}$) is higher, but even with $f_{diag,P} = 1$, this effect is almost negligible. Thus, if the $d_{S,PDM}$ rises, the necessary replication rate will decrease.
- The $d_{S,BDM}$ has no effect on the three metrics (figure 4.29), due to the same reason as in case of the $f_{diag,B}$.
- The $d_{S,arb1}$ (figure 4.30) has practically no effect on the three parameters. This is valid even if the

Parameter (↑)	SPFM	LFM	PMHF	p_{nec}
$f_{diag,P}$	↑	↓	↑	↑ / ↓
$f_{diag,B}$	↑	↑	o	o / ↓
f_{arb1}	↑	↓	↑	↑ / ↓
f_{arb2}	↓	↑	↑	↑
DC_{PM}	↑	↑	↓	↓
DC_{BM}	o	↑	o	↓
$d_{S,CM}$	↑	↑	↓	↓
$d_{S,PM}$	↑	o	↓	↓
$d_{S,BM}$	o	↑	↓	↓
$d_{S,PDM}$	o	↑	↓	↓
$d_{S,BDM}$	o	o	o	o
$d_{S,arb1}$	o	o	o	o
$d_{S,arb2}$	↑	↑	↓	↓

Table 4.1: Sensitivity of the SPFM, LFM, PMHF and the necessary replication rate to the parameters of the dynamic redundant architecture

failure rate of the arbitration logic is high, due to an increase in the parameter f_{arb1}. Figure 4.30 shows the case, where $f_{arb1} = 1$. The SPFM is not affected, since the two related failure modes of the arbitration logic (i.e. that the arbitration logic fails to switch to the backup module, or that it switches to it unnecessarily) are both multiple point faults. The LFM is not affected, since both of these failure modes have the potential to be latent, therefore their distribution is irrelevant from the LFM's perspective. The PMHF is slightly worse if the $d_{S,arb1}$ increases, but this difference is negligible. Therefore, the necessary replication rate is not affected by the $d_{S,arb1}$.

- The $d_{S,arb2}$ (figure 4.31) has practically no effect on the three metrics as long as the f_{arb2} is small (as assumed in this section). Figure 4.31 shows the sensitivity assuming $f_{arb2} = 1$, which is highly unlikely, but allows the reader to see how the SPFM, LFM and PMHF are affected. As it can be seen from the mentioned figure, all three metrics improve if the $d_{S,arb2}$ increases. The reason for this behaviour is that the $d_{S,arb2}$ affects the failure rate related to the failure mode "Arbitration corrupts the output (output is passive)", which is considered being safety critical in this fail-operational context. Hence, the necessary replication rate decreases with an increasing $d_{S,arb2}$.

As it can be seen from the list above, most parameters can affect the necessary replication rate. But it is also obvious that the level of impact depends on the other parameters. The implications given above and the Figures 4.19-4.31 offer guidance to deal with a specific practical problem. In addition to that, table 4.1 gives an overview of the results. Assuming that the parameters in the first column increase, the effects on the three metrics and the dependencies on other parameters are given in the table. Note that the symbols ↑ and ↓ denote increase or decrease in that particular metric, respectively. The entry o stands for no effect.

In the next step, to support the sensitivity analysis of the TMR architecture, the parameters f_{vote1}, f_{vote2}, DC_{TMR}, $d_{S,M1}$, $d_{S,M2}$, $d_{S,M3}$, $d_{S,CM}$ and $d_{S,voter}$ are introduced, in accordance with the following equations (for the notations related to the failure rates see Figure 4.12):

$$\lambda_{M1,PVSG2} = \lambda \times p \times (1 - d_{S,M1}) \tag{4.17}$$

$$\lambda_{M2,PVSG2} = \lambda \times p \times (1 - d_{S,M2}) \tag{4.18}$$

$$\lambda_{M3,PVSG2} = \lambda \times p \times (1 - d_{S,M3}) \tag{4.19}$$

$$\lambda_{CM} = \lambda \times (1 - p) \times (1 - d_{S,CM}) \tag{4.20}$$

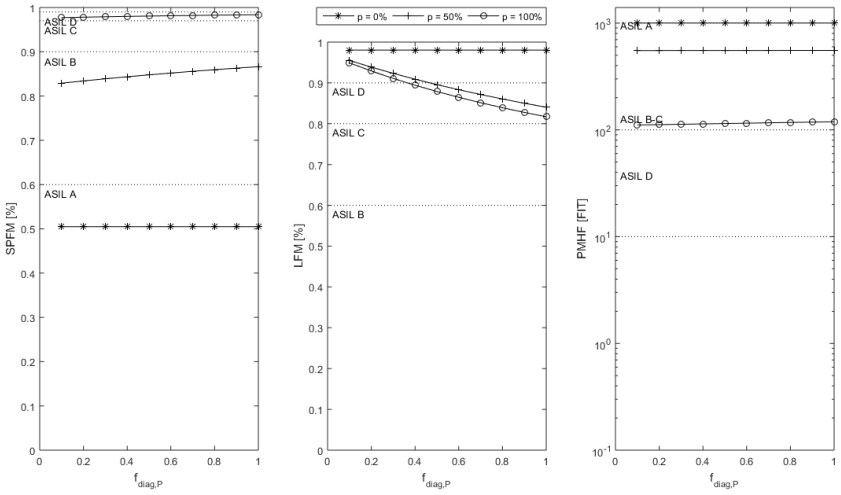


Figure 4.19: Sensitivity of the results to the parameter $f_{diag,P}$

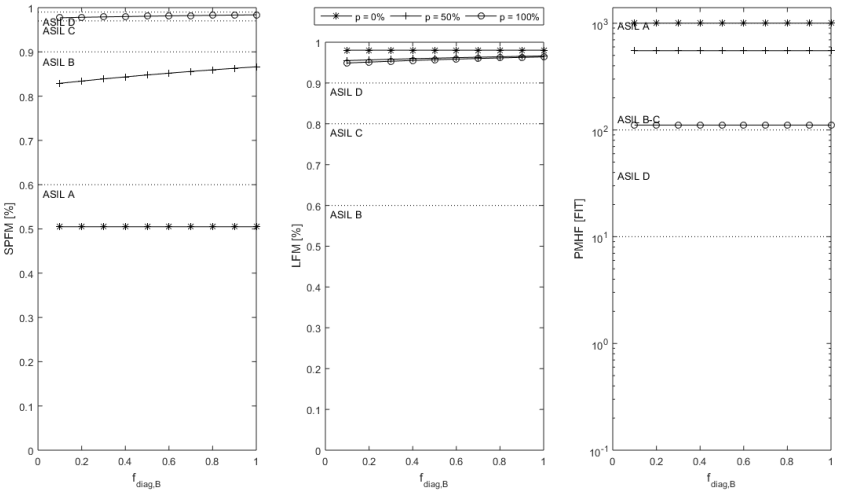


Figure 4.20: Sensitivity of the results to the parameter $f_{diag,B}$

$$\lambda_{Vot,WO} = \lambda \times f_{vote1} \tag{4.21}$$

$$\lambda_{Vot,CORR} = \lambda \times f_{vote2} \times (1 - d_{S,voter}) \tag{4.22}$$

The assumed values in section 4.4.3 for the parameters defined above were:

- $f_{vote1} = 0.005$
- $f_{vote2} = 0.001$
- $DC_{TMR} = 0.99$
- $d_{S,M1} = d_{S,M2} = d_{S,M3} = d_{S,CM} = d_{S,voter} = 0.5$

The sensitivity of the results to the parameters related to the TMR architecture, defined above, can be summarised as the following:

- With increasing f_{vote1} (figure 4.32), the SPFM and PMHF are increasing, while the LFM is declining. The reason for this is that if the f_{vote1} gets higher, the portion of latent multiple point

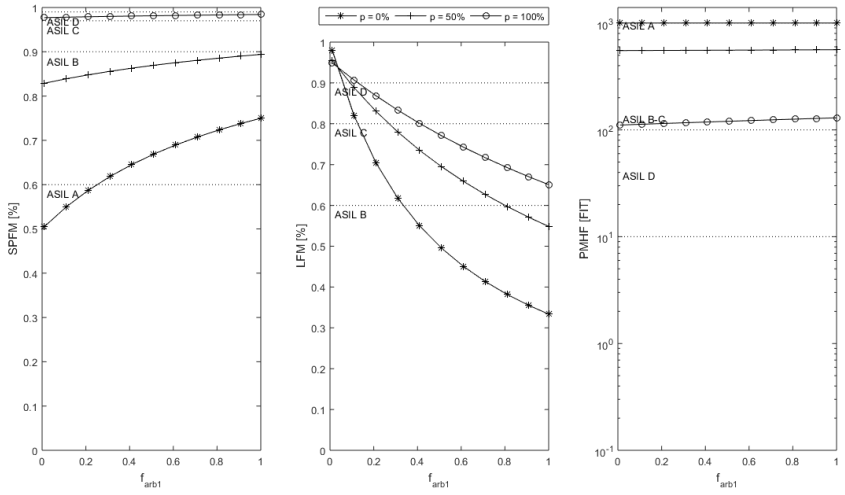


Figure 4.21: Sensitivity of the results to the parameter f_{arb1}

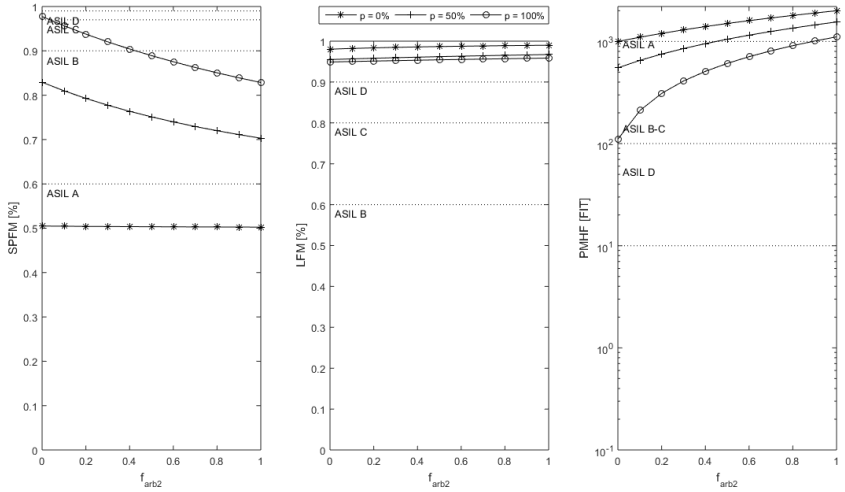


Figure 4.22: Sensitivity of the results to the parameter f_{arb2}

faults increases as well. Based on this, the necessary replication rate can be higher or lower, depending on the ASIL (and therefore depending on the target values of the three metrics).

- The effect of increasing f_{vote2} is completely different (figure 4.33): Since with increasing f_{vote2} the portion of single point faults is increasing too, the SPFM and the PMHF are getting worse. The LFM is almost unaffected, but is slightly increasing. Therefore, the necessary replication rate is higher if the f_{vote2} is rising.
- The sensitivity of the results to the parameter DC_{TMR} is as in case of the DC_{PM}, due to the same reasons (figure 4.34). The effect on the required replication rate is also identical.
- Since Modules 1, 2 and 3 are all equal regarding their failure modes, rates and behaviour, the sensitivity to the parameters $d_{S,M1}$, $d_{S,M2}$ and $d_{S,M3}$ is also identical (figure 4.35). The sensitivity to these parameters is highly dependent on the parameter DC_{TMR} and the replication rate. If the former is high (e.g. 99 %, as assumed in section 4.4.3), the SPFM is almost not affected. With lower DC_{TMR} it is more obvious that the SPFM is increasing with increasing $d_{S,M1}$, $d_{S,M2}$ and $d_{S,M3}$, since the portion of the safe faults is rising. The LFM is not affected, assuming that if the voter detects a deviating module, this failure is reported to the driver (as assumed in section 4.4.3). The PMHF is declining if the $d_{S,M1}$, $d_{S,M2}$ and $d_{S,M3}$ are increasing, since the failure rate potentially

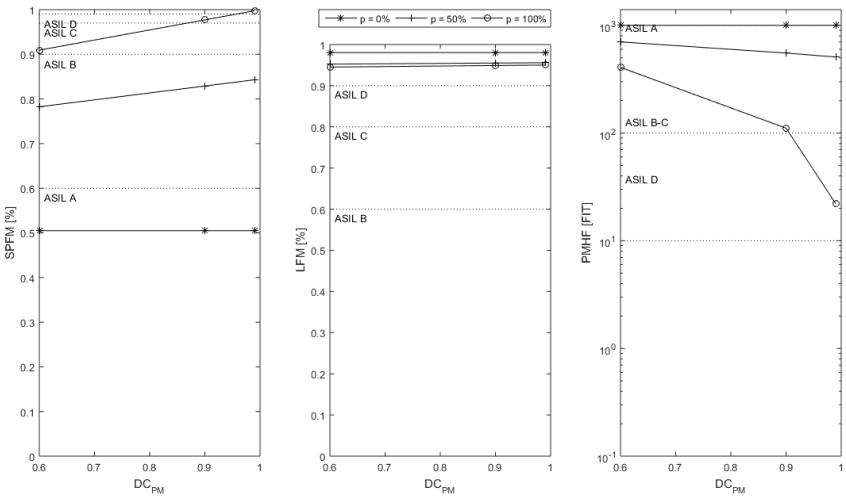


Figure 4.23: Sensitivity of the results to the parameter DC_{PM}

Parameter (↑)	SPFM	LFM	PMHF	p_{nec}
f_{vote1}	↑	↓	↑	↑ / ↓
f_{vote2}	↓	o / ↑	↑	↑
DC_{TMR}	↑	↑	↓	↓
$d_{S,Mx}$	↑	o	↓	↓
$d_{S,CM}$	↑	↑	↓	↓
$d_{S,voter}$	↑	↑	↓	↓

Table 4.2: Sensitivity of the SPFM, LFM, PMHF and the necessary replication rate to the parameters of the TMR architecture

leading to a safety goal violation is decreasing. Since the magnitude of the effect on the SPFM and LFM by changes in the $d_{S,M1}$, $d_{S,M2}$ and $d_{S,M3}$ depends on the overall failure rate of the modules, in case of a higher replication rate, the effect is more significant. This can be clearly seen in figure 4.35. Therefore, with increasing $d_{S,M1}$, $d_{S,M2}$ or $d_{S,M3}$, the necessary replication rate will be lower.

- The sensitivity to the $d_{S,CM}$ is similar, but the magnitudes are completely different (figure 4.36). All three parameters are improving if the $d_{S,CM}$ increases. The improvement is more dramatic at lower replication rates, since the failure rate of the common module is higher in those cases. The improvements of the SPFM and PMHF have the same root cause: The portion of safe faults is increasing. The LFM, on the other hand, is improving at lower replication rates, because at lower $d_{S,CM}$, the $\lambda_{SPF,RF}$ of the common module is higher, leading to a lower LFM (see equation 2.2). Hence, if the $d_{S,CM}$ is higher, the necessary replication rate is lower.
- Since the $d_{S,voter}$ describes the portion of safe faults in the voter itself, it is obvious that all three metrics will improve if the $d_{S,voter}$ increases. The magnitude of this effect depends on the related failure rates of the voter, i.e. on the parameter $f_{vote2} = 1$. If the f_{vote2} is as low as assumed in this section, the effect is negligible. Figure 4.37 shows the case with $f_{vote2} = 1$, to visualise the sensitivity better.

Similarly to Table 4.1, Table 4.2 summarises the results of the sensitivity analysis for the TMR architecture.

Summary - Comparison of the investigated architectures

Based on the qualitative and quantitative results, it can be stated that both investigated fail-operational architectures can cope even with the ASIL D requirements of the [ISO11]. Critical factors are the diagnostic coverage of the diagnostic functions (i.e. the diagnostic modules in the dynamic redundant

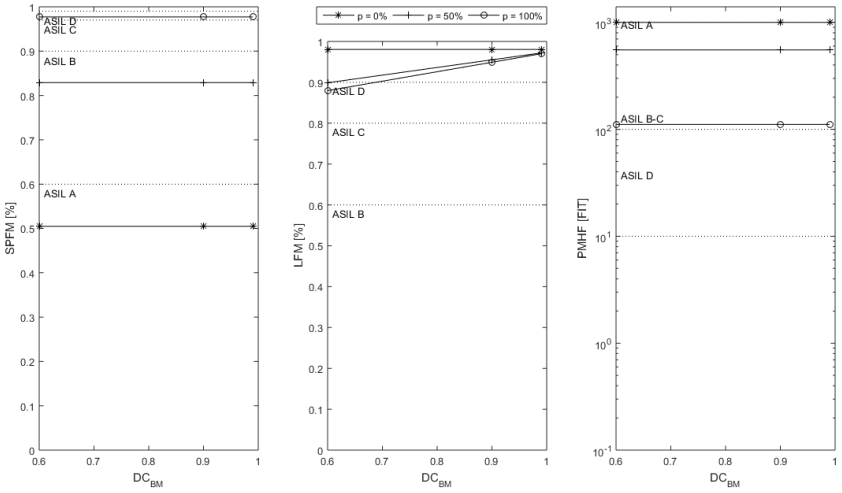


Figure 4.24: Sensitivity of the results to the parameter DC_{BM}

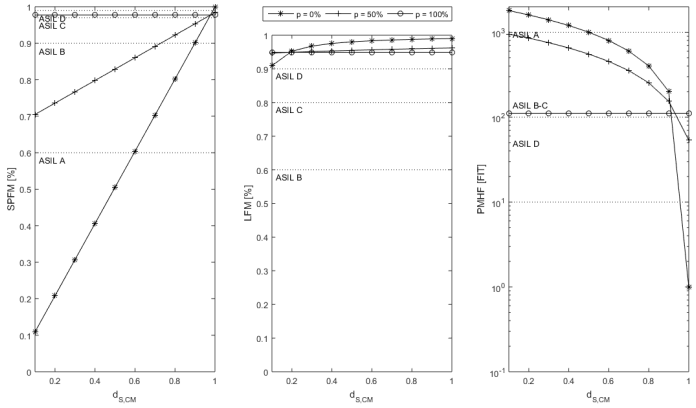


Figure 4.25: Sensitivity of the results to the parameter $d_{S,CM}$ in case of the dynamic redundant architecture

architecture and the voter in the TMR case), the failure rates of SPF sources (i.e. the arbitration logic and the voter) and the appropriate driver warning in case of faults with potential to remain latent.

Table 4.3 shows the quantitative results, valid for both safety goals. As it can be seen, both architectures can reach very similar values for all three metrics. The dynamic redundant architecture can even achieve better results, if the achieved DC is sufficiently high. It is assumed in general that voters tend to have higher DC-s than diagnostic modules. On the other hand, there are various practical examples, where a very high DC (and therefore SPFM) can be reached with a well-founded diagnostic concept. Voters tend to perform better, if the low level HW faults are not well known. The reason for this is that voters compare output values of higher level architectural elements. Hence, failures can be detected by them very effectively, independently of the faults causing them. If this statement is turned around, it can be stated that dynamic redundant architectures perform better, if the low level faults can be identified and detected appropriately. This can be used as input for the topology selection process in the HW design.

For the incompletely redundant architectures it has been shown that the replication rate significantly affects the SPFM and the PMHF but not the LFM. For high ASIL-s (i.e. C and D) practically 100 % replication rate is necessary. For ASIL B, this replication rate can be significantly lower (about 70 %). For ASIL A, assuming that $SPFM = 60\%$ is applied (derived from the [IEC10]), the replication rate can be very low (about 15 %).

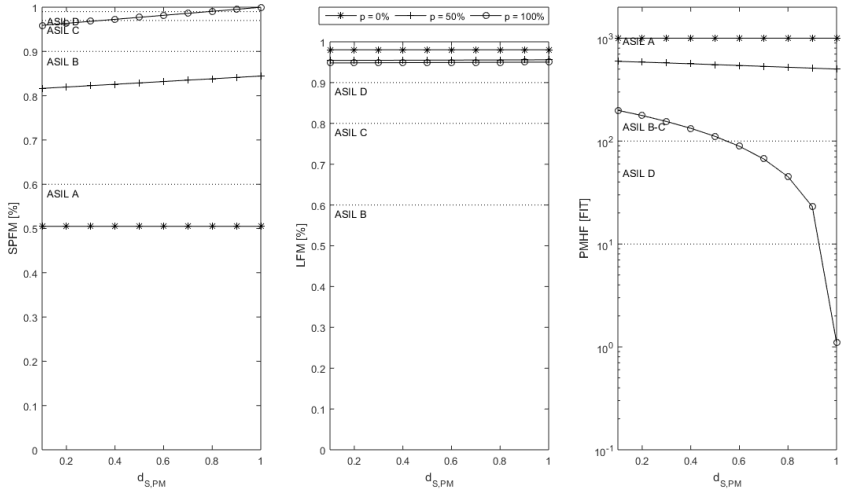


Figure 4.26: Sensitivity of the results to the parameter $d_{S,PM}$

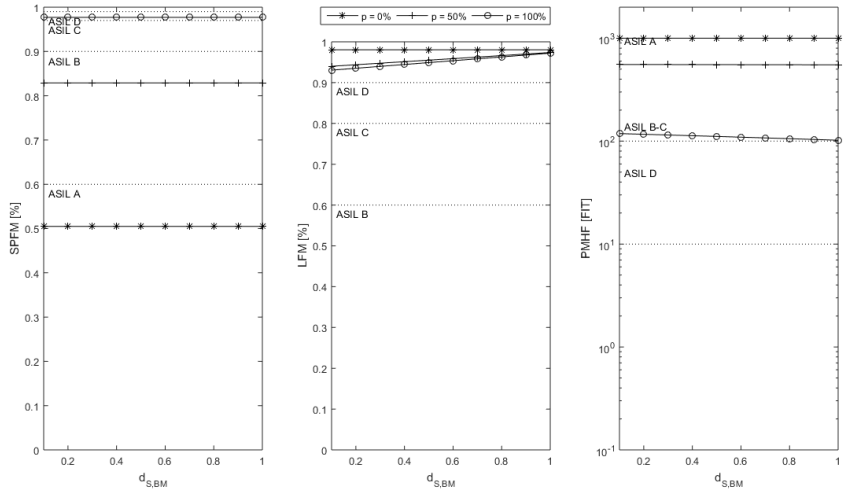


Figure 4.27: Sensitivity of the results to the parameter $d_{S,BM}$

The assumptions made related to the failure rate magnitudes of the architectural elements would probably fit to most real-life applications. But to provide results that are more generic, the sensitivity of the results to the input parameters (e.g. failure rates, diagnostic coverages) is investigated in detail (subsection 4.4.3). It can be stated, that the majority of the parameters has an influence on the three metrics, and the necessary replication rate (for a given ASIL). But nonetheless, the statement still applies that for ASIL A and B the required replication rate can be relatively low for many practical applications, even if deviating from the initial assumptions made by this paper.

Based on the results of this section, it can be stated, that the SPFM and PMHF are two metrics applicable to determine the necessary degree of replication. But, as already mentioned in section 4.3.1, there are no target values for ASIL A. This issue can be addressed using two distinct approaches. The first option is to simply carry over comparable target values from the [IEC10]:

Extension and optimisation proposal: HW fault tolerance targets in the [ISO11]

Option 1: Add requirements to the SPFM, LFM and PMHF metrics for ASIL A, as recommendation:

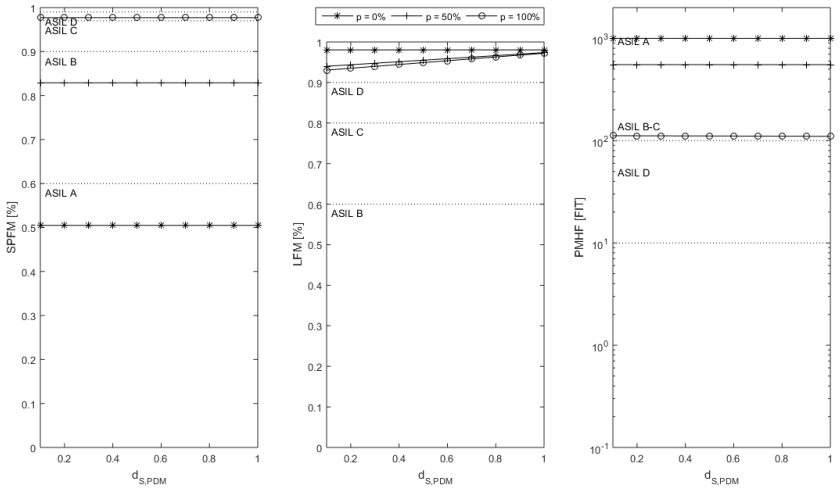


Figure 4.28: Sensitivity of the results to the parameter $d_{S,PDM}$

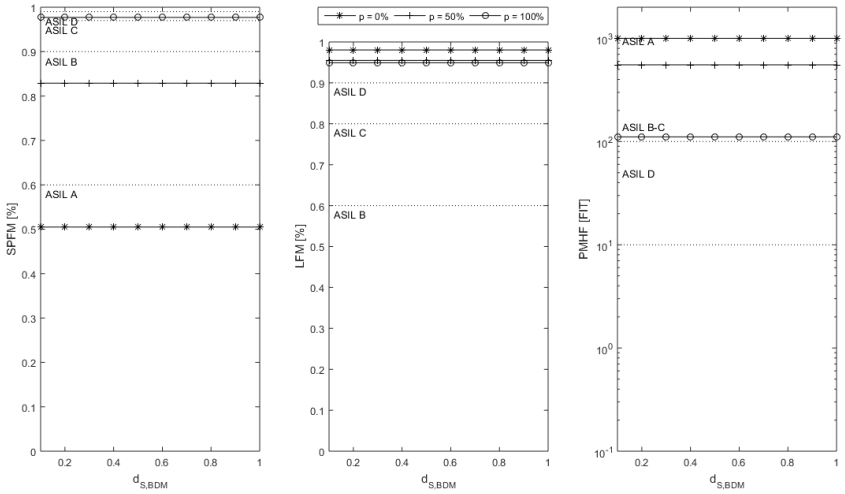


Figure 4.29: Sensitivity of the results to the parameter $d_{S,BDM}$

- [ISO11][part 5, table 4]: ASIL A: SPFM $\geq$ 60%
- [ISO11][part 5, table 6]: ASIL A: PMHF $<$ 1000FIT

The second option is based on the concept of the *replication rate*. Using this rate on a higher component level, its calculation can require significantly less effort, being therefore more appropriate for systems with ASIL A or B fail-operational safety goals.

Extension and optimisation proposal: HW fault tolerance targets in the [ISO11]

Option 2: Extend the standard by a new metric, the Replication Rate Metric (RRM): $RRM \geq X\%$ means that the HW elements contributing to at least $X\%$ of the safety related failure rate (related to the fail-operational safety goal) are replicated. This metric would apply for ASIL A and B, as alternative to the HW architectural metrics in the [ISO11][part 5, clause 8], with the following target values (with regards to the ASIL of the fail-operational safety goal):

- [ISO11][part 5]: ASIL A: $RRM \geq 20\%$

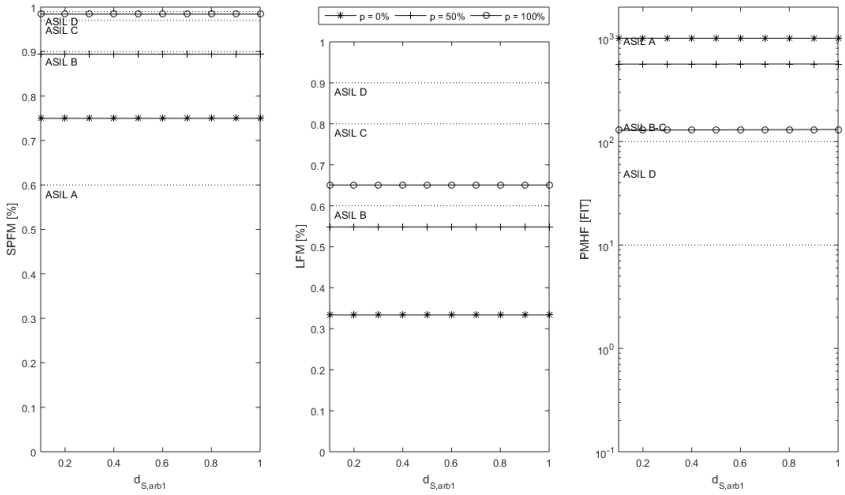


Figure 4.30: Sensitivity of the results to the parameter $d_{S,arb1}$, assuming $f_{arb1} = 1$

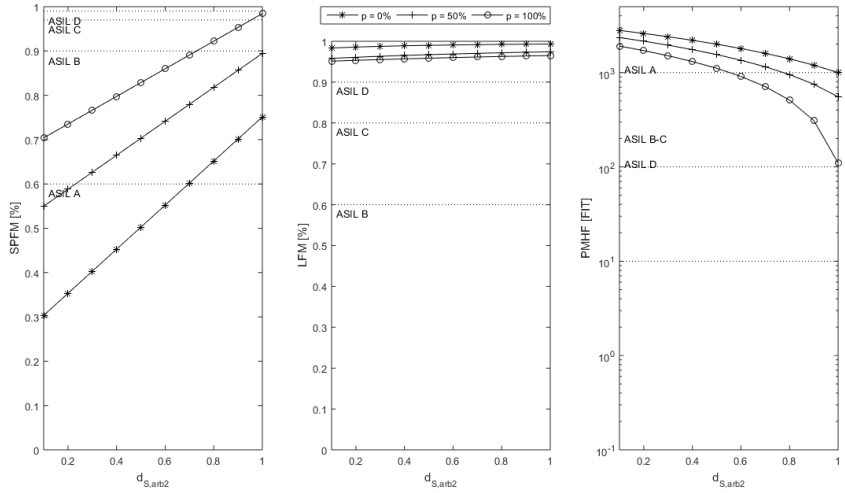


Figure 4.31: Sensitivity of the results to the parameter $d_{S,arb2}$, assuming $f_{arb2} = 1$

- [ISO11][part 5]: ASIL B: $RRM \geq 80\%$

NOTE 1: these values have to be based on a common agreement of domain experts. NOTE 2: for ASIL C and D, the SPFM, LFM and PMHF are more appropriate quantitative targets.

4.4.4 Fail-operational aspects in software development

As already discussed in section 3.1.2, the nature of fail-operational SW is different from fail-operational HW. SW has systematic faults only, which can be prevented by sufficient process rigour to a *certain extent*. As it has been already shown in section 4.3.1, the [ISO11][part 6] is very process-focused for ASIL A and B, and requires the implementation of sophisticated fault detection and reaction measures ony for ASIL C and D. Based on these facts, the following can be stated:

- The applied process rigour and methods can remain the same for fail-operational as for fail-safe SW, since the potential risk related to the former is not higher.
- For ASIL A and B, it shall be clarified whether fault tolerance mechanisms can be replaced by fault prevention measures.

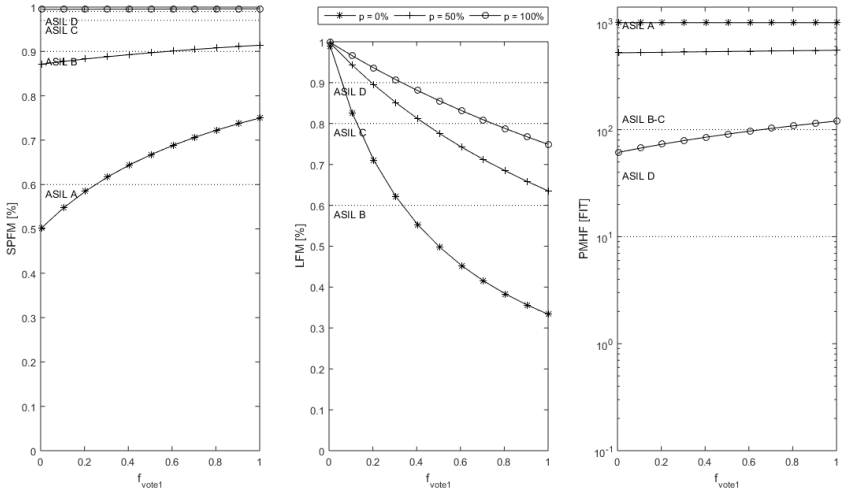


Figure 4.32: Sensitivity of the results to the parameter f_{vote1}

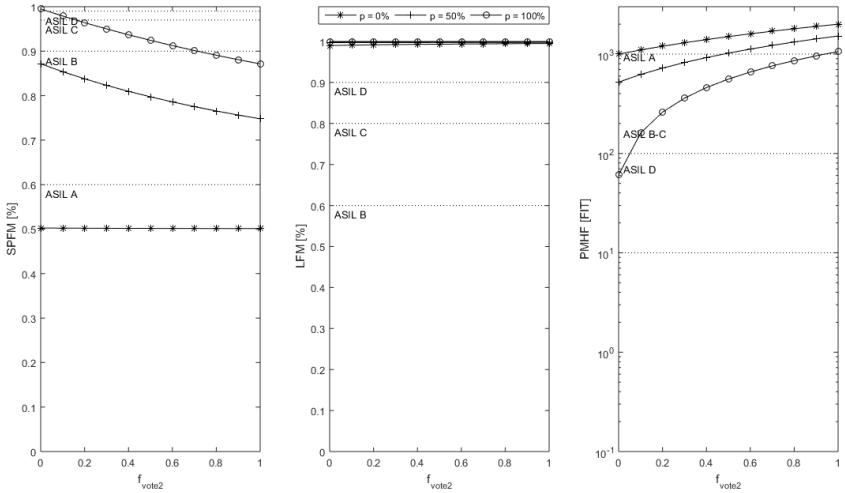


Figure 4.33: Sensitivity of the results to the parameter f_{vote2}

- If SW fault tolerance measures are to be applied, this has to happen on the architectural design level. Unit design and implementation are not affected by them.
- The verification of the unit, integration and embedded SW level can remain unchanged for fail-operational SW.

Therefore, fail-operational aspects in SW development are to be addressed on the upper left side of the V-model [VDI04]:

- Specification of the SW safety requirements
- SW architectural design

Fault tolerance mechanisms in the SW safety requirement specification

The SW safety requirement specification addresses safety related SW-functions, as required by the [ISO11][part 6, clause 6.4.1]. Therefore functions related to fault tolerance shall be addressed in this work product. This leads to the first optimisation proposal for this part of the standard.

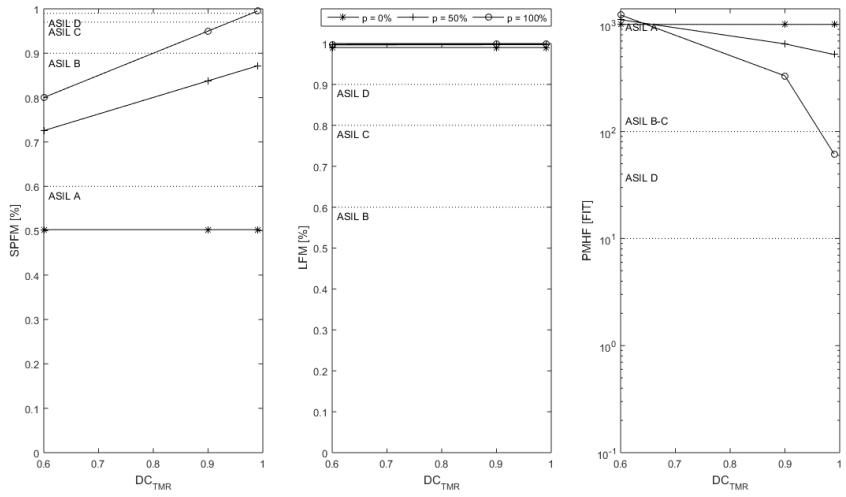


Figure 4.34: Sensitivity of the results to the parameter DC_{TMR}

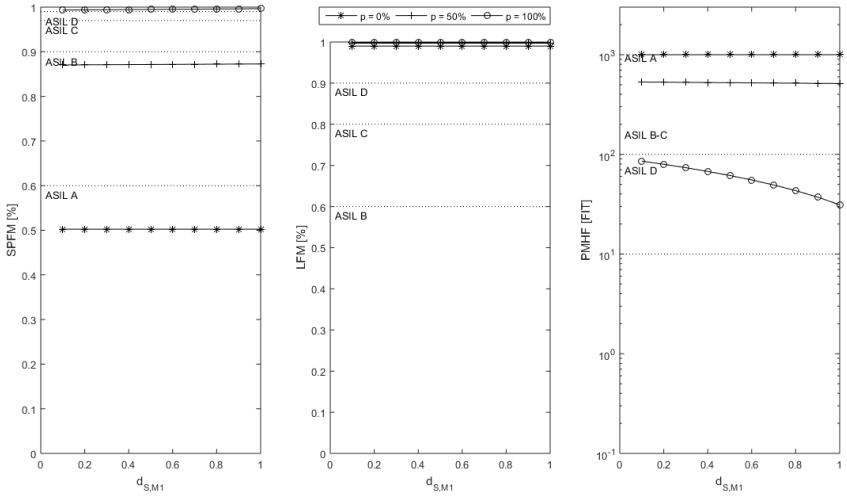


Figure 4.35: Sensitivity of the results to the parameter $d_{S,Mx}$

Extension and optimisation proposal: SW fault tolerance measures in the SW safety requirement specification

Extension to the example in the [ISO11][part 6, clause 6.4.1]: - Functions related to fault tolerance (in case of fail-operational SW)

Fault tolerance mechanisms in the SW architectural design

In order to provide a solid basis for the detailed investigation of the clauses in the [ISO11][part 6, clause 7], there is one core question to be answered: Under which circumstances shall fault tolerance mechanisms be preferred over fault prevention measures in the development. For this, first of all, it has to be understood, *what* fault prevention measures are capable of, and *why* we cannot trust them completely. The main principle of Automotive SPICE is, that using a sufficiently rigorous process is the cornerstone of good SW quality [QMC15]. The [ISO11] extends this approach by defining required methods based on the ASIL of the related SW component. Hence, fault prevention measures in the SW

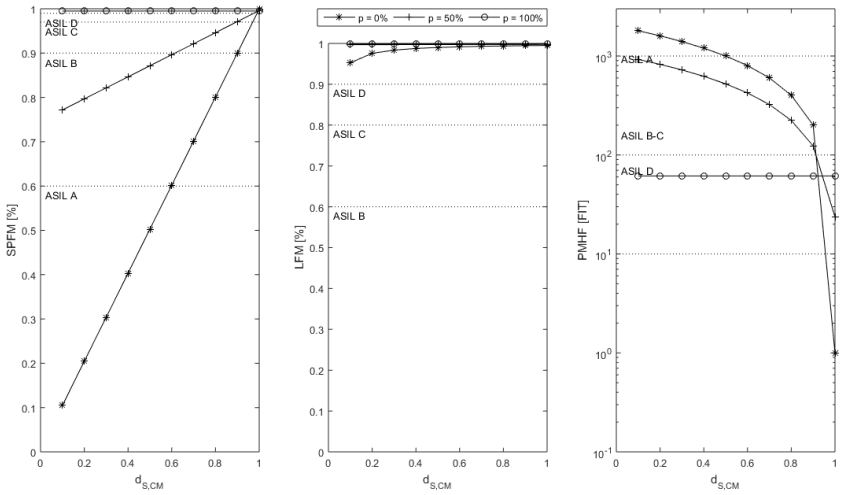


Figure 4.36: Sensitivity of the results to the parameter $d_{S,CM}$ in case of the TMR architecture

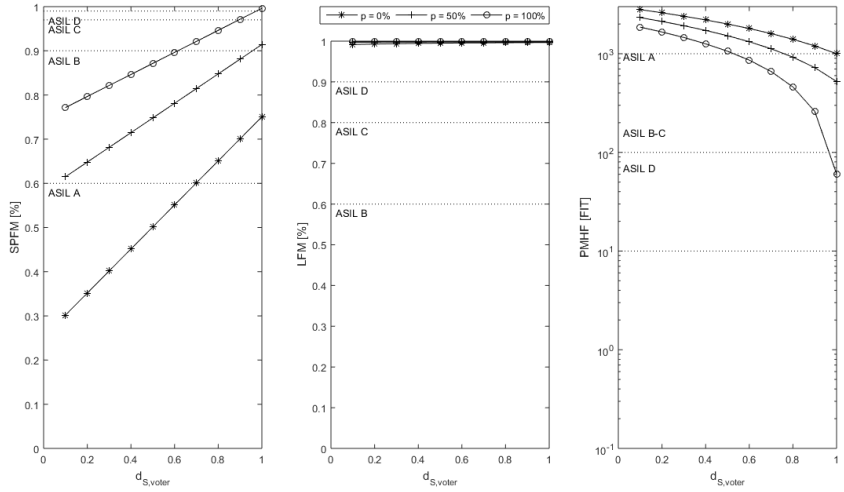


Figure 4.37: Sensitivity of the results to the parameter $d_{S,voter}$, assuming $f_{vote2} = 1$

development include all measures in the V-model that can *contribute* to a fault-free SW. This includes the application of the V-model itself, the utilisation of coding guidelines, unit testing, static verification, etc. Whether these measures succeed or fail depend on various parameters (e.g. complexity).

Based on this rationale, and to be consistent with the [ISO11][part 6, table 4 and table5], table 4.4 and show an extension proposal, to cope with this issue.

As it can be seen, fault tolerance mechanisms are highly recommended for higher ASIL-s. But since both are alternative entries, it is still the designer's responsibility to select the appropriate combination. Beyond that, the designer has to make this decision not globally, for the entire SW, but for each architectural element. This is addressed by the next optimisation proposal:

Extension and optimisation proposal: SW fault prevention versus fault tolerance

Additional requirement that can be integrated in the [ISO11][part 6, clause 6] or [clause 7]: In case of fail-operational SW an appropriate combination of fault prevention measures and fault tolerance mechanisms shall be applied, in accordance with table 4.4.

	Dyn. red. $DC = 90\%$	Dyn. red. $DC = 99\%$	TMR without driver warning	TMR with driver warning
SPFM [%]	97,7	99,8	99,5	99,5
LFM [%]	94,9	97	50,2	99,8
PMHF [FIT]	111	22	61	61

Table 4.3: Results of the quantitative analysis of two typical fail-operational architectures

Extension and optimisation proposal: Fault prevention versus fault tolerance table

Measures		ASIL A	ASIL B	ASIL C	ASIL D
1a	Fault prevention measures	++	++	+	+
1b	Fault tolerance mechanisms	+	+	++	++

Table 4.4: Required measures related to fail-operational SW elements with high complexity

NOTE: Low complexity can be applied as rationale to rely on fault prevention measures.

Since table 4.4 can be applied on various levels of the SW architecture, it can be assumed that most designers would develop heterogeneous architectures, using an appropriate combination of fault prevention measures and fault tolerance mechanisms. Therefore, it is very important that these decisions are included in the SW architectural design, along with the related rationale.

Extension and optimisation proposal: Including decisions related to SW fault prevention and fault tolerance in the architectural design

Additional requirements that can be integrated in the [ISO11][part 6] between the clauses [7.4.8] and [7.4.9]:

In case of fail-operational SW for every safety-related software component, it shall be defined if fault tolerance mechanisms or fault prevention measures are applied, and a rationale shall be given.

Also the extension of the notes of the [ISO11] may be beneficial, to address static and dynamic architectural design aspects specific to fail-operational SW:

Extension and optimisation proposal: Static and dynamic architectural design aspects of fail-operation SW

Extension of the [ISO11][part 6, clause 7.4.5, NOTE 1]:
– Applied measures to ensure the availability of the SW components, if applicable.
Extension of the [ISO11][part 6, clause 7.4.5, NOTE 2]:
– Reconfiguration of the SW in case of a fault, if applicable.

The next part of the [ISO11] that needs to be investigated, are the tables 4 and 5. These tables define error detection and error reaction mechanisms, respectively, on the SW architecture level. The application of these mechanisms is challenging even for fail-safe SW, therefore great care shall be taken when applying them to fail-operational products.

As said before, table 4 requires certain error detection mechanisms depending on the ASIL:

- *Range checks of input and output data* can detect only severe SW failures, and if the input and output variables do not use the entire available range (with regards to their data type). Their effectiveness

is questionable, since in-range failures cannot be detected by them at all. Another issue with this requirement is that it is unclear on which level these range checks shall be applied. Despite these open questions, this type of mechanism is applicable for fail-operational SW. The question is rather if fault detection shall be applied for lower ASIL-s as well.
- The term *plausibility check* can mean various types of detection mechanisms, since the term *plausibility* is very broad. The related note suggests that this entry means a more complex (i.e. model based) check of the output of a SW unit. Plausibility checks can be applied for fail-operational SW as well. It is highly recommended only for ASIL D.
- *Detection of data errors* is a mechanism focusing rather on random HW faults affecting data storage. But since SW functions rely on safely stored data, this has to be considered in the SW architecture design. Error detection mechanisms can be used extensively in fail-operational systems, preferably in combination with error correction features. This mechanism is recommended for all ASIL-s.
- *External monitoring facility* provides sufficiently independent monitoring for the SW and the microcontroller executing it. It's advantage is its design and spatial independence from the main microcontroller. Since this is a very sophisticated mechanism, it is highly recommended only for ASIL D. It is perfectly applicable for fail-operational SW, but it is of paramount importance to define an appropriate reaction mechanism. Since this external monitoring facility cannot differentiate between SW and HW related faults, the reaction mechanism shall be supported by independent HW. This implies a multi-version redundancy supported by two microcontrollers.
- *Control flow monitoring* is intended to detect if the program execution sequence is corrupted (i.e. if SW units are executed in the wrong order or are not executed at all). These control flow violations can occur due to three types of reasons: architectural faults, unit level faults and HW faults. *Architecture level faults* include scheduling and control flow design faults. Potential *unit level causes* include longer execution time, unconditional jumps and hidden control flow within the unit. Control flow monitoring can be used for fail-operational SW as well, but the reaction mechanisms shall be paid attention to.
- *Diverse programming* applies a separate monitoring mechanism based on a diverse implementation of the Software Component (SWC) in focus. As it is noted in the [ISO11], no N-version programming is meant here. For example a functional path can be monitored by a parallel set of SWC-s. This measure can be applied perfectly for fail-operational architectures.

The applicability of [ISO11][part 6, table 4] can be summed up as follows:

- All mechanisms listed in the table can be applied for fail-operational SW as well.
- The reaction mechanisms shall be tailored to fulfil the needs of fail-operational products.
- For ASIL A and B, only range checks are required. The effectiveness of those is very limited. Therefore SW safety at these ASIL-s is almost completely based on the sufficient process rigour. This complies with the proposed table 4.4.
- For ASIL C, control flow monitoring is required to cover timing and scheduling issues, focusing therefore mainly on architecture related faults in the SW.
- Diverse design and plausibility checks, required for ASIL D only are highly sophisticated detection mechanisms, with a high detection probability.

Based on that it can be stated, that [ISO11][part 6, table 4] can be applied to fail-operational SW without modifications.

When it comes to fault tolerance mechanisms in the SW, the detection part is far more simple than the reaction. If a fault is detected in the SW, simply repeating the *same function* with the *same inputs* will lead to the same erroneous outputs. In case of fail-safe systems (which the [ISO11] was mainly meant for), the reaction can be plain and simple: trigger the transition into the safe state by the SW. Due to this reason, it is very important to investigate the mechanisms required by the [ISO11][part 6, table 5.]:

- *Static recovery mechanisms* are one of the most common single-version techniques (see section 3.1.2). The curiosity of this recommendation (this mechanism is only recommended for all ASIL-s) is that it is not necessary for fail-safe systems (as mentioned above) but could support a fault tolerant SW architecture.
- *Graceful degradation* denotes a topology, where in case of a fault, less important functions can be deactivated. This can be applied only for fail-safe functions, due to obvious reasons. Graceful

Extension and optimisation proposal: Mechanisms for error handling at the SW architectural level for fail-safe SW

Methods		A	B	C	D
1a	Fail-safe transition triggered by software	++	++	++	++
1b	Graceful degradation	o	o	+	+

Table 4.5: Mechanisms for error handling at the software architectural level for fail-safe SW

Extension and optimisation proposal: Mechanisms for error handling at the SW architectural level for fail-operational SW

Methods		A	B	C	D
1a	Single version fault tolerance techniques	+	+	+	+
1b	Multi-version fault tolerance techniques	+	+	++	++
2	Correcting codes for data	o	+	+	++

Table 4.6: Mechanisms for error handling at the software architectural level for fail-operational SW

NOTE 1 Single version techniques include fault containment, fault recovery, exception handling, checkpoint and restart and process pairs.
NOTE 2 Multi-version techniques include recovery blocks, N-version programming, and N-self-checking programming.

degradation can be still beneficial in fail-operational products, though. By deactivating functions, that do not have to remain operational even in the presence of a fault, can still be very valuable for the fail-operational portion of the same SW. For example, if a fail-safe function leads to control-flow violations, its deactivation can help the rest of the SW to continue to operate. Graceful degradation has one significant draw-back: It increases the complexity of the SW, if the functions are not independent. In that case, some functions cannot be deactivated without deactivating other functions or changing their parameters. Graceful degradation is highly recommended for ASIL C and D.

- *Independent parallel* redundancy is one of the most common multi-version fault tolerance techniques (see section 3.1.2), and is not necessary to be applied for fail-safe SW. This mechanism is highly recommended for ASIL D only.
- *Correction codes for data* are the appropriate error handling mechanisms for data errors. Due to obvious reasons, this is only necessary for fail-operational systems. This technique is often used as one combined measure with error detection codes. Error detection and correction codes can cover SW faults as well, but are more often utilised to deal with random HW faults affecting the volatile or non-volatile memory. Error correction codes are only recommended for all ASIL-s.

The applicability of [ISO11][part 6, table 5] can be concluded as follows:

- The table contains an arbitrary list of error handling mechanisms applicable either for fail-safe or fail-operational SW or both. It is therefore both too specific and incomplete, making it difficult to apply in the practice.
- Therefore the ASIL should be less decisive, but the applicability and the appropriateness of these mechanisms for the particular application.

Based on these results, the applicability of the [ISO11][part 6, table 5] may be improved by introducing tables 4.5 and 4.6.
The rationale behind this improvement proposal can be concluded in the following points:

- Split up the tables for fail-safe and fail-operational SW. This allows to set the degrees of recommendation properly.

- Add Fail-safe transition as basic measure for fail-safe systems, in accordance with the current state-of-the-art.
- Eliminate specific fault tolerance measures and replace them with two generic entries, to allow the designers the free choice for their particular applications.
- Change the degree of recommendation of multi-version techniques to equal or higher than that of single-version techniques. The reason for this is that multi-version techniques have better fault tolerance capabilities due to the higher degree of diversity.
- Change correction codes for data from alternative entry to exclusive entry, since its application is independent from the fault tolerance techniques of the SW itself.
- Change the degree of recommendation for correction codes for data to highly recommended in case of ASIL D.
- Exclude graceful degradation from the fail-operational table, since it is included in the single version techniques.

Comparison of fault tolerant SW topologies [Fab16]

Similarly to the HW in section 4.4.3, in this chapter typical fault tolerant SW architectures are compared from the viewpoint of an automotive application. In order to do so, a state-of-the-art SW architecture comparison method is applied. [Sum15] gives a very thorough and up-to-date overview of the SW architecture evaluation methods currently available. This paper introduces methods like the Scenario-based Architecture Analysis Method (SAAM), Architecture Trade-Off Analysis Method (ATAM), Architecture Level Modifiability Analysis (ALMA), Scenario-based Architecture Analysis Method for Complex Scenarios (SAAMCS), Software-Based Architecture Re-engineering (SBAR), Architectural Level Prediction of Software Maintenance (ALPSM), Extending SAAM by Integration in the Domain (ESAAMI) and the Software Architecture Comparison Method (SACAM). Selecting the right method is the first task to solve in this section. Since the architectures to be compared are defined only on a very high level and focus on one single SWC with its in- and outputs, the methods that require information about the detailed design and implementation of the SW are not applicable: SBAR, ESAAMI, ATAM. Some methods on the other hand, have only a very limited scope, and are therefore not applicable: ALMA (focuses on modifiability), SAAMCS (focuses on modifiability and feasibility) and ALPSM (focuses on maintainability). This leaves two options: the SAAM and the SACAM. The method chosen for this section is the *SACAM*, due to the following reasons: It is meant to compare architectures, its process is described more in detail in the literature, and it does not consider indirect scenarios (i.e. tasks the initial architecture cannot cope with).

[Ver03] introduces the SACAM in detail by explaining its steps, concepts, methods and by giving a comprehensible example. The steps of this analysis method are the following:

1. Preparation: Identify the business goals and investigate available documentation.
2. Criteria collation: Derive comparison criteria from the goals identified in the first step and define quality attribute scenarios based on them.
3. Determination of extraction directives: Determine which architecture views and patterns are related to the scenarios defined above.
4. View and indicator extraction: Extract architectural views for each architecture candidate and search for the indicators identified above.
5. Scoring: Score the fitness of each candidate.
6. Summary: Summarize the results and select the best variant.

This section will follow these steps with some adoptions wherever necessary.

Preparation and criteria collation Since the example is relatively simple and has a limited scope, the first two steps will be performed in one. The context of this comparison is a safety-related automotive application that has to show fail-operational behaviour. In the comparison, the following stakeholders will be considered:

- SW project management
- Safety engineer

- SW engineers
- SW test engineers

In the next step, criteria shall be derived for each stakeholder. Various criteria can be found in the literature, including the following:

- Complexity [Ort15]
- Reusability [Sum15] [Yeo02]
- Variability or flexibility [Ver03] [Yeo02]
- Maintainability or modifiability [Sum15] [Tha05] [Ort15] [Yeo02] [Ver03] [Car98]
- Portability (to other platforms) [Yeo02]
- Testability [Sum15]
- Security [Sum15]
- Safety [Ver03]
- Availability or reliability [Sum15] [Car98] [Tha05] [Yeo02]
- Integrity [Yeo02]
- Functionality [Tha05]
- Usability [Tha05]
- Learnability [Tha05]
- Performance [Sum15] [Ver03] [Car98] [Yeo02]
- Response time [Tha05] [Ort15]
- Costs [Tha05]
- Team size [Tha05]
- Development time [Tha05]

For the analysis the following criteria will be applied:

- Development effort (relevant for: SW project management)
- Modifiability (relevant for: SW project management, SW engineer)
- Fault tolerance capability (relevant for: safety engineer)
- Real-time behaviour (in case of a fault) (relevant for: safety engineer)
- Complexity (relevant for: SW engineer, SW test engineer, safety engineer)
- Resource usage (including processing time and memory consumption) (relevant for: SW engineer)
- Testability (relevant for: SW test engineer)

As last point, scenarios need to be derived to support the evaluation of the criteria. Although several architecture analysis methods (e.g. SACAM, ATAM, SAAM) use scenarios to evaluate the previously identified criteria, the term *scenario* is used completely differently in the related literature [Cle96] [Car98] [Ver03] [Kru95]. Common in all referred sources is that scenarios are always derived from the criteria to support their evaluation. [Kru95], [Car98] and [Cle96] use use cases (e.g. ”the client receives periodic updates at the specified rate”) and events (e.g. ”a server suffers failure and is rebooted”) as scenarios. [Ver03] defines scenarios as functional and non-functional requirements (e.g. ”The sunroof must be able to accept and execute user commands 250 ms after power-on.”). For the purpose of this thesis an appropriate combination of these two approaches is applied, leading to the following scenarios:

- Development effort: The SW shall be developed with a given team within a given time frame. The evaluation focuses on the development effort increase related to the base architecture. This scenario is based on the use-case approach.
- Modifiability: The function of the SWC in focus (SWC B in figure 4.38) is changed due to a change in the overlying requirements. It is assumed that the input and output vectors of the SWC B are not changed; only the transfer function between them. This scenario is based on the use-case approach.
- Fault tolerance capability: The SW shall remain functional even if the SWC in focus fails. This scenario is based on the functional requirement approach.
- Real-time behaviour: The SW shall re-establish full functionality within a given time-frame if the SWC in focus fails. This scenario is based on the non-functional requirement approach.
- Complexity: The complexity shall be minimized to enhance the comprehensibility and testability of the SWC in focus. This scenario is based on the non-functional requirement approach.

- Performance: The processing time and memory consumption of the SW shall be minimized. Evaluation focuses on the processing time and memory consumption increase related to the base architecture. The processing time assumes sequential processing on a single microcontroller core. This allows the re-use of the results of this paper for lower ASIL-s, where no multi-core or multi-controller architectures are utilised. This scenario is based on the non-functional requirement approach.
- Testability: The SW is tested on the unit and integrated SW level. This scenario is based on the use-case approach.

Determination of extraction directives This step is intended to identify which architectural views are necessary to obtain the required information and to identify patterns and metrics that shall be looked at during the evaluation. In this section, the 4+1 approach of [Kru95] is applied. [Kru95] already gives very detailed guidance on how to use this method in the practice but [Yeo02] provides an even more state-of-the-art interpretation using Unified Modelling Language (UML) notations to support the evaluation of pre-defined criteria. Based on these two papers the following approach is used for the 4+1 views of [Kru95]:

1. *Logical:* Static block diagrams (showing the SWC-s with their in- and outputs) and UML state machine diagrams will be applied. The static diagrams are simple block diagrams showing the modules and their interfaces. No UML class or composite diagrams have been applied for the sake of simplicity.
2. *Development:* As already stated by [Kru95], for small systems (like the example in this section), this view can be identical with the logical view. Therefore the development view is omitted.
3. *Process:* UML sequence diagrams showing the interaction time-line between the SWC-s will be applied.
4. *Physical:* As already stated by [Kru95], the physical view is pointless if the entire SW is running on the same microcontroller. Therefore this view will be omitted.
5. *Scenarios:* The scenarios mentioned above are applied. The views defined above support the evaluation of the scenarios. The link between scenarios and views (among other information) is shown in table 4.7.

In the next step, it shall be defined which view is used for the evaluation of which criteria, and on which indicators it shall be looked at. Indicators are design patterns or design properties that support the evaluation of the identified criteria. The architecture analysis methods mentioned at the start of this section use high or low level metrics and indicators for this purpose. The examples investigated here are only representing typical architectural design patterns and lack the level of detail necessary for highly sophisticated metrics. Hence, the evaluation will be based on high level indicators. The evaluation may therefore seem arbitrary but defining quantitative metrics without knowing the detailed design wouldn't be different. The relations between the criteria, views and indicators can be seen in table 4.7.

Note, that the criteria "complexity" and "development effort" are related and, to a certain extent, even overlapping. Both are related to the number of SWCs, obviously. But whilst development effort is strongly related to the functional extent of the individual SWCs, and their ASIL-s, these indicators are loosely connected to complexity. The latter (i.e. complexity), on the other hand is significantly affected by the number of interfaces, interactions and state transitions, making the SWCs more difficult to comprehend and test.

View and indicator extraction This step is the actual analysis step: based on the rules defined in the previous step, the architectural variants are evaluated. These architecture variants are the following:

1. As starting point (figure 4.38), a very simple SW architecture is chosen. It was not the intention to analyse a complete SW architecture, since an automotive SW architecture may consist of dozens of SWC-s. The starting SWC architecture is modelling practically only the SWC B with its inputs and outputs. The SWC-s A and C only represent all SWC-s producing the inputs and using the outputs of SWC B. The principle behind this simplification is that as already mentioned above, the SW architecture designer needs to define for each safety related SWC if fault prevention or fault

Criteria	Views	Indicators
Development effort	Block diagrams	Number of SWC-s Functional extent of the SWC-s ASIL of the SWC-s
Modifiability	Block diagrams Sequence diagrams	Number of SWC-s affected by the change Number of interactions affected by the change
Fault tolerance capability	Block diagrams	Number and type of faults that can be tolerated
Real-time behaviour	Sequence diagrams State machine diagrams	Number and duration of state transitions and interactions following the failure of SWC B_x
Complexity	Block diagrams Sequence diagrams State machine diagrams	Number of SWC-s Number of interfaces Number of interactions Number of state transitions
Performance	Block diagrams	Number of SWC-s Functional extent of the SWC-s
Testability	Block diagrams	Test stipulation possibilities on the interfaces

Table 4.7: SACAM - Relations between criteria, views and indicators

tolerance is applied. Therefore, this comparison example represents a typical decision-making scenario for a given SWC.

2. generic single-version architecture (figure 4.41),
3. N-version programming (figure 4.44),
4. N-self-checking programming (figure 4.47),
5. gecovery blocks (figure 4.50).

First, the four architectural alternatives are introduced. The starting point is a simple architecture shown in figure 4.38. The model consists of three SWC-s. SWC B is the one in focus, getting its inputs from SWC A and providing its outputs to SWC C. Both the inputs and the outputs are represented by a vector: $\vec{i}$ and $\vec{o}$, respectively. Since the extension proposal in table 4.4 requires fault tolerance mechanisms for ASIL D only, it can be assumed that SWC B is rated ASIL D.

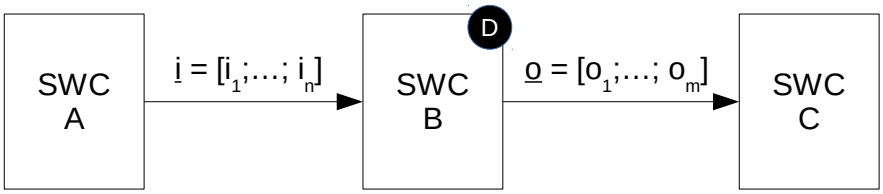


Figure 4.38: Block diagram of the starting software component model

The sequence diagram shown in figure 4.39 shows the interaction between these three SWC-s and the sequential order they occur. SWC A provides the input vector $\vec{i}$ to SWC B, which then calculates its output vector $\vec{o}$ and provides it to SWC C, in this order.

As shown in the related state diagram (figure 4.40), the state transitions of this architecture are very simple. As long as the SWC B is working fine, the SW is in the normal state. If SWC B fails, the SW moves into the failure state where no function is provided. Normal state can be recovered if the SWC B heals. The obvious fact that this architecture cannot tolerate any SW failures is reflected by this diagram.

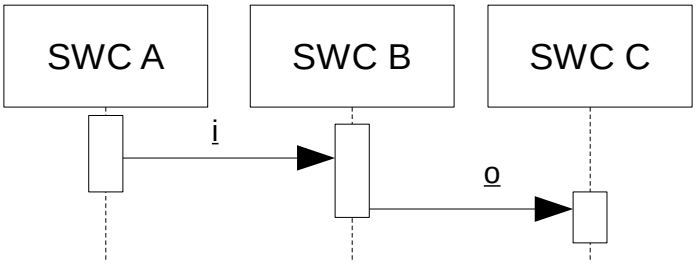


Figure 4.39: Sequence diagram of the starting software component model

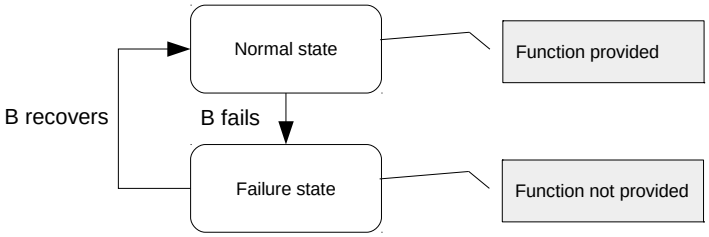


Figure 4.40: State machine diagram of the starting software component model

The first alternative fault tolerant topology is a *single-version* solution, shown in figure 4.41. As already discussed in section 3.1.2, this architecture is based on a combination of fault detection, fault containment and fault recovery. In order to support these steps, the original architecture has been extended by the following SWC-s:

- SWC D is a diagnostic component, detecting safety critical faults of SWC B. Faults detected by this SWC are reported to other SWC-s by an error flag.
- SWC I is an interface component cutting the output of SWC B from SWC C for the time of the reconfiguration, if a fault is reported by SWC D.
- SWC CpM handles the checkpoint memory. If SWC D reports a fault, this component reloads the previously stored content of the checkpoint memory into SWC B.
- SWP denotes the partitioning measures, ensuring interference freeness of SWC B from other SWC-s. This freedom from interference can be based on memory partitioning, control flow monitoring and the decoupling of the interfaces.

The dynamic behaviour of this architecture is shown in figure 4.42. Although it is "only" a single version solution, the interactions between the SWC-s are complicated. The diagram is split into two parts: The upper part shows the sequence if no failure occurs, whilst the lower one represents the interactions if SWC B fails.

As long as SWC B is working as specified, the sequence is not much different from the one of the starting architecture. The only difference is that $\vec{o}$ is provided to SWC D as well. SWC D checks the output vector, as described above.

If a failure occurs, SWC D detects and reports this to the SWC I first, to cut the interaction between SWC B and C. After that the SWC CpM is notified by the error flag (ef), and the checkpoint memory is loaded into the Random Access Memory (RAM) (symbolised here by SWC B, as its RAM area), to recover the last failure-less state. Then the output vector $\vec{o}$ is provided again to SWC I and further to C. If the fault was only a transient, this measure is sufficient. If necessary, SWC CpM can be extended with a function slightly modifying the content of the checkpoint memory, to avoid the same failure happening again. The same goal can be reached by enhancing SWC B with a function modifying its own input vector $\vec{i}$.

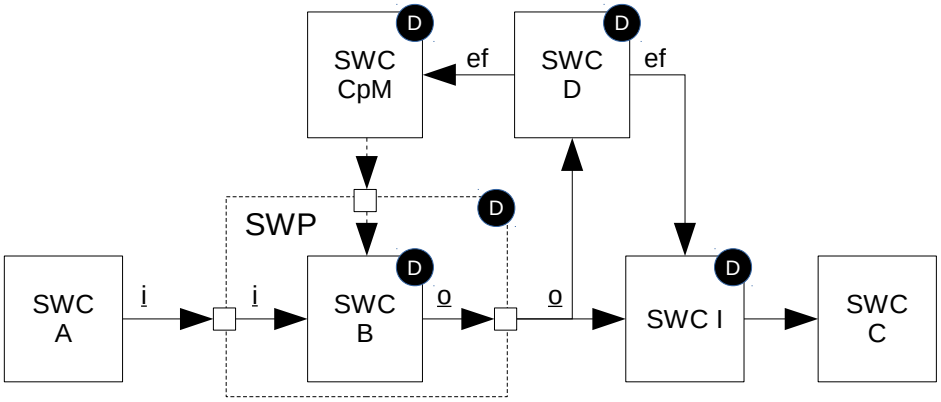


Figure 4.41: Block diagram of the extended software model - single version fault topology

As shown in the related state machine diagram (see figure 4.43), the state transitions of the single version topology are very straightforward. The SW is in a normal state until SWC B fails. Then for a short time an intermediate state is entered, where the output of SWC B is isolated from SWC C. Then in the next state, the checkpoint memory is reloaded. This diagram shows the biggest issue with this topology. If SWC B keeps failing over and over again (i.e. has a permanent fault, not just a transient one), the SW will end up circling around in these three states. This leads to a permanent failure of the SW itself.

Multi-version techniques offer one main advantage over single-version ones: diversity. The first multi-version architecture investigated here is based on diversity of N parallel paths: *N-version programming*. The example shown in figure 4.44 uses three parallel paths, since this is the lowest possible number in a voter-based architecture.

As it can be seen in the block diagram in figure 4.44, the original architecture has been extended by three SWC-s:

- SWC-s B_2 and B_3 are diverse implementations of SWC B_1, which is equal to the original SWC B. The level of diversity may vary. The most common level of diversity is to use the same specification but different designs and implementation. In accordance with [Ede15] ASIL decomposition can be applied. Therefore the SWC-s B_x can be rated ASIL B(D).
- SWC V represents a majority voter.

The dynamic behaviour of the N-version programming topology is shown in figure 4.45. All three variants of the SWC B get the inputs from SWC A and provide their outputs (vectors $\vec{o_1}$, $\vec{o_2}$ and $\vec{o_3}$ respectively) to SWC V. The voter generates the final output vector $\vec{o}$ to SWC C. The good thing about voter-based architectures is that the interaction is the same even if B_1, B_2 or B_3 fails.

The related state machine diagram (figure 4.46) shows one of the key advantages of this architecture. Since the voter (i.e. the SWC V) performs the voting of the three input vectors from the SWC-s B_1, B_2 and B_3 all the time, there is no extra action necessary for fault tolerance. The state machine diagram of the starting point architecture is simply extended by one extra state symbolising the state where the failure of one SWC B_x is tolerated. On the other hand, figure 4.46 also shows the major difference of voter-based architectures, compared to diagnostic mechanism based ones. The voter generates its output (to SWC C) based on its three inputs: If at least two "agree", they are considered as correct. Note, that the term "agree" means here, that the voting is not necessarily exact. Depending on the diversity of the three SWC-s B_1, B_2 and B_3, certain deviations may be allowable. It is important to emphasise, that the voter will regard erroneous inputs as correct if they match. This is the key weakness of voter-based fault tolerance.

Another aspect that needs to be mentioned, is the related warning and degradation concept. In case of multi-version techniques, the fault toleration state cannot be used infinitely. A second failure (or

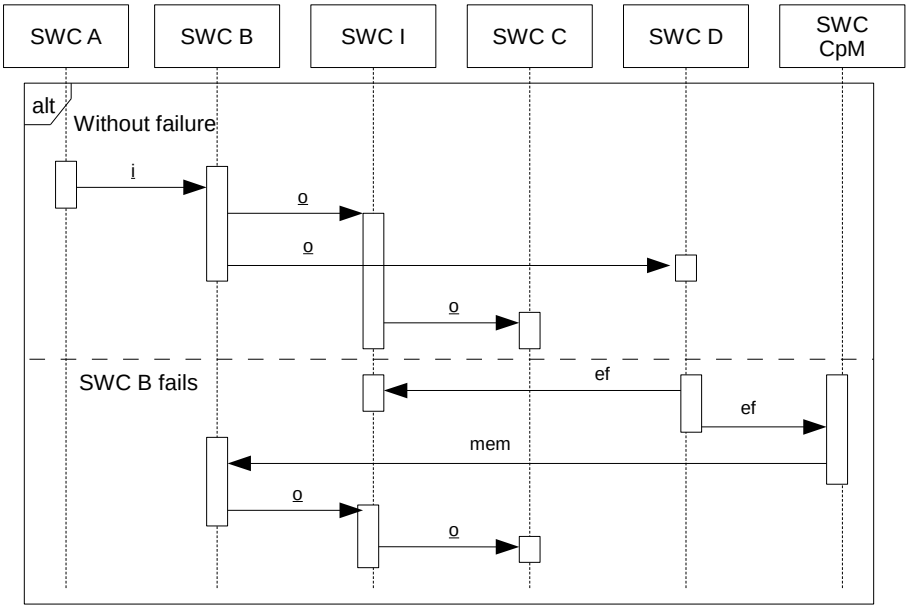


Figure 4.42: Sequence diagram of the extended software model - single version fault topology

deviation) could be safety critical. For this purpose, the software or the driver may take actions in fault toleration state. Graceful degradation of the function implemented by SWC B_1 may be an example for the former, the driver seeking assistance at a repair shop for the latter. This part of the concept is not investigated in detail by this paper, for the sake of simplicity.

A different approach is used by the *N-self checking programming* shown in figure 4.47. In this case, the starting topology is extended by the following SWC-s:

- SWC B_2 is a diverse implementation of the original SWC B (here SWC B_1).
- SWC S is a switch, selecting one of the output vectors of SWC B_1 or B_2. As long as no failure is reported by the SWC D_1, the output of B_1 is selected by default. If an error is reported by the error flag output (denoted with ef in figure 4.47), the output of B_2 is routed through. Since after a failure, B_2 takes over alone, no ASIL decomposition, like in case of the N-version programming, can be applied.
- SWC SWC D_1 and D_2 implement failure detection functions, like the SWC D in case of the single version architecture. The role of SWC D_2 is to detect latent faults in the software, supporting an appropriate driver warning and degradation concept, as mentioned above.

As in case of the single version topology, the interactions vary depending on whether a failure is present (figure 4.48). The input vector $\vec{i}$ is provided to both the SWC B_1 and B_2, independently if a failure is present. Both these modules provide their output vectors $\vec{o_x}$ to their respective diagnostic modules (SWC D_1 and SWC D_2). This allows the healing of the system, by detecting if B_1 or B_2 recover.

If B_1 fails, SWC D_1 detects that and sends an error flag to SWC S. This SWC then routes the output vector $\vec{o_2}$ of SWC B_2 through to SWC C, as long as B_2 is working properly. If B_2 fails, SWC D_2 detects that and sends an error flag to SWC S. The reaction on this depends on if B_1 is already in a failed state. The output $\vec{o}$ of SWC S may be affected if B_1 has already failed. In that case, the software may enter a failure state, where the output is set to a pre-defined value.

The related state machine diagram (see figure figure 4.49) shows that the states and transitions are very similar to those of the N-version programming. If one of the SWC-s B_x fail, the function can still be provided. If the second instance also fails, the software has to enter a failure state.

Figure 4.50 shows an even more complex topology, the *recovery block technique*. This technique can be regarded as a combination of the single-version technique and the N-version programming presented

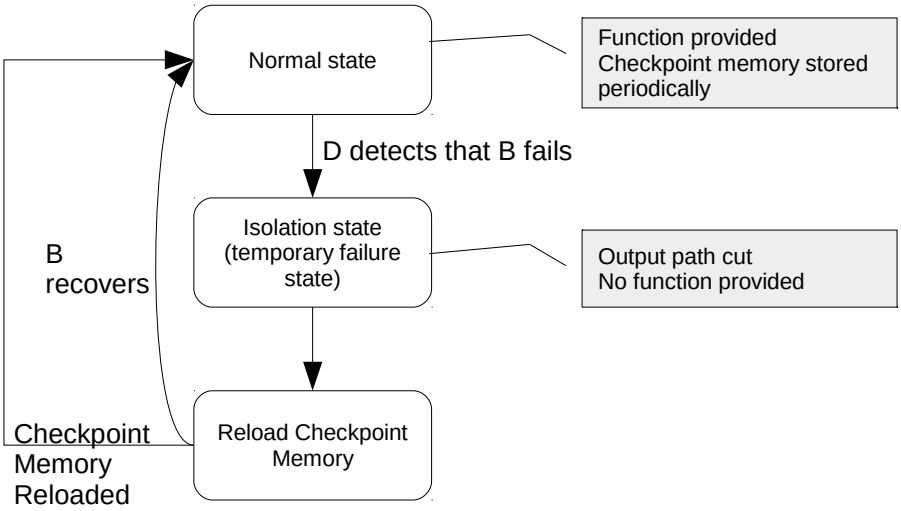


Figure 4.43: State machine diagram of the extended software model - single version fault topology

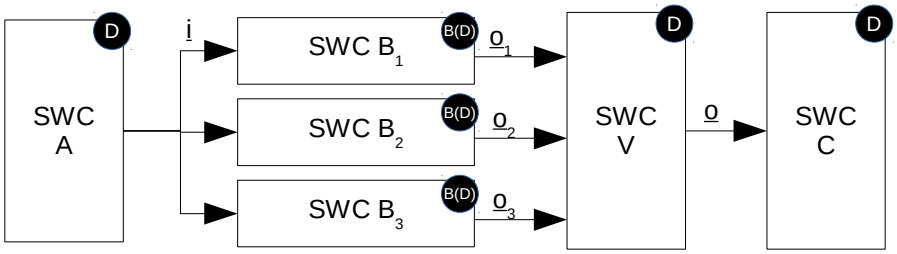


Figure 4.44: Block diagram of the extended software model - N-version programming

above. Several additional SWC-s have been introduced. The SWC-s B_2, S, and D have the same function as in case of the N-self checking topology. The SWC CpM has the same function as the SWC of the same name in the single-version architecture. Basically, the recovery block topology enhances the N-self checking layout with the checkpoint memory. If the SWC B_1 fails, it is detected and reported by the SWC D to the SWC CpM. Then the checkpoint memory is loaded into SWC B_2 and SWC S routes its output through, as can be seen in the figures 4.51 and 4.52.

Scoring In this step the architecture alternatives can be rated, based on the evaluation criteria and indicators defined in table 4.7. For rating and scoring the following scheme will be applied:

- +2: very good
- +1: good
- 0: fair
- -1: poor
- -2: very poor

For each criteria, this rating may be therefore also considered as ranking, showing how the three fault-tolerant variants and the starting point architecture relate to each other.

The evaluation of the five architectural variants in accordance with the previously defined criteria is as follows (see table 4.8):

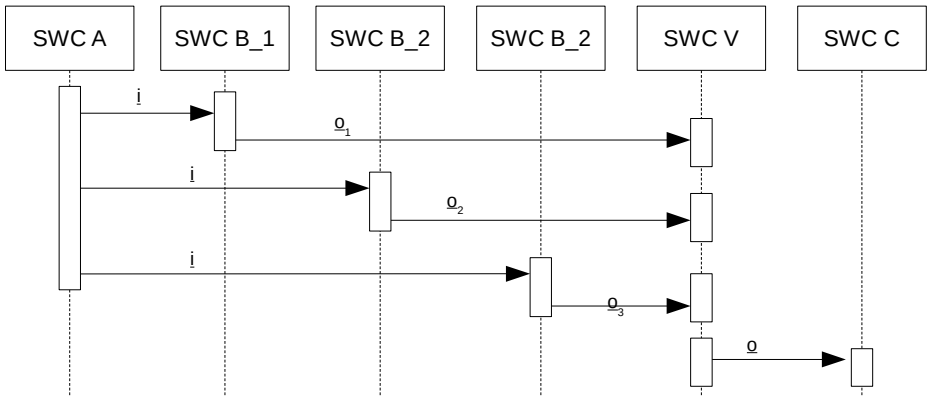


Figure 4.45: Sequence diagram of the extended software model - N-version programming

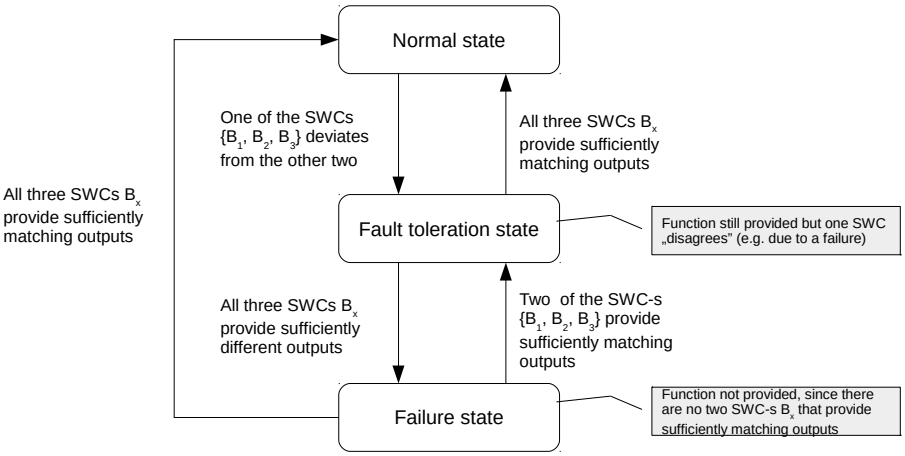


Figure 4.46: State machine diagram of the extended software model - N-version programming

- *Development effort:* As stated by table 4.7, the indicators for the estimation of the development effort are the number, the functional extent and the ASIL of SWC-s. Without hard evidence, the functional extent is very difficult to estimate. Due to the lack of detailed knowledge about the exact functionalities and the implementation, the evaluation of this criteria is challenging, and can be based only on experience based expert judgement. The single version architecture involves the same amount of additional SWC-s as the N-version programming variant. The N-self-checking- and the Recovery block topologies require one SWC more than the other two. The functional extent of the SWC CpM in case of the single version technique and the recovery block topology may vary strongly, depending on how many states and variables need to be stored and restored in case of a failure. The SWC I of the same variant is considered as fairly simple, since it only has to decouple the output temporarily in case of a failure. In case of the multi-version techniques, the main contributors for the development effort are the SWC-s B_x and D_x. The voter and the switch are considered significantly simpler. In case of a given functionality of the SWC B of the starting point architecture, the functional extent of the SWC-s B_x and D_x is comparable to that of the original SWC B. Therefore, the multi-version techniques are considered being worse in terms of development effort, than their single-version counterpart. The N-version programming has two main advantages over the N-self-checking variants in terms of development effort: ASIL decomposition is possible, and it involves one SWC less. The development effort of the N-self-

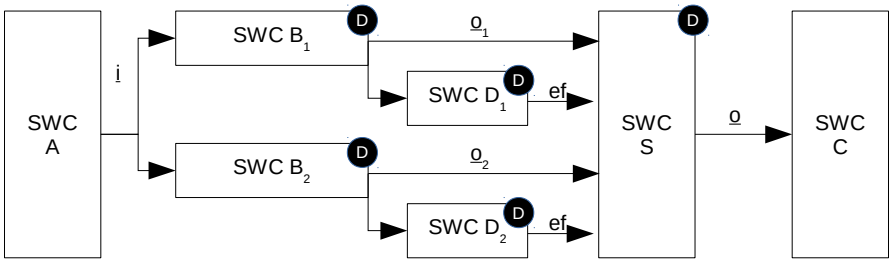


Figure 4.47: Block diagram of the extended software model - N-self checking programming

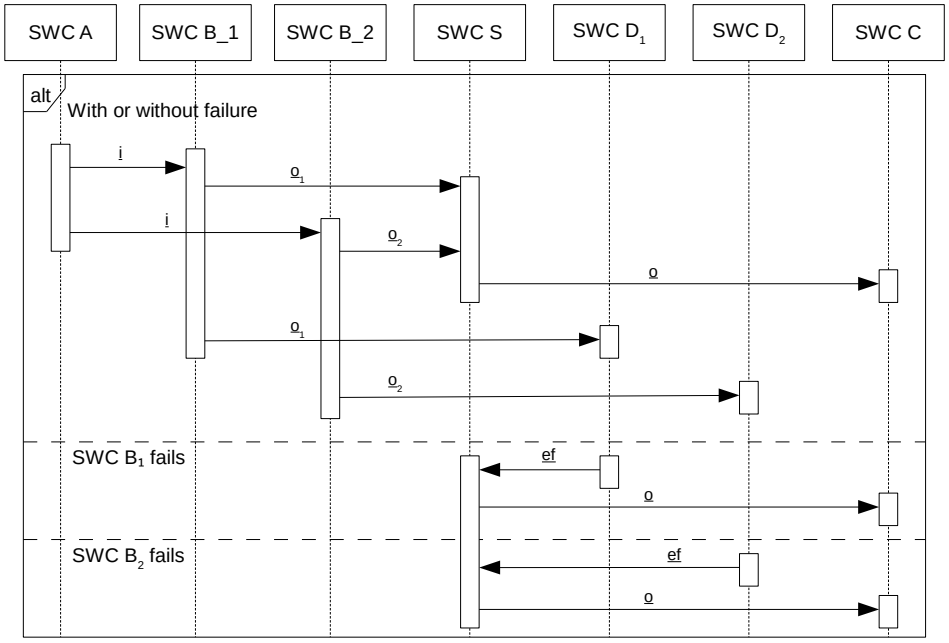


Figure 4.48: Sequence diagram of the extended software model - N-self checking programming

checking topology is therefore the higher among the two. Which leads to the recovery block topology, which includes the additional modules of the single version and the N-self-checking architectures, leading to the highest development effort of them all.

- *Modifiability:* Modifiability (as shown in table 4.7) is reflected by the number of SWC-s and interactions affected by a change in the functionality of the original SWC B. This criteria is important to evaluate, which side-effects changes have. In case of the single version technique, if the SWC B is changed, the SWC-s D and CpM have to be adapted. Regarding the interactions, the situation is similar: only the interaction from the SWC CpM to B is affected if the structure of the checkpoint memory has to be changed. In case of the N-version programming variant, all B_x SWC-s and in worst case even the voter have to be modified. The latter can be necessary, if the voting principle and the tolerance threshold of the voter (in case of inexact voting) need re-adjustment. The interactions are not affected. In case of the N-self-checking and the recovery block architecture, both B_x SWC-s and the diagnostic SWC have to be adapted. The interactions of the N-self-checking topology remain unchanged. Those of the recovery block architecture have to be changed the same way as in case of the single-version technique. Due to this, the modifiability of the single-version technique is considered being the best among the four fault-tolerant variants. The N-self-checking alternative is rated to be worse than its N-version programming competitor,

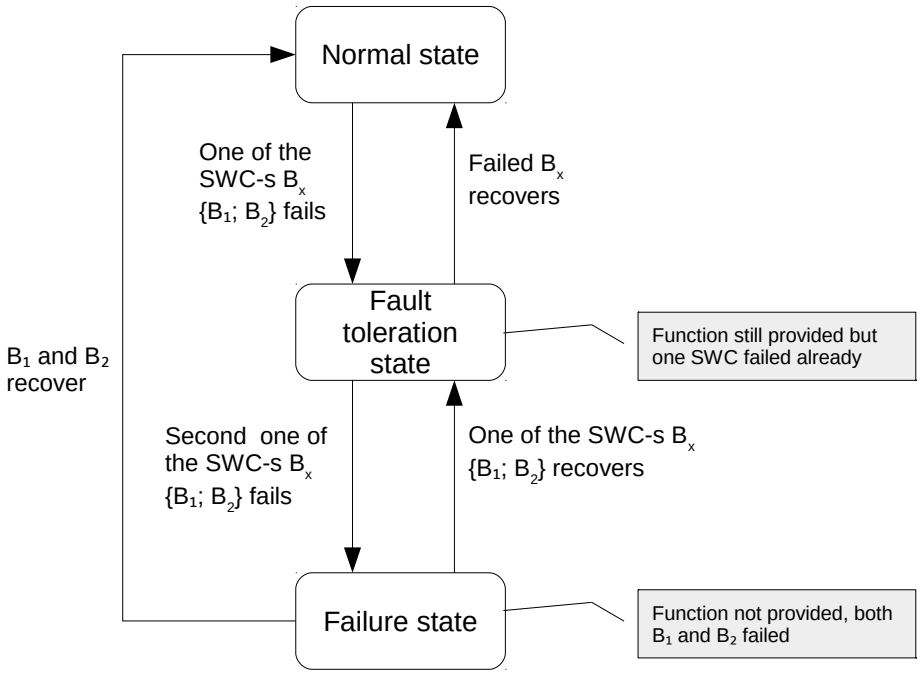


Figure 4.49: State machine diagram of the extended software model - N-self checking programming

due to the higher number (3 vs 2) of potentially complex SWC-s affected by the change of the SWC B_1. If the voter needs re-adjustment, and is fairly complex (which is rather unlikely), this advantage diminishes.

- *Fault tolerance capability:* As already stated above, the starting point architecture cannot tolerate any sort of faults. All other architectures can tolerate one fault but the single-version topology has one weakness. Faults of permanent nature can be tolerated only if the content of the checkpoint memory is slightly modified before executing the SWC B again. The effectiveness of this measure is not deterministic. Therefore, the fault-tolerant capability of the single-version topology is less potent than that of the multi-version variants.
- *Real-time behaviour (in case of a fault):* All architectures including a checkpoint-memory-based logic have an issue with this criteria, since in case of a fault an additional step is necessary (as shown in the respective state-machine diagrams). If the re-loading of the checkpoint memory and the re-executing of the SWC B_x takes too much time, the real-time behaviour of the SW can be adversely affected. The other architectures are not sensitive to this matter, since an (almost) immediate hand-over to the next instance of the SWC B is possible.
- *Complexity:* In case of the single-version topology the three additional modules and the required partitioning measures increase the complexity. The sequence diagram also shows an increase in the interaction complexity, in case of a failure. On the other hand, the interaction complexity remains the same as long as no failure occurs. The state machine diagram shows one additional state that needs to be passed in case of a failure. In case of the N-version programming architecture, the complexity increase is not significant. The static and dynamic aspects of the extended architecture are straightforward and relatively easy to understand. The number of additional interfaces and interactions is manageable, the state-machine of the SW is simple. The complexity of the N-self-checking topology is between that of the single-version technique and the N-version programming. The number of additional modules and interfaces is manageable, but the separate failure handling and the related interactions are increasing the complexity of the architecture. The need for an extra diagnostic function increases the complexity but the reaction is very straightforward. The state-machine of this topology is practically identical to that of the N-version programming architecture. Recovery blocks have definitely the highest complexity of all four investigated fault-

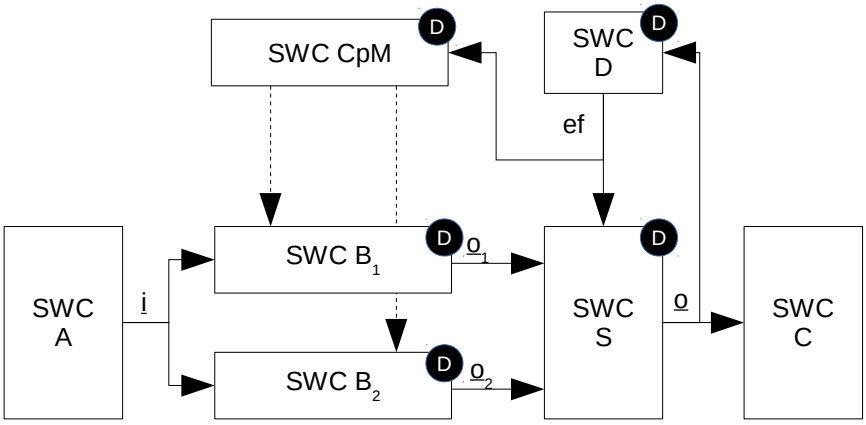


Figure 4.50: Block diagram of the extended software model - Recovery block

tolerant architectures due to the additional SWC-s, their complexity, and the relatively complex interactions and state-machine.

- *Performance:* In case of the single-version topology, the increase in the processing time is caused by the additional modules: the SWC D has to perform its fault detection functions even if no fault occurs. In case of a fault, the processing times of the SWC-s I and CpM can also be significant. The isolation of the output vector $\vec{o}$ is probably not too time-consuming, but the processing time of SWC I is constant, independently of the state of the software. The processing time to load the checkpoint memory into the RAM depends highly on the amount of data. In case of this architecture, additional Non-Volatile Random Access Memory (NVRAM) is consumed for the checkpoint memory, and more RAM space is needed for the variables of the new modules. The processing time increases dramatically in case of the N-version programming due to the tripled SWC B and the voter. Since the interactions are identical in the case with and without failure, the processing time increase is present all the time. The memory consumption increase is identical in its extent to the processing time multiplication. The three additional SWC-s require significantly more memory than the single SWC B. The required processing time and consumed memory increases as significantly as in case of the N-version programming. Although here is only one additional variant of SWC needed, but the SWC D can have the same complexity as well. The processing time consumption of the SWC S is negligible compared to the other SWC-s. The performance of the recovery blocks is twofold. The processing time in the failure-free case is not increased significantly. Only the routing function of the SWC S is to be considered additionally, but that SWC is fairly simple. In case of a failure, the situation looks completely different. This aspect is investigated at the bullet point "real-time behaviour". On the other hand, the memory consumption is the highest among the investigated topologies.
- *Testability:* All topologies can be tested very well on the SWC level. The testing on the integration level is slightly more difficult in case of the solutions involving check-point memories, since the HW-SW-interaction is more crucial in those cases. Therefore, integration testing on the target HW, involving the target base SW is essential.

Note, that no total score has been calculated. Since the numbers in table 4.8 are rather ranking values, a sum would not make sense. But still, an overall qualitative evaluation is possible.

Summary Based on table 4.8, the comparison results can be summarized as the following:

- If fault prevention can be applied (i.e. the fault tolerance capability is irrelevant), the starting point architecture is definitely the best solution. All evaluation parameters (except for the fault tolerance capability) get worse if an existing fail-safe SW architecture is extended to cope with fail-operational requirements. Therefore, fail-operational SW is always a bigger challenge both for

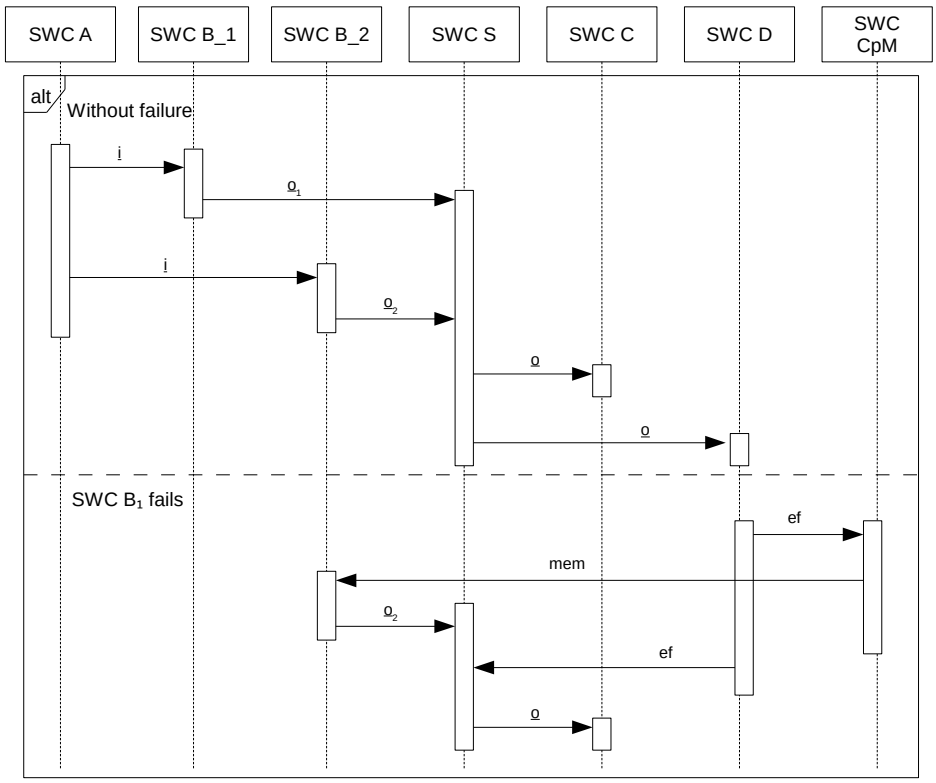


Figure 4.51: Sequence diagram of the extended software model - Recovery block

the organisation (see development effort) and from technological point of view (see the increased processing time, memory consumption, etc.).

- The fault tolerance capabilities of the three multi-version topologies are identical. The single-version architecture can tolerate HW-induced faults very well, but has issues with severe (systematic) SW faults due to the lack of diversity. Slightly changing the input vector $\vec{i}$ and/or the content of the checkpoint memory can lead to improvements, but cannot eliminate this concept weakness completely.
- In case of time-critical SW or SWC-s, the topologies based on checkpoint and recovery may not be applicable.
- If a high level of fault tolerance is required for a certain SWC, the N-version programming and the N-self checking architectures offer probably the best compromise. The N-version programming has the advantage that ASIL decomposition can be applied. On the other hand, if the SWC D can be kept very simple, the N-self checking has another advantage: only two diverse redundant implementations of the SWC B have to be found.

4.5 Aspects of technical independence [Sch14]

As already shown in the literature research (section 3), diversity and independence are two key factors of redundant architectures. The [ISO11] addresses this topic by requiring implicitly and explicitly independence or freedom from interference, where needed. Besides that, an analysis of dependent failures (or Dependent Failure Analysis (DFA)) is obligatory wherever independence or freedom from interference is to be proven. The DFA itself is mentioned on several occasions in the [ISO11]. In the core parts (i.e. parts 4, 5 and 6), a DFA is required both for the HW ([ISO11][part 5, clause 7.4.3.5])

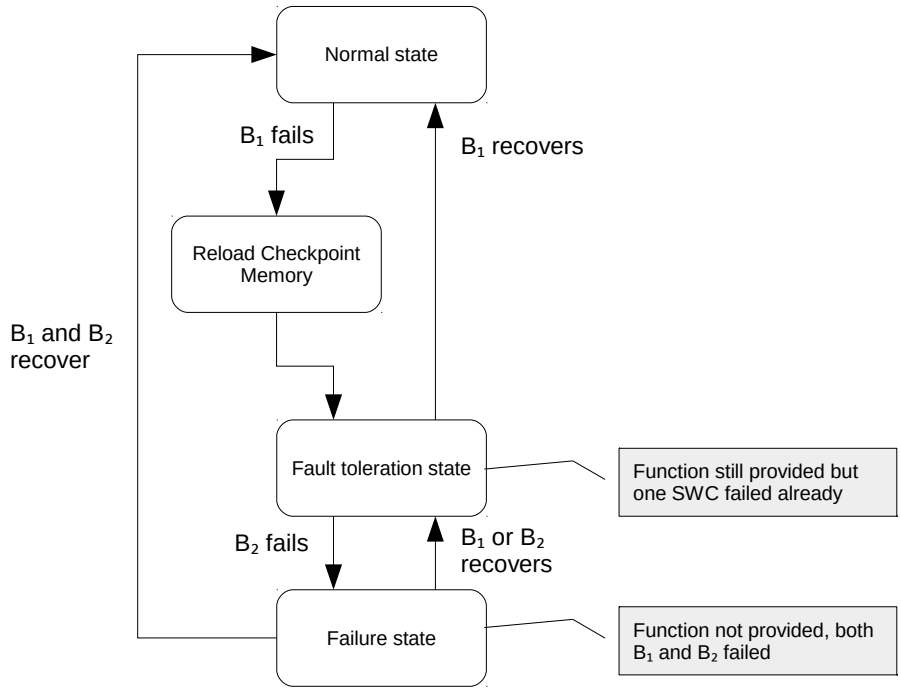


Figure 4.52: State machine diagram of the extended software model - Recovery block

and the SW ([ISO11][part 6, clause 7.4.12]). And finally, the [ISO11][part 9, clause 7] defines generic requirements related to this type of analysis.

As already shown in section 3.5.2, there is not much literature on DFA-s. In fact, [Cha09] reflects the same philosophy as the [ISO11][part 8, clause 7], offering only very little and inaccurate guidance on *how* to perform a structured analysis. The current industrial experience shows that there is no established DFA method complying with the [ISO11] that fits into an automotive analysis landscape and lifecycle.

In order to fill this gap, in this section a DFA method is derived from state-of-the-art DFA techniques from other domains.

Distinction between common cause and cascading failures

As already discussed in section 3.5.2, [Cha09] differentiates between cascading failures and common cause failures, depending whether their root cause is originated inside or outside of the system. At the same time, the same paper questions the correctness of this approach, but still sticks to the common understanding in the automotive industry. This thesis applies the same approach as [Cha09]: Distinction between common cause and cascading failures is essential. Since in some cases, only freedom from interference is required (and not full independence), the definition of the two dependent failure types has to be stated more precisely.

The definitions of the [ISO11] (described also in section 2.2) are very generic leading to the misunderstanding reflected in [Cha09]. In order to use definitions sensible for the DFA, the following shall be understood: dependent failures are undesirable if they affect elements that are meant to be independent. This leads to the following new terms:

- *Relevant common cause failures:* failure of two or more elements *meant to be independent*, resulting from a single specific event or root cause.
- *Relevant cascading failures:* failure of an element causing another element (or elements) *meant to be independent*, to fail.

	Starting point	Single-version	N-version programming	N-self checking	Recovery blocks
Development effort	+2	0	+1	-1	-2
Modifiability	+2	0	-1	-2	-2
Fault tolerance capability	-2	0	+2	+2	+2
Real-time behaviour	+2	-2	+2	+2	-2
Complexity	+2	-1	+1	0	-2
Performance	+2	+1	-2	-2	+1
Testability	+2	+1	+2	+2	+1

Table 4.8: Comparison of fault tolerant SW topologies

Using these subsets of common cause and cascading failures, the origin of the fault and whether they are internal or external become irrelevant. Therefore, the main distinction between the two dependent failure types is their fault propagation behaviour: In case of common cause failures, the two elements, meant to be independent, fail in *parallel* because of a common root cause. Whilst in case of cascading failure of two elements meant to be independent, the failure of one element leads to the failure of the other one, *sequentially*.

DFA analysis procedure

First of all, a basic analysis procedure shall be derived by simplifying and extending the methods shown in [Balo7] and [Ern15]. This process consists of the following three steps:

1. Identify functions where independence or freedom from interference is required
2. Identify potential dependencies / coupling factors
3. Define appropriate counter-measures

As it can be seen in the list above, the method introduced here is explicitly focusing on *functions*. The goal of this approach is to yield a generic analysis technique that can be applied on various levels:

- Logical functional architectural level
- Concept preliminary architectural level
- System architectural level
- HW and SW architectural level

Functions where independence or freedom from interference is required

As already shown in section 3.2.2, [Balo7] identifies the spots in the design where a DFA is required by categorising the AND-gates of the FTA. This is not always practicable in automotive systems, due to the fact that an FTA is not always performed. Besides that in automotive systems, the AND-gates are actually already pre-selected. The reason for this is that the [ISO11] does not require the analysis of each possible fault combination, but only of those that can be derived based on the (functional or technical) safety concept. Based on this, the following categories can be derived:

- *Diverse redundant functions (including ASIL-decomposition)*: Diversity is one aspect of independence and independence is the cornerstone of ASIL-decomposition. Such design decisions have to be included in the safety concept and therefore in the MPF analysis as well. Note, that this class is close to categories 1 and 3 of [Balo7] (although that one only includes redundant HW). For class 1, [Balo7] requires a full DFA, whilst for class 3 only a simplified one. In the analysis method presented in this section, a full DFA will be applied for both.

- *Safety mechanisms (and the functions they are focusing on)*: These are the core content of each safety concept. Therefore, they definitely have to be included in the MPF analysis. Note that this represents category 1 AND-gates of [Bal07]. This category has been selected as top priority by [Bal07], and is of great importance also in systems conforming to the [ISO11]. For this class, [Bal07] requires a full DFA.
- *Coexisting functions with different ASIL-s*: In order to ensure that functions with lower ASIL-s do not impair functions with higher ones, freedom from interference has to be ensured. In all these cases a limited DFA shall be performed, focusing only on cascading failures.

Categories 2 and 4 of [Bal07] (where [Bal07] requires only a very limited qualitative risk mitigation) are not covered by the three classes listed above, with the following rationale:

- Category 2 addresses mechanical HW faults, which are excluded from the scope of the [ISO11].
- Category 4 covers implausible MPF-s or is covered by the redundant functions or the safety mechanism classes of the categorisation above.

The advantage of applying these three categories (over analysing each AND-gate in an FTA) is that all three can be easily identified based on existing design documents. Architecture documents on the system, HW and SW level (including the ASIL allocation to the components) can be used to identify all three categories. Technical and/or functional safety concepts can be used to identify redundant functions and safety mechanisms. Safety analyses can support the allocation of safety mechanisms to their respective diagnosed functions. Therefore, the first step of the proposed DFA technique can be performed on the chosen level by simply utilising already existing work products.

Dependency model - potential coupling factors

Before moving on to the dependency model that represents the core result of this section, the following merged list shows coupling factors based on [ISO11][part 9, clause 7], [Bal07], [Cha09], [Mos95],[Ern15], [Gan74] and [Pen12]:

- Requirement specification faults [Bal07] [Cha09]
- Development/design faults [Bal07] [Cha09] [Mos95] [Gan74] [ISO11]
- Common supporting systems [Mos95] [ISO11]
- Physical signal routing [Cha09]
- Identical part or system design [Mos95] [Pen12]
- Same component or system location [Mos95]
- Unintended impact by data or timing (in case of SW) [Ern15]
- SW faults [Cha09]
- HW faults [Bal07] [Cha09] [Gan74] [ISO11]
- Production faults [Bal07] [Cha09] [Mos95] [Pen12] [ISO11]
- Operational, repair and maintenance faults [Bal07] [Mos95] [Gan74] [Pen12] [ISO11]
- Installation faults [Bal07] [Mos95] [ISO11]
- Environmental influences [Bal07] [Cha09] [Mos95] [Ern15] [Gan74] [ISO11]
- Misuse [Cha09]
- Stress [ISO11]

The list above has several issues that makes a DFA very complicated. First of all, some of the categories are very generic. For example, there are thousands of potential HW or SW faults that should be analysed during a DFA, which is impossible. Besides that, the list is based on field experience and expert judgement, lacking a clear and structured model that could guide the analyst through the process. The third issue is that all lists in the mentioned sources are tailored for a specific application and are not suitable as generic model. In order to fill these gaps, figure 4.53 shows a derived dependency model, including the following dependencies:

1. *Common inputs:* Faults of common inputs can corrupt both functions.
2. *Communication between the functions:* The recipient function can be affected by the false information.

3. *Unintended impact:* Coupling mechanism via a path that is not part of the design (e.g. cross-talk between signal lines). One function can affect the other via this path during normal operation or in case of a fault.
4. *Overlapping:* Components that are used by both functions (e.g. same H-bridge used by two shut-down paths). Any fault of these components affects both functions.
5. *Identical components:* Components with identical design used in each path (e.g. identical power supply chips used for two redundant microcontrollers). These components may fail simultaneously due to a systematic fault (e.g. temperature robustness issues).
6. *Common resources:* Faults of commonly used resources can corrupt both functions (e.g. same RAM region used by two SW functions.
7. *Systematic coupling mechanisms:* This category denotes all systematic causes (caused by human or tool errors) that can lead to the simultaneous failure of both functions.
8. *Common environmental influences:* This category denotes common disturbances caused by environmental factors affecting both functions.

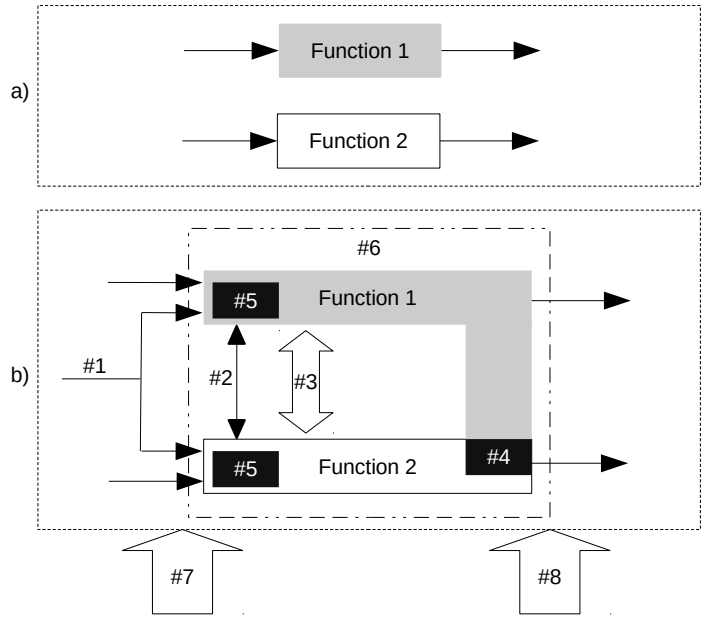


Figure 4.53: Dependency model, a) Completely independent function pair, b) Potential dependencies between two functions

Then, applying a structured procedure, the functions chosen to be analysed can be investigated for each potential coupling factor. Figure 4.54 shows how a simple architectural design description can support the identification of common inputs, communication, overlapping and identical components. This is one of the main advantages over the currently applied methods: These dependency types can be analysed using existing design work products.

The remaining dependency types can be analysed only by using pre-defined catalogues. These catalogues shall be defined for the particular domain in focus.

- *Common resources* can be identified in two steps:
  1. Identify the related resources (e.g. clock, RAM, power supply, execution time) based on a pre-defined catalogue. Note that this catalogue needs thorough domain expertise.
  2. Identify the commonly used ones.
- *Unintended impact.* Note that this catalogue needs thorough domain expertise.
- *Systematic coupling mechanisms* can be derived from the lifecycle of the particular application in focus. Faults in each step of the lifecycle (e.g. concept, design, management, implementation,

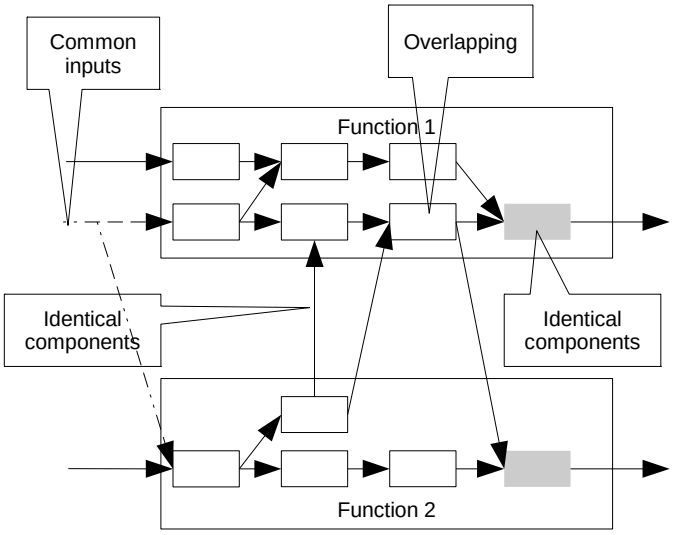


Figure 4.54: Identifying dependencies based on a structured analysis

manufacturing) shall be addressed. Note that this catalogue is plain and straightforward. The main issue that needs to be kept in mind is that each domain has its own lifecycle. This is why the model presented in this chapter keeps this category so generic. Note that most lists of the investigated sources address these systematic coupling mechanisms (e.g. manufacturing faults, installation faults), defined for their particular application.

- *Common environmental influences* can be identified in two steps:
  1. Identify environmental factors (e.g. electromagnetic radiation, temperature, α-particles). Note, that this catalogue is relatively simple and straightforward, since the integration environments of most applications are well-known. This list can be even derived based on commonly used environmental test standards (e.g. LV 124).
  2. Identify each environmental factor that can lead to a common mode malfunction.

Since in some cases the distinction between common cause and cascading failures is essential, it is important to categorize the coupling factors above:

- The following coupling factors can lead to relevant common cause failures: common inputs, identical components, overlapping, common resources, systematic coupling mechanisms, common environmental influences.
- The following coupling factors can lead to relevant cascading failures: communication, unintended impact, overlapping, common resources.

As it can be seen, the dependency types *overlapping* and *common resources* can lead to both types of dependent faults. It has to be noted here that the fault propagation is different in the two cases, though. For example, a SWC overwriting a RAM region of another SWC is a cascading failure. At the same time the corruption of a larger RAM region due to e.g. intense α-radiation, affecting two SWC-s, is a common cause failure. Both are originated from a common resource but their fault propagation behaviour is different.

Define possible counter-measures

If dependencies are identified using the method described in the previous chapter, these coupling shall be eliminated or controlled. This can be achieved by breaking the fault propagation paths of common cause or cascading faults.

Figure 4.55 a) shows the following three possibilities to deal with dependencies related to relevant common cause failures:

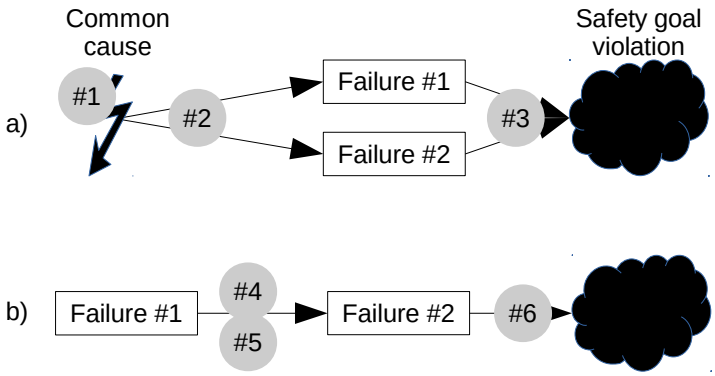


Figure 4.55: Techniques to eliminate or control dependencies, a) Relevant common cause failures, b) Relevant cascading failures

1. *Eliminate the root causes:* No common inputs, no common resources, eliminate systematic coupling mechanisms (e.g. by applying a state-of-the-art development process; using appropriate production control measures); etc.
2. *Control the root causes:* Implement safety mechanisms that can control the failures of common inputs, common resources, identical components, overlapping components, etc.
3. *Diverse design:* due to this, the two functions fail in different ways so that the safety goal is not violated.

Based on figure 4.55, relevant cascading failures can be eliminated or controlled the following three ways:

5. *Eliminate the coupling mechanisms:* No communication between the two functions, no unintended interfaces, no common resources.
6. *Control the initial failures:* Detect failures of function 1 and react to them (e.g. memory protection in modern microcontrollers).
7. *Control the effects of the caused failure:* Detect failures of function 2 and react to them. Note that this should be performed by a third, independent function. This makes this approach very difficult.

Interfaces of the DFA to other analysis techniques

Since the DFA cannot be the only analysis performed for an automotive product, it is important to investigate how it fits into the overall analysis landscape. In most analysis concepts, the basis is provided by an FMEA and/or an FTA. Therefore, in this section the interfaces to these analysis techniques will be investigated.

An FMEA can provide limited support to the DFA due to the reason that its analysis capabilities related to MPF-s are limited. If the FMEA is done properly, and the interfaces to the DFA are kept in mind, the following supporting information can be derived from it:

- Functions and related safety mechanisms can be allocated to each other.
- Cascading failures can be analysed in a very systematic way (if using failure nets).
- Failure modes related to common environmental influences and unintended impacts can be identified. This contribution may be limited.
- Dependencies by overlapping can be identified. For this purpose, an inductive analysis direction can be very beneficial.
- Dependencies by common inputs can be determined.

Contrary to the FMEA, the FTA can analyse MPF-s properly. Therefore any DFA can be supported by an FTA extensively. First of all (as already seen in [Balo7]), each AND-gate should be subject to a DFA. Besides that, cascading failures can be identified as well. Cut-set analysis can point out various types of coupling mechanisms: common resources, identical components, overlapping and common inputs. It is important to emphasise that cut-set analysis cannot provide these results automatically.

Figure 4.56 shows a simple fault tree, indicating two dependencies. The two cut-sets of this fault tree are {Sensor 1 electrical failure; Sensor 2 electrical failure} (marked with [1]) and {Sensor power supply failure} (marked with [2]). These suggest potential identical components and a common resource, respectively.

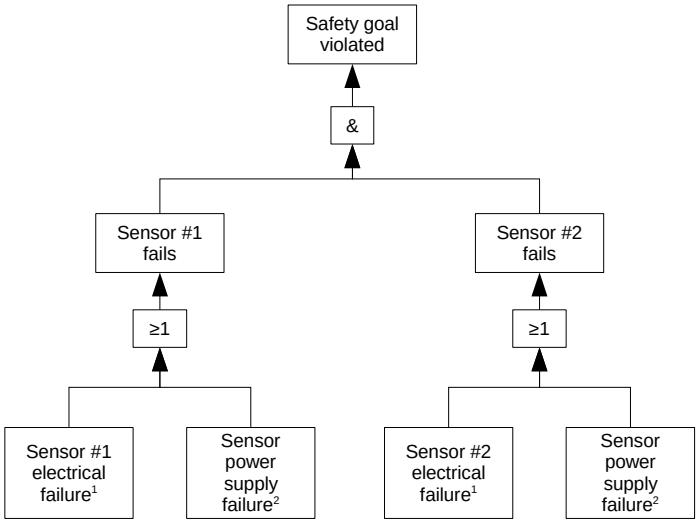


Figure 4.56: Fault tree indicating dependent failures

4.6 Summary - Fail-operational systems from the standpoint of the [ISO11]

As presented in the previous sections, the [ISO11] can be applied for fail-operational items. There are several extensions and clarifications necessary. This is no surprise given the fact that the [ISO11] has been primarily meant for fail-safe items. The main results of this chapter can be summarized as follows:

- The applicability of fault tolerant system-, HW- and SW architectures commonly used in other domains has to be evaluated for the automotive domain. Basic comparison of such architectures has been performed in sections 4.4.3 and 4.4.4, using state-of-the-art qualitative and quantitative analysis techniques. Based on that, the strengths and weaknesses of these architectures have been pointed out and hints to their applicabilities have been given.
- For the HW, the impact of using incompletely redundant architectures has been investigated, leading to a proposed new HW metric to be used instead of the SPFM and LFM for ASIL A and B.
- For the SW it has been shown how important the appropriate combination of fault prevention and fault tolerance measures is. An extension proposal for the [ISO11] has been given that fits into the overall approach of the [ISO11][part 6].
- A structured and generic DFA method has been developed that fits into a common automotive analysis landscape.

The key factor for the success or failure of a fail-operational product development will be how the designers cope with incompletely redundant architectures. Most designers will most probably prefer such architectures, due to several reasons: the [ISO11] does not require complete redundancy; costs; packaging space; weight. Therefore, the challenge is to find an optimum, where the fault tolerance capabilities of the item are sufficient, and the design targets related to costs, packaging space and weight are met. This aspect is addressed in the next chapter of this thesis.

5 The redundancy allocation problem in automotive systems

5.1 Introduction to reliability optimisation problems

5.1.1 Introduction to and classification of reliability optimisation problems and their solutions [Hiredb]

As already stated at the end of section 4.6, optimising incompletely redundant system-, HW- and SW architectures will be of paramount importance.

Reliability optimisation has been in the focus of research since the 1960-s [Pra00]. The considered solutions include the enhancement of component reliability, application of redundancy, or the combination of both [Pra00]. Due to the five decades of research, there is a vast amount of literature on this subject. The first paper reviewing the literature on this matter until 1977 was [Kuo77]. Subsequently [Pra00] reviewed the advance from 1977 until 2000. Based on that, [Wan07] summarises the results of the publications between 2000 and 2007. And last but not least, [Ibr13] gives a very up-to-date outline and classification of recent papers on this topic.

In order to narrow down the literature research for the issue in focus, the classification of the papers related to reliability optimisation needs to be understood. Based on [Kuo77], [Pra00], [Wan07] and [Ibr13], the related papers can be categorised according to the following parameters:

1. System architecture
2. Homogeneity of the redundant components
3. Number of system states
4. Optimisation model
5. Reliability modelling (or calculation method)
6. Determinism of the target(s) and constraint(s)
7. Perfect or imperfect diagnostic coverage
8. Allocation model
9. Solution method
10. Field of application

One of the most important characteristics is the type of the *architecture* the optimisation is applied on (figure 5.1). The mathematical and logical models that are used for the optimisation depend on the topology, due to obvious reasons. Research has focused on the following architectures so far:

- Series-parallel systems (figure 5.1 a)) are rather uncommon. In this case, the system consists of n parallel paths forming a static redundant architecture, each consisting of 1...m stages. The system fails, if all n paths fail. A path fails, if one of its component fails.
- Parallel-series systems (figure 5.1 b)) represent the most common architecture used for Redundancy Allocation Problem (RAP) [Pec15]. These systems are gained by taking a serial system consisting of m stages and adding redundant elements ($n-1$ at maximum) to each subsystem. The task is then to find the optimal level of redundancy and/or the optimal component selection for each stage. This topology is based on static redundancy. Such a system fails, if one of the stages fails. One of the stages fails only, if all components in that stage fail.
- Element standby systems (including hot, warm and cold standby) (figure 5.1 c)) are very similar to the parallel-series systems, but assume a dynamic redundant architecture instead of a static one. Each stage consists of a primary unit (P_{xy}) and $n-1$ spare units (S_{xz}). If the primary unit fails, the next spare steps in. Note, that in case of cold or warm standby, the calculation of the reliability is mathematically far more complex [Ng11].

- System standby systems (including hot, warm and cold standby) (figure 5.1 d)) are based on series-parallel systems, but in this case there is a primary path and spares that can step in in case of a failure. This topology is therefore based on dynamic redundancy.
- k-out-of-n systems look identical to parallel-series systems (figure 5.1 b), [Cha12]) but in this case a stage fails if $n - k + 1$ components in that stage fail.
- Bridge topologies (figure 5.1 e)) represent one of the complex architectures for reliability-relevant systems.
- General networks and unspecified architectures represent complex architectures that cannot be categorised as any of the topologies listed above. The biggest challenge of these architectures is, that their reliability functions cannot be calculated by relatively simple generic equations (like the example equation 5.1). In these cases the reliability can be calculated only based on the specific architecture ([Win86], [Kuooo], [Yum93], [Red97]). For these architectures, it is assumed that the global architecture (i.e. the connections between the subsystem) remains the same, only the redundancy inside these subsystems can be enhanced.

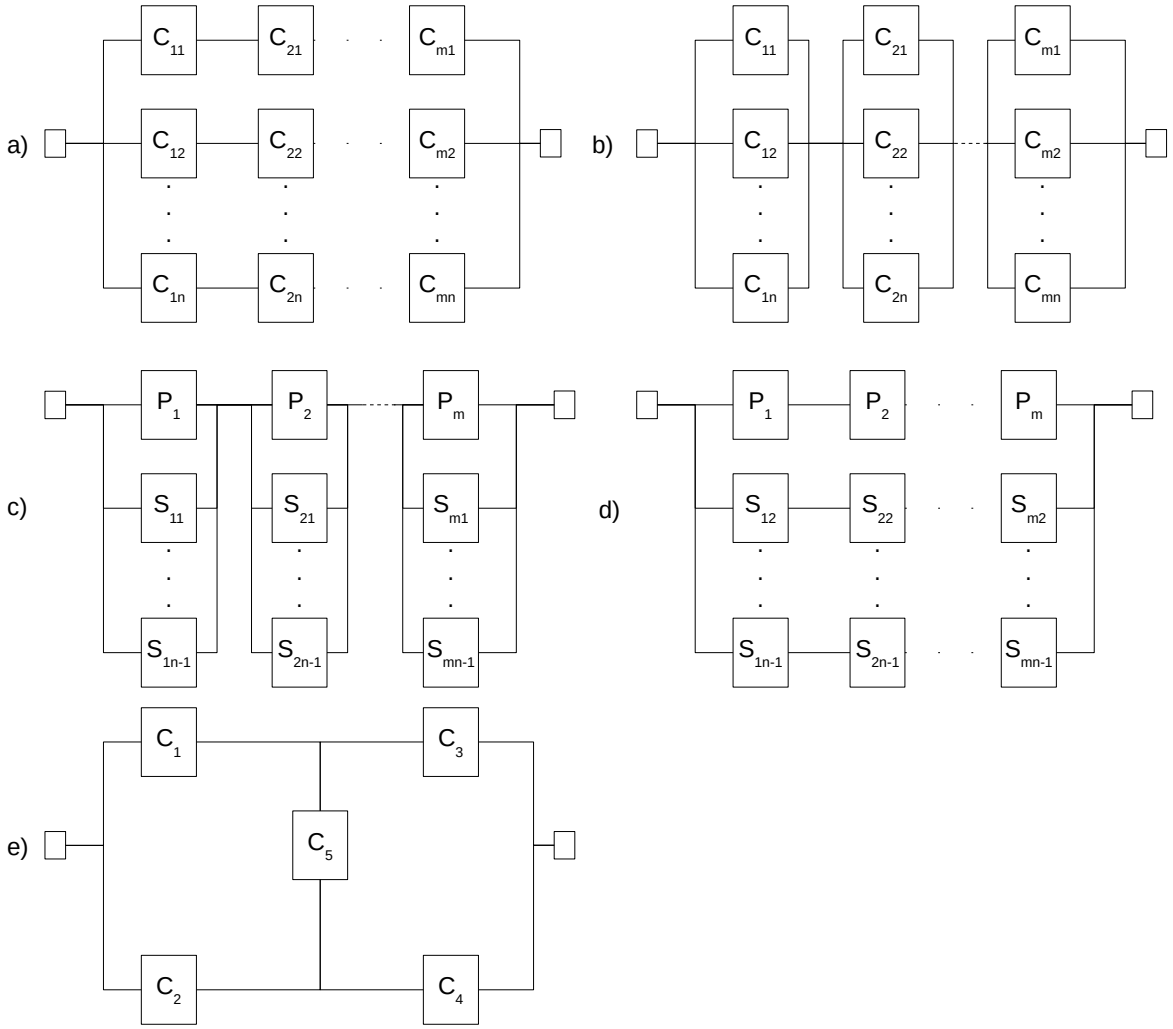


Figure 5.1: Redundancy optimisation - Architectural variants [Kuo77], [Praoo], [Wano7], [Ibr13]

Another key factor is the *homogeneity of the redundant components* [Nako7]:

- In case of *homogeneous redundancy*, the problem is limited to the determination of the number of redundant components.

- In case of *heterogeneous (i.e. diverse) redundancy* (also called component-mixing), not only the number of redundant components but also their exact type has to be optimised.

The papers can also be distinguished based on the assumed *number of the states* of the system and its components:

- In case of *binary* models, the components and therefore the system has two states: functioning or failed.
- *Multi-state* models consider various performance levels of the components and the system. In that case the algorithm to determine the system state, based on the individual component states, is critical. As shown in [Wan07], most papers on this matter apply complex mathematics: the universal generating function approach ([Lev01], [Lis00], [Lis01], [Li12]). On the other hand, [Hua08] and [Aga09] both calculate the system state as a probability based on the sub-states of the components. In the investigated literature and in the overview papers, no method is shown where system level *failure modes* are *mapped* to their respective *causes* in the subsystems and components.

The *optimisation model* defines the target(s) and constraint(s) of the optimisation. There are infinite possibilities here, but the following ones are the most common in the literature:

- Reliability maximisation, considering a defined set of constraints (e.g. cost).
- Cost minimisation, considering a defined set of constraints (e.g. reliability).
- Multi-objective problems define the target not as a singular value but as a set of values. In this case, for example, the solution with the highest reliability and the lowest cost is targeted. Obviously, in most cases, these sub-goals are contradictory. The optimisation algorithm therefore provides several potentially optimal results, and allows the designer to choose [Cha12].

If reliability is a target or a constraint, *reliability modelling* is also a differentiating aspect. Reliability as such can be represented by various mathematical factors, depending on the available data, the failure model and the target industry:

- MTTF
- Failure rate
- Other reliability related factors

An important characteristic aspect of the reliability, cost, etc. is whether they are *deterministic*, since this has a significant impact on the mathematical model.

- In most papers, component attributes (e.g. cost, reliability) are deterministic (e.g. [Smi04]). This simplifies the calculation of the target and constraint factors.
- In some cases, though, these factors are fuzzy [Xie13].

Another factor that affects the optimisation results essentially, is whether the diagnostic coverage is assumed to be 100%:

- Most papers consider $DC = 100\%$, also known as *perfect switching*.
- Some research focuses on the more realistic case of *imperfect switching* ([Ham14], [Utk95], [Kno12], [Mis99], [Sup04]). Imperfection can result from failures of the switching from one instance to a redundant counterpart, or from gaps in the diagnostic functions.

There are various ways to improve the reliability of the given architecture. This is defined as *allocation model*:

- In case of *redundancy allocation* the means of optimisation is to add redundant elements.
- In case of *reliability (or availability) allocation* applies technical measures to improve the reliability of a certain component (e.g. additional maintenance).
- A combination of the two methods above is also possible [Pra00].
- *Components or subsystems allocation* is based on the selection and the arrangement of the components or subsystems [Pec15].

After the problem is sufficiently defined by the factors above, the last aspect to clarify is the *solution method*:

- *Exact solutions* are based on mathematical programming methods (e.g. reduced gradient methods [Praoo]) that are able to identify the absolute optimum. These methods are very computation intensive, sometimes even impossible, and therefore seldom used.
- *Heuristic methods*
- *Meta-heuristic solutions*, including Evolutionary Algorithm (EA), Particle Swarm Optimisation (PSO), Simulated Annealing (SA). These represent the most promising methods that have been extensively researched in the 21^{st} century [Praoo].

Regarding the *field of application*, the major part of the related literature focuses on systems and HW. In these cases, the reliability is understood as lack of random HW faults. There are some papers focusing on SW related aspects of reliability optimisation, though. [Ale15] for example considers the deployment of SW components on different ECU-s, as a method for reliability optimisation. [Ash93] is concerned with the RAP of SW using Commercial Off The Shelf (COTS) SW components. But a very common RAP mathematical problem statement is used, without any SW related specifics considered.

Besides the factors listed in the sections above, there is also an assumption applied by all papers that is intended to simplify the model: Failures of the individual components are *independent* ([Sha13], [Smio4], [Yaoo9]). This means, that the failed components do not damage the system (i.e. there are no cascading failures between the redundant elements); and that redundant elements do not fail simultaneously (i.e. there are no common cause failures of redundant elements).

Most papers are not related to any specific industrial domain and remain therefore highly theoretical and focused on the mathematical solution of the problem [Ibr13]. Due to this, there are only a few case studies where the applicability of these methods can be judged. There is only one paper that addresses automotive systems in its title: [Buho9]. This paper states that *"redundancy allocation is a widely used method [...] in the automotive domain"*, but this statement is not supported by any evidence. Although the examples used by the authors are automotive systems, the RAP problem statement and the solution do not consider any specifics of automotive systems (see section 5.2.1). Based on the experience of the author, the RAP is practically unknown in the automotive industry. The main reason for this is that redundant architectures are avoided due to their impact on cost, weight and packaging space.

5.1.2 Simple RAP example [Yao09]

In order to explain how a particular RAP is solved, a brief overview of [Yaoo9] is given below. The RAP is solved for a parallel-series system (figure 5.1 b), using reliability and cost as two objectives, and weight as the only constraint. Note, that this is therefore a multi-objective problem statement. The reliability is considered to be deterministic, and is represented as mathematical reliability. There is no component-mixing assumed, the subsystems incorporate homogeneous redundancy. The components are either fault-free or failed (i.e. a binary state model is applied). Perfect switching is assumed. The allocation model is pure redundancy allocation. The solution is based on a multi-objective EA. The optimisation is applied on system level, focusing on the HW.

The reliability of the system is calculated as:

$$R = \prod_{i=1}^{s} [1 - (1 - r_i)^{a_i}] \tag{5.1}$$

where s is the number of the subsystems, r_i is the reliability of the components in subsystem i, a_i is the number of components in the subsystem i.

The cost of the particular system configuration can be calculated as:

$$C = \sum_{i=1}^{s} c_i \left(a_i + e^{\theta_i a_i} \right) \tag{5.2}$$

where c_i is the cost of one component in the subsystem i, θ_i is a constant, and $e^{\theta_i a_i}$ is the additional cost of the component interconnections.

The weight can be calculated as:

$$W = \sum_{i=1}^{s} w_i \left(a_i + e^{\gamma_i a_i}\right) \tag{5.3}$$

where w_i is the cost of one component in the subsystem i, γ_i is a constant, and $e^{\theta_i a_i}$ is the additional weight of the component interconnections.

Based on these equations, the formal description of the multi-objective RAP is described as the following: Find the optimal a_i, $i = 1, ..., s$

$$Max\, R = \prod_{i=1}^{s} [1 - (1 - r_i)^{a_i}] \tag{5.4}$$

and

$$Min\, C = \sum_{i=1}^{s} c_i \left(a_i + e^{\theta_i a_i}\right) \tag{5.5}$$

Subject to the following constraint:

$$g_1 = W = \sum_{i=1}^{s} w_i \left(a_i + e^{\gamma_i a_i}\right) \leq W_c \tag{5.6}$$

Based on that the optimal $p = (a_1, a_2, ..., a_s)$ ($a_i = [1, n_s]$) is determined using a genetic algorithm.

5.1.3 Generic workflow to solve a RAP

As it can be seen from the example above, a RAP can be solved in the following steps:

1. Define the parameters of the RAP, as defined in section 5.1.1.
2. Find a mathematical model for the architectures, objectives and constraints, based on the parameters above.
3. Define the optimisation algorithm.
4. Set up the input value set to start with. This includes the base architecture and the set of available components.
5. Perform the optimisation.

Section 5.2 will follow exactly these steps.

Figure 5.2 shows a simplified RAP solution process. As it can be seen, the key factors for the solution are the model of the component selection, the model of the fitness calculation (i.e. the method to calculate how good a certain variant is) and the optimisation method itself. Note, that this figure already implies a meta-heuristic solution. For further details of the selection of the optimisation method, see section 5.2.4.

5.2 Solution of an automotive RAP

5.2.1 Define the parameters of an automotive RAP

Using the eight factors defined in section 5.1, an automotive RAP can be classified as follows:

1. *System architecture:* Most automotive system architectures are serial or unspecified architectures. Besides that, the resulting system architectures show various specialities, which will be explained below.
2. *Homogeneity of the redundant components:* As already explained in section 4.5, diverse redundancy is essential for fail-operational systems. Therefore, component mixing is highly recommended for automotive RAP-s.

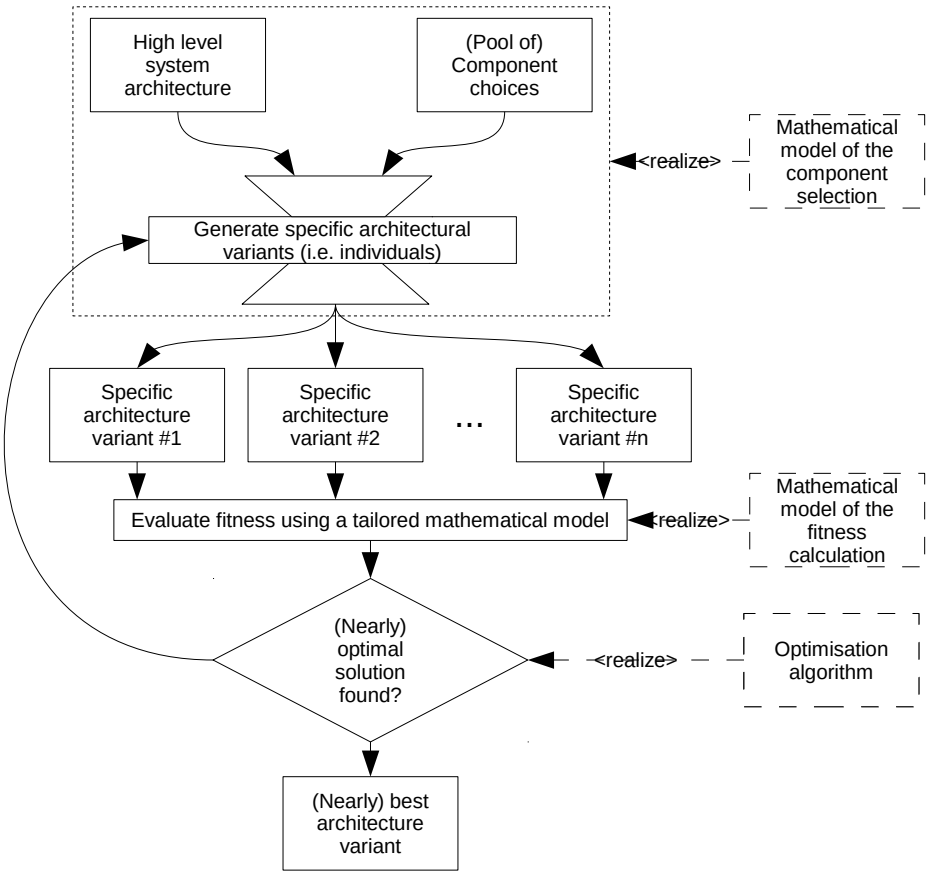


Figure 5.2: Simplified RAP solution process

3. *Number of system states:* Due to the reliability modelling (see below), multi-state systems shall be assumed, where the states can be represented by { fault-free; failure mode 1; failure mode 2; ...; failure mode n }. The calculation method of the overall system state based on the subsystem or component sates is highly complicated.
4. *Optimisation model:* The optimisation model can be carried over from the industrial domain independent literature in most cases. Cost, weight and reliability can be used as targets and constraints in various combinations. In many applications, packaging space need can be an additional constraint. Related to the optimisation model, there is one aspect that can reduce the size of the solution space: level of expected redundancy. Due to the high cost pressure and the lack of packaging space, it is hardly possible to use redundancy concepts higher than the 3^{rd} order. Therefore, TMR and dynamic redundant architectures (as investigated in section 4.4.3) can be taken as possible subsystem architectural variants. Hence, in case of an automotive RAP the question is not *how much* redundancy shall be applied, but rather *whether* it has to be applied, and *where* does it make the most sense.
5. *Reliability modelling (or calculation method):* As already presented in section 2.4.4, the HW metrics of the [ISO11] shall be used as reliability factor. When applying these metrics, another issue is to be dealt with: imperfect diagnostics. The RRM developed in section 4.4.3 can be an alternative. Note, that in case of more than one safety goal, the HW metrics have to be met for all of these safety goals. This may imply contradictory requirements on the optimisation.
6. *Determinism of the target(s) and constraint(s):* The failure rates of automotive components are treated as deterministic by the [ISO11]. Cost, weight and packaging space are not completely deterministic, since the system level cost, weight and packaging space is not simply the sum of the component

level values. But for the sake of simplicity these can be treated as deterministic.

7. *Perfect or imperfect diagnostic coverage:* Imperfect diagnostic coverage is assumed. This is the basis for the calculation of the HW metrics of the [ISO11].
8. *Allocation model:* Typical automotive design decisions imply a combination of redundancy and component allocation with component mixing.
9. *Solution method:* As stated in [Praoo], meta-heuristic methods offer the most promising results. A specific selection will be made in section 5.2.4.
10. *Field of application:* As shown in section 4.4.3, the optimal utilisation of component redundancies is one of the key issues in case of safety related HW. This is especially true for ASIL A and B products, where a relatively low level of redundancy is sufficient to cope with the HW metrics of the [ISO11]. Due to this, RAP is also relevant on the system level, focusing on the random HW failures of the subsystems and their components. As shown in 4.4.4, the allocation of redundant elements in case of fail-operational safety related SW is based mainly on the complexity of the SWC-s and their allocated ASIL. Therefore, the RAP (as for example shown in [Ash93]) is not relevant. SWC deployment may be an aspect in theory (as shown in [Ale15]) but in practice, most safety related functions cannot be easily deployed in other ECU-s due to integration-related technical -, and intellectual property protection reasons. Nonetheless some aspects of the methods developed in the following chapters can be applied on SW too.

Besides that, there is one important characteristic that is related to two of these classification criteria: to the architecture and the allocation model. The architectural models presented in section 5.1 all assume a 1-to-n connection between the elements of the base architecture and the related redundant elements. In automotive products, functional integration is very common on the system and on the component level. For instance one single semiconductor product may consist of a watchdog and a power supply function. In this example, it may occur that this single component is replicated by introducing two additional components: a separate watchdog and a power supply Integrated Circuit (IC). In this case an m-to-n relation between the elements of the base architecture and the related redundant elements shall be assumed.

Based on the literature research presented in 5.1 and the characteristics identified above, it can be stated that the following specifics of automotive systems have not been investigated yet in relation with RAP-s:

- HW metrics of the [ISO11] used as target or constraint.
- Failure-based multi-state system dynamics, where the states represent failure modes that have to be mapped to high level failure modes of the system.
- m-to-n redundancy allocation.

5.2.2 Mathematical notations

This subsection introduces the notations (based on [Smio4]) used in section 5.2.3. Note, that the index X in the notations (as e.g. in s_X) denotes options of elements, that particular notation can refer to (as e.g. in $s_{C_{ij}}$ denotes the number of failure modes of component C_{ij}, and s_S denotes the same for the system). Therefore the index X can have one of the following values, depending on the particular notation:

- S: system,
- C_{ij}: component j in subsystem i,
- a_{ij}: component choice j for subsystem i.

Matrix and vector elements are always represented by the vector name and the indices in parentheses, separated by commas (e.g. $\mathbf{M}^{M}_{a_{ij}}(a,b)$) for better readability.

The system architecture is represented by the following notations:

m	number of subsystems (1)
S_i	subsystem i
C_{ij}	component j in subsystem i
C	set of components in the system
$\mathbf{n}$	vector containing the number of components in subsystem i (1)
	vector containing the number of minimally
$\boldsymbol{\kappa}$	required functioning components in each subsystem
t	operational time of the system (1)

The failure models on the component and system level are denoted by the following:

$\mathbf{f_X}$	failure mode vector of component C, or of component choice a_{ij}, or of the system (1)
$\boldsymbol{\lambda_X}$	failure rate vector of component C, or of component choice a_{ij}, or of the system (FIT)
s_X	number of failure modes of component C_i or of component choice a_{ij} (1)
s_s	number of failure modes on the system level (1)
	failure mode criticality vector failure rate vector of component C,
$\boldsymbol{\gamma_X}$	or of component choice a_{ij}, or of the system (1)
$\mathbf{M_{X\rightarrow S}}$	FEMM between component C or component choice a_{ij} and the system level (1)
$\lambda_{X,crit}$	critical failure rate of component C, or of component choice a_{ij}, or of the system (1)
$\lambda_{X,crit,d}$	detected critical failure rate of component C, or of component choice a_{ij}, or of the system (FIT)
$\lambda_{X,crit,ud}$	undetected critical failure rate of component C, or of component choice a_{ij}, or of the system (FIT)
$\lambda_{X,uncrit}$	uncritical failure rate of component C, or of component choice a_{ij}, or of the system (FIT)
λ_{SPF}	SPF failure rate of the system (FIT)
λ_{SR}	safety related failure rate of the system (FIT)

Notations on the targets and constraints of the automotive RAP:

w_{ij}	weight of component a_{ij} (g)
g_{ij}	cost of component a_{ij} (EUR)
p_{ij}	packaging space need of component a_{ij} (cm^3)
W	system weight (g)
G	system cost (EUR)
P	system packaging space (cm^3)
W_{max}	system weight constraint (g)
G_{max}	system cost constraint (EUR)
P_{max}	system packaging space constraint (cm^3)
$SPFM$	SPFM of the system (%)
LFM	LFM of the system (%)
$PMHF$	PMHF of the system (FIT)
Q_{S_i}	failure probability of subsystem S_i (1)
$SPFM_{min}$	SPFM constraint (%)
LFM_{min}	LFM constraint (%)
$PMHF_{max}$	PMHF constraint (%)
F_{fit}	fitness function
z_S	function calculating the system level PMHF
z_{S_i}	function calculating the failure probability of subsystem S_i

The notations related to the component selection:

$A_i = \{a_{i1}, a_{i2}...a_{iq_i}\}$	set of available components in subsystem i
$A = \{A_i...A_m\}$	set of available components
q_i	number of available component variants in subsystem i
$\mathbf{Y} = [\mathbf{y}_1...\mathbf{y}_m]$	matrix representing the component choices of all subsystems
$\mathbf{y_i}$	vector representing the component choices in subsystem i
$\mathbf{b} = [b_1, ..., b_i, ...b_m]^T$	vector containing the number of the primary unit of subsystem i, in case of $k_i = 1$ (1)
$\mathbf{I}$	matrix representing an individual
Φ	component selection function
Ψ	failure model combination function
$\mathbf{V}$	virtual component allocation matrix
$\mathbf{Y}^*$	matrix representing the component choices of all subsystems including the indirectly chosen type 2 virtual components

The notations related to the EA:

$\mathbf{\Omega}$	3-dimensional matrix representing a population
N_Ω	population size (1)
ρ	number of the actual generation (1)
ρ_{max}	maximal number of the generations (1)
Θ	penalty function
π_{SPFM}	penalty factor related to the SPFM (EUR/%)
π_{LFM}	penalty factor related to the LFM (EUR/%)
π_{PMHF}	penalty factor related to the PMHF (EUR/FIT)
π_W	penalty factor related to the weight (EUR/g)
π_P	penalty factor related to the packaging space (EUR/cm^3)

5.2.3 Mathematical model to represent the architectures, objectives and constraints of an automotive RAP

Based on the three gaps identified at the end of section 5.2.1, this section is structured as follows:

1. In first step, the mathematical model of the failure-effect mapping is defined, to support the representation of the complex multi-state system dynamics of automotive E/E systems. This model mimics the FMEDA and FTA methods, to calculate the HW metrics of the [ISO11].
2. In the second step, the mathematical model of the component selection is defined, matching to the mathematical failure model.
3. After that, possible model simplifications in case of parallel-serial architectures are given.
4. In the final step, an outlook is given, which aspects need to be addressed in case of m-to-n redundancy allocation.

Problem statement

In order to develop an appropriate mathematical model, first the optimisation problem statement is defined as the following:

$$Min \quad G = \sum_{i=1}^{m} \sum_{j=1}^{n_i} g_{ij} \tag{5.7}$$

Subject to the following constraints:

$$SPFM \geq SPFM_{min} \tag{5.8}$$

$$LFM \geq LFM_{min} \tag{5.9}$$

$$PMHF < PMHF_{max} \tag{5.10}$$

$$W = \sum_{i=1}^{m} \sum_{j=1}^{n_i} w_{ij} \leq W_{max} \tag{5.11}$$

$$P = \sum_{i=1}^{m} \sum_{j=1}^{n_i} p_{ij} \leq P_{max} \tag{5.12}$$

The packaging space need is defined as the volume of the packaging space "consumed" by the element in focus. This includes the element's physical volume and the required clearance to adjacent elements. Therefore, it is expressed in cm^3.

$$n_i = \sum_{j=1}^{p_i} y_{ij} \leq 3 \tag{5.13}$$

Architectural modelling of automotive systems

As already noted in section 5.1.1, in most cases, the RAP is solved for a parallel-serial architecture. In that section, it is also noted, that this topology has to be understood in terms of the *logical* structure of the system, not the physical one. A parallel-serial architecture (as shown in figure 5.3 a)) is characterised by its fault propagation behaviour: the system works as long as all of its subsystems work, and each subsystem works as long as a sufficient number of components (in that subsystem) work. This is represented in the fault tree shown in figure 5.4. According to this, many physically complex automotive architectures can be translated into a serial logical architecture. These, in turn, can be extended into parallel-serial architectures by adding redundancy.

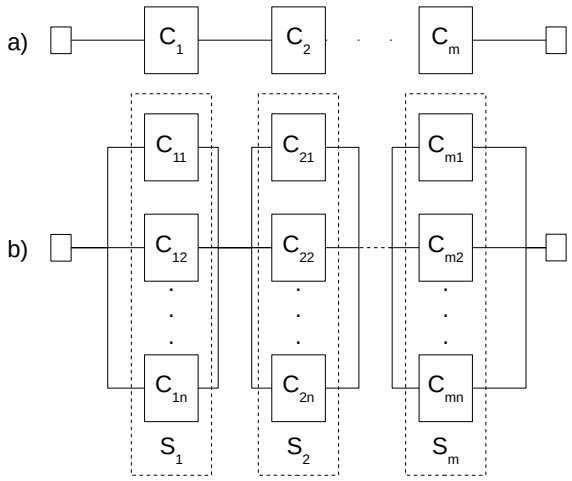


Figure 5.3: Serial and parallel-serial architectures

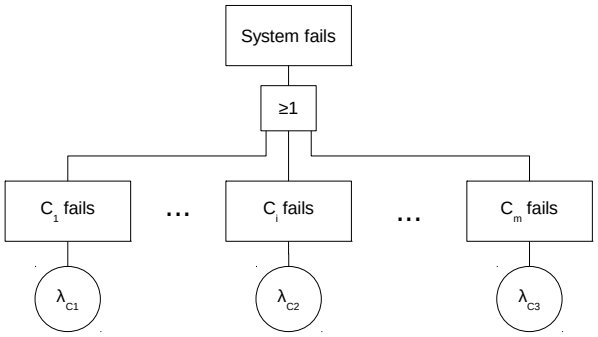


Figure 5.4: Fault tree - Serial architecture

Figure 5.5 shows a complex physical architecture, that cannot be transformed into a parallel-serial logical architecture. This can be the case, if one of the following conditions is fulfilled:

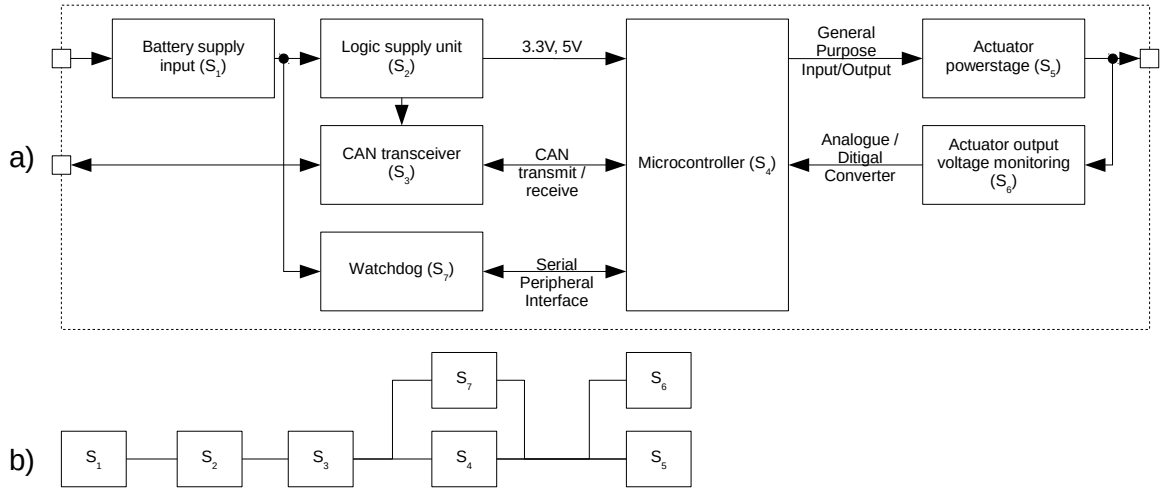


Figure 5.5: Complex automotive ECU architecture - a) physical architecture, b) logical architecture

- There is at least one subsystem, that has safe faults only.
- There is at least one subsystem, that has MPF-s. Note, that MPF-s in this case need to be understood as MPF-s, if only one component is included in a subsystem.

The former case is obvious. For the latter, the following two instances need to be distinguished:

- MPF-s of the components that have SPF-s as well. E.g. the microcontroller may fail in a way that directly leads to a system failure (i.e. SPF), but may have other failure modes, that are only critical in combination with another failure mode (i.e. MPF).
- If there are subsystems of the architecture that have no SPF-s, but MPF-s only. E.g. if the watchdog (S_7) fails to detect the failure of the microcontroller (S_4), it is not critical, until the microcontroller itself fails. The actuator output voltage monitoring (S_6) serves only diagnostic purposes, therefore its stuck-at failures are only critical in combination with failures of the actuator powerstage (S_5). These are typically subsystems, that are involved in safety mechanisms.

Common in both the serial and the complex architectures is, that each system consists of m subsystems (S_i, where $i \in \mathbb{N}$ and $0 < i \leq m$), which consist of $\mathbf{n}(i)$ components (C_{ij}, where $j \in \mathbb{N}$ and $0 < j \leq \mathbf{n}(i)$).

In the next sections it will be presented, how the automotive RAP can be solved for parallel-serial -, and for complex architectures. The former will be derived from the latter in section 5.2.3, by applying model simplifications. The referred section will also give guidance, how some complex architectures can be transformed into parallel-serial topologies, for the sake of simplifying the RAP.

Mathematical model for calculating SPFM, LFM and PMHF

Since the goal is to use the SPFM, LFM and PMHF as constraints in the automotive RAP, the key is to develop a model of the system, that can support their calculation. In the practice, this is performed by an FMEDA and a quantitative FTA (like for example in section 4.4.3). The mathematical model developed in this section will therefore mimic these analyses by providing a dynamic model based on linear algebra. Since this is a model, it is important to clearly define its assumptions and boundaries:

- Each subsystem S_i is regarded as consisting of $\mathbf{n}(i)$ components. These components are either in a dynamic or static redundant architecture. In section 4.4.3, these two architectures have been identified as most promising for automotive applications. In case of dynamic redundancy, the subsystem includes the primary unit C_{i1} and $\mathbf{n}(i) - 1$ spare ones, in a hot standby architecture. In case of a static redundant architecture, a k-out-of-n (k-o-o-n) topology is assumed, where $\boldsymbol{\kappa}(i)$

components need to work to provide the required function. Note, that in the mathematical model, $\kappa(i) = 1$ and $\mathbf{n}(i) > 1$ represent the dynamic redundant architecture.
- Each subsystem can contain each component type only once, due to the diverse redundancy requirement.
- The components in a subsystem are regarded as functionally equal, but may have different failure modes.
- The failure modes of each component are disjoint. Therefore, the sum of their failure rates is equal to the safety related failure rate of the component.
- Failures of the individual components are *independent*. As already mentioned in section 5.1, this is a common assumption. See section 5.2.3 for a model extension, where this assumption is probably no longer valid.
- Exponential cumulative distribution is used, with the following simplification: $Q = 1 - e^{-\lambda t} \approx \lambda t$. This is a commonly applied simplification, also used by the [ISO11] [part 10].
- Imperfect switching is represented by DC values. Failure modes and failure rates of arbitration logic and voters are neglected or included in the DC value. As shown in section 4.4.3, this is a plausible assumption. This premise is also applied by [Kno12], [Mis99] and [Sup04].

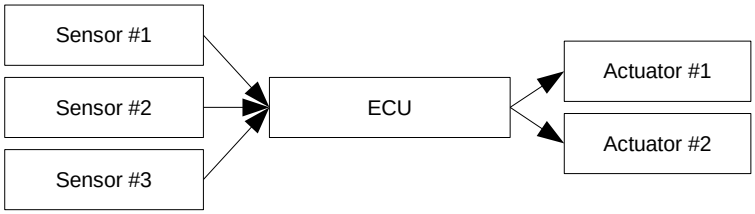


Figure 5.6: Example for a parallel-serial system

To show how this model is constructed, a simple example model is used, that shows the architectural variants, that the model has to deal with:

- Subsystems without redundancy (the ECU in figure 5.6).
- Subsystems with redundancy and a k-o-o-n architecture (the actuator in figure 5.6, using a classic dynamic redundant architecture).
- Subsystems with redundancy and a TMR architecture (the sensor in figure 5.6). Note that the sensor subsystem consisting of three sensors could be also arranged in a dynamic redundant architecture. In that case, Sensor #2 and Sensor #3 would be the two backups. The decision to use a TMR is completely arbitrary for the example.

The system level failure model is kept as simple as possible, consisting of the following three vectors. The failure mode vector of the system:

$$\mathbf{f}_S = \begin{bmatrix} \mathbf{f}_S(1) & \mathbf{f}_S(2) & ... & \mathbf{f}_S(s_s) \end{bmatrix}^T \tag{5.14}$$

The failure rate vector of the system:

$$\boldsymbol{\lambda}_S = \begin{bmatrix} \boldsymbol{\lambda}_S(1) & \boldsymbol{\lambda}_S(2) & ... & \boldsymbol{\lambda}_S(s_s) \end{bmatrix}^T \tag{5.15}$$

where $\boldsymbol{\lambda}_S(i)$ is the corresponding failure rate of failure mode $\mathbf{f}_S(i)$, and $i = 1, ..., s_s$.

The elements of the criticality vector on the system level:

$$\gamma_S(i) = \begin{cases} 0 & \text{if } \mathbf{f}_S(i) \text{ is not safety critical,} \\ 1 & \text{if } \mathbf{f}_S(i) \text{ is safety critical.} \end{cases} \tag{5.16}$$

where $i = 1, ..., s_s$. It has to be noted here, that as already shown in section 2.4.4, the HW metrics of the [ISO11] are calculated for a particular safety goal. The vector γ_S may contain from 0 up to s_s 1-s.

Depending on the level of the analysis, these critical failure modes may represent different numbers of safety goals. The failure model developed in this section does not differentiate between these safety goals. If the evaluation needs to be performed for various safety goals separately, separate γ_S vectors need to be defined for each one of them. The rest of the model may remain the same.

Besides the failure-related information, the following vectors describe crucial architectural properties of the system:

- The vector $\mathbf{n}$ defines the number of components in each subsystem. This vector would be $\mathbf{n} = [3; 1; 2]$ in the example system shown in figure 5.6.
- The vector $\boldsymbol{\kappa}$ defines the k value of a k-o-o-n architecture, in each subsystem. This vector would be $\boldsymbol{\kappa} = [2; 1; 1]$ in the example system shown in figure 5.6.

The *components* are modelled using the following mathematical representation. For each component C_{ij} (meaning the j-th component of the i-th subsystem), a failure modes and related failure rates are defined by the vectors $\mathbf{f}_{C_{ij}}$ and $\boldsymbol{\lambda}_{C_{ij}}$, like on the system level.

After having modelled the failure modes and - rates, the failure propagation shall be represented. As shown in figure 2.9, the [ISO11] distinguishes the following:

- Failures that can lead to a safety goal violation if no safety mechanism is present, on their own.
- Failures that can lead to a safety goal violation in combination with another, independent failure.

This is represented by so called FEMM-s, mapping failure modes of component C_i to their effects on the system level. A FEMM on the *actual components* either represents the SPF -, or the MPF mapping. The elements of these matrices are determined as follows:

$$\mathbf{M}^{S}_{C_{ij}\to S}(a,b) = \begin{cases} 0 & \text{if } \mathbf{f}_{C_{ij}}(b) \text{ does not lead to } \mathbf{f}_S(a) \text{ in the absence of a safety mechanism,} \\ 1 & \text{if } \mathbf{f}_{C_{ij}}(b) \text{ leads to } \mathbf{f}_S(a) \text{ in the absence of a safety mechanism.} \end{cases} \tag{5.17}$$

$$\mathbf{M}^{M}_{C_{ij}\to S}(a,b) = \begin{cases} 0 & \text{if } \mathbf{f}_{C_{ij}}(b) \text{ does not lead to } \mathbf{f}_S(a) \text{ in combination with another, independent fault,} \\ 1 & \text{if } \mathbf{f}_{C_{ij}}(b) \text{ leads to } \mathbf{f}_S(a) \text{ in combination with another, independent fault.} \end{cases} \tag{5.18}$$

where $a = 1, ..., s_s$ and $b = 1, ..., s_i$ denote the indices of the matrix.

For the components, both the $\mathbf{M}^{S}_{C_{ij}\to S}$ and the $\mathbf{M}^{M}_{C_{ij}\to S}$ are to be interpreted in the particular architecture of the S_i. Note, that the interpretation of the FEMM of *component choices* used in the RAP later differs from this, as explained in section 5.2.3.

The meaning of matrix $\mathbf{M}^{S}_{C_{ij}\to S}$ is explained by figure 5.7, showing an excerpt of an FMEA failure net (see section 2.5.1). As it can be seen, the linkage between failure modes and their effects is an m-to-n connection. Any failure mode may have various effects, and each effect can be caused by several failure modes. The interpretation of matrix $\mathbf{M}^{M}_{C_{ij}\to S}$ is similar, but for MPF-s only.

Based on the FEMM-s, the component level failure mode criticality vectors (considering either SPF-s or MPF-s) can be derived to simplify further calculations:

$$\gamma^{S}_{C_{ij}} = min\left(\mathbf{u}, \mathbf{M}^{ST}_{C_i\to S} \cdot \gamma_S\right) \tag{5.19}$$

$$\gamma^{M}_{C_{ij}} = min\left(\mathbf{u}, \mathbf{M}^{MT}_{C_i\to S} \cdot \gamma_S\right) \tag{5.20}$$

where $\mathbf{u} = \begin{pmatrix} 1 & 1 & ... & 1 \end{pmatrix}$ represents a vector of the appropriate length, where each value is 1. Note, that the minimum function is necessary to cope with cases, where the vector γ_S contains more than one 1-s, i.e. where more than one system level failure mode is safety critical.

After having the component level failure modes mapped to the system level ones, and having determined the criticality of the former ones for SPF and MPF cases, the DC shall be modelled. The DC is represented by two vectors, one for the SPF, one for the MPF evaluation:

- The vector $\delta^{S}_{C_{ij}}$ includes the DC values for the failure modes of the vector $\mathbf{f}_{C_{ij}}$, with regards to SPF-s. This DC shows the detected percentage of the potentially critical component level faults, being therefore controlled, not leading to a safety goal violation.

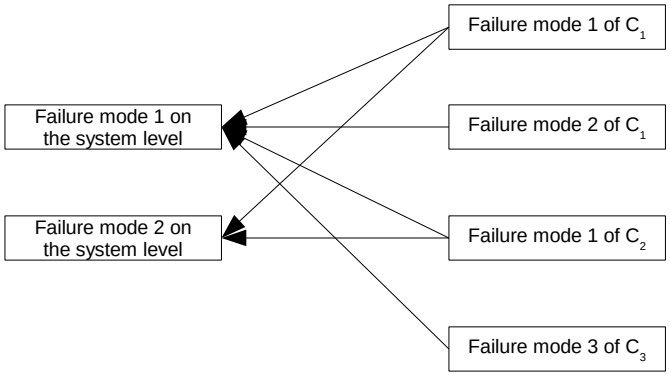


Figure 5.7: Explanation of the FEMM - Possible failure net of the example serial architecture

- The vector $\delta^M_{C_{ij}}$ includes the failure mode coverage values for the failure modes of the vector $\mathbf{f}_{C_{ij}}$, with regards to MPF-s. This DC shows the detected percentage of the potentially latent component level faults.

Note, that in accordance with the [ISO11] [part 5], these two vectors can be different.

Based on these, the failure rate values required for the calculation of the SPFM and LFM can be calculated using the following equations.

SPF/RF failure rate vector and scalar values:

$$\boldsymbol{\lambda}^{SPF/RF}_{C_{ij}} = \boldsymbol{\gamma}^{ST}_{C_{ij}} \circ \left(\boldsymbol{\lambda}_{C_{ij}} \circ \left(\mathbf{u} - \boldsymbol{\delta}^{S}_{C_{ij}}\right)\right) \tag{5.21}$$

$$\lambda^{SPF/RF}_{C_{ij}} = \boldsymbol{\gamma}^{ST}_{C_{ij}} \cdot \left(\boldsymbol{\lambda}_{C_{ij}} \circ \left(\mathbf{u} - \boldsymbol{\delta}^{S}_{C_{ij}}\right)\right) \tag{5.22}$$

MPF,L failure rate scalar value:

$$\lambda^{MPF,L}_{C_{ij}} = \boldsymbol{\gamma}^{M}_{C_{ij}}{}^{T} \cdot \left(\left(\boldsymbol{\lambda}_{C_{ij}} - \boldsymbol{\lambda}^{SPF/RF}_{C_{ij}}\right) \circ \left(\mathbf{u} - \boldsymbol{\delta}^{M}_{C_{ij}}\right)\right) \tag{5.23}$$

Based on these, the SPFM and the LFM can be calculated, using equations 2.1 and 2.2, with the following input values:

$$\lambda_{SR} = \sum_{i=1}^{m} \sum_{j=1}^{n_i} \sum_{k=1}^{s_{C_{ij}}} \boldsymbol{\lambda}_{C_{ij}}(k) \tag{5.24}$$

$$\lambda_{SPF/RF} = \sum_{i=1}^{m} \sum_{j=1}^{n_i} \lambda^{SPF/RF}_{C_{ij}} \tag{5.25}$$

$$\lambda_{MPF,L} = \sum_{i=1}^{m} \sum_{j=1}^{n_i} \lambda^{MPF,L}_{C_{ij}} \tag{5.26}$$

Figure 5.8 shows the exemplary FMEDA for the example system of figure 5.6, while figure 5.9 represents the mapping of the vectors and matrices of the model to the FMEDA table.

The PMHF calculation can be performed using a quantitative FTA, as shown in section 4.4.3. A generic formula, representing this FTA can be specified as follows:

$$PMHF = z_S\left(Q_{S_1}, Q_{S_2}, ..., Q_{S_m}, t\right) \tag{5.27}$$

Note, that (as also stated in [Win86], [Kuooo], [Yum93] and [Red97]) for complex architectures, an architecture-specific function is necessary. In section 5.2.3 it will be shown, that this generic function can be specialised for parallel-serial architectures.

					SPF evaluation					MPF evaluation				
					1	0	1			1	0	1		
Arch.	Component (C_{ij})	Failure mode (f_{Cij})		Failure rate [FIT] (λ_{Cij})	δ_{Cij}	δ_{Cij}	δ_{Cij}	DC_{SPF} (δ_{Sij})	$\lambda_{SPF/RF}$	δ_{Cij}	δ_{Cij}	δ_{Cij}	DC_{MPF}	$\lambda_{MPF,L}$
TMR	Sensor #1 (C_{11})	Sensor #1 signal too high	f_C11,1	2	1	1	0	99%	0,02	1	1	0	100%	0
		Sensor #1 signal too low	f_C11,2	2	0	0	1	99%	0,02	0	0	1	100%	0
		Sensor #1 signal oscillates	f_C11,3	4	0	0	0	99%	0	0	0	0	100%	0
	Sensor #2 (C_{12})	Sensor #2 signal too high	f_C12,1	4	1	1	0	99%	0,04	1	1	0	100%	0
		Sensor #2 signal too low	f_C12,2	4	0	0	1	99%	0,04	0	0	1	100%	0
		Sensor #2 signal stuck at max	f_C12,3	2	1	1	0	99%	0,02	1	1	0	100%	0
		Sensor #2 signal stuck at min	f_C12,4	2	0	0	1	99%	0,02	0	0	1	100%	0
	Sensor #3 (C_{13})	Sensor #3 signal too high	f_C13,1	9	0	0	0	99%	0	0	0	0	100%	0
		Sensor #3 signal too low	f_C13,2	9	1	1	0	99%	0,09	1	1	0	100%	0
Single	ECU (C_{21})	Read sensor #1 signal too high	f_C21,1	10	1	1	0	0%	10	0	0	0	0%	0
		Read sensor #1 signal too low	f_C21,2	10	0	0	1	0%	10	0	0	0	0%	0
		Read sensor #2 signal too high	f_C21,3	8	1	1	0	0%	8	0	0	0	0%	0
		Read sensor #2 signal too low	f_C21,4	8	0	0	1	0%	8	0	0	0	0%	0
		Read sensor #3 signal too high	f_C21,5	15	0	0	0	0%	0	0	0	0	0%	0
		Read sensor #3 signal too low	f_C21,6	15	1	1	0	0%	15	0	0	0	0%	0
		Internal computation failure - safe	f_C21,7	250	0	0	0	0%	0	0	0	0	0%	0
		Internal computation failure - dang.	f_C21,8	250	1	1	1	0%	250	0	0	0	0%	0
		Actuator #1 control failure - active	f_C21,9	110	1	1	0	0%	110	0	0	0	0%	0
		Actuator #1 control failure - passive	f_C21,10	110	0	0	0	0%	0	0	0	0	0%	0
		Actuator #2 control failure - active	f_C21,11	80	1	1	0	0%	80	0	0	0	0%	0
		Actuator #2 control failure - passive	f_C21,12	80	0	0	0	0%	0	0	0	0	0%	0
Dyn. red. /w hot stby	Actuator #1 (C_{31})	Actuator #1 fails - active	f_C31,1	20	1	1	0	90%	2	0	0	0	100%	0
		Actuator #1 fails - passive	f_C31,2	20	0	0	0	90%	0	0	0	0	100%	0
	Actuator #2 (C_{32})	Actuator #2 fails - active	f_C32,1	30	0	0	0	90%	0	1	1	0	90%	3
		Actuator #2 fails - passive	f_C32,2	30	0	0	0	90%	0	0	0	0	90%	0
		Actuator #2 fails - intermittent	f_C32,3	10	0	0	0	90%	0	1	1	0	90%	1
				$\Sigma\lambda_{SR}$=	1094			$\Sigma\lambda_{SPF/RF}$=	493,25				$\Sigma\lambda_{MPF,L}$=	4,00

Figure 5.8: Example FMEDA for the failure modelling example system

The subsystem failure probability, in turn, can be calculated using the following formula:

$$Q_{S_i} = z_{S_i}\left(\boldsymbol{\lambda}_{Ci1},...,\boldsymbol{\lambda}_{Cin_i},\boldsymbol{\delta}^S_{Ci1},...,\boldsymbol{\delta}^S_{Cin_i},\boldsymbol{\delta}^M_{Ci1},...,\boldsymbol{\delta}^M_{Cin_i},\boldsymbol{\gamma}^S_{Ci1},...,\boldsymbol{\gamma}^S_{Cin_i},\boldsymbol{\gamma}^M_{Ci1},...,\boldsymbol{\gamma}^M_{Cin_i},\mathbf{n},\boldsymbol{\kappa},t\right) \tag{5.28}$$

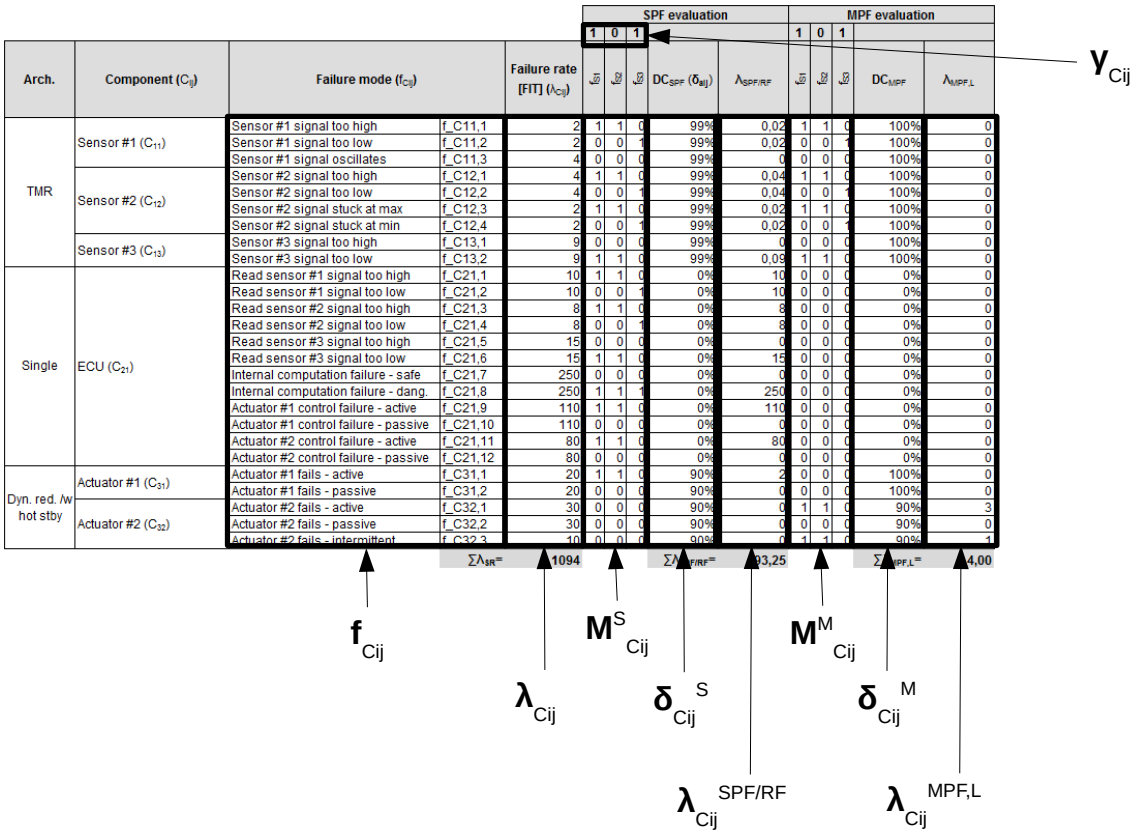
The rationale behind this equation is that the subsystem level failure probability depends on the following:

- Failure rates of the component level failure modes ($\boldsymbol{\lambda}_{Cij}$)
- DC related to SPF-s and MPF-s ($\boldsymbol{\delta}^S_{Cij}$ and $\boldsymbol{\delta}^M_{Cij}$)
- Criticality of the component level failure modes, as SPF-s and MPF-s ($\boldsymbol{\gamma}^S_{Cij}$ and $\boldsymbol{\gamma}^M_{Cij}$)
- Architecture of subsystem S_i ($\mathbf{n}$ and $\boldsymbol{\kappa}$)
- System operational time (t)

At this point, the principle of the functions z_S and z_{S_i} shall be investigated more in detail. [Win86], [Kuooo], [Yum93] and [Red97] give very thorough definitions and mathematical models for complex architectures. But the following aspects of the calculation method applied by all four papers needs caution:

- The reliability of the subsystems and the components is simply represented by the classical representation of reliability: mathematical probability.
- The probability that two parallel subsystems fail is represented by the product of the two subsystems' probabilities.

For automotive systems, using the PMHF as one of the reliability metric, the calculation is more complicated. The FTA, that needs to be mimicked by the mathematical model, shall distinguish between certain failure modes of the subsystems, instead of using one single reliability metric. In the example shown in figure 5.5 b), the subsystems S_4 (the microcontroller) and S_7 (the watchdog) may appear as two parallel subsystems, but their reliability still cannot be calculated by simply multiplying Q_{S_4} and Q_{S_7}. Figure 5.10 shows an example, why. As it can be seen, the loss of the function of the ECU may be caused by a very specific combination of failures of S_4 and S_7. Note, that this is only an excerpt



Arch.	Component (C_{ij})	Failure mode (f_{Cij})		Failure rate [FIT] (λ_{Cij})	SPF eval.			DC_{SPF} (δ_{Sij})	$\lambda_{SPF/RF}$	MPF eval.			DC_{MPF}	$\lambda_{MPF,L}$
					1	0	1			1	0	1		
TMR	Sensor #1 (C_{11})	Sensor #1 signal too high	f_C11,1	2	1	1	0	99%	0,02	1	1	0	100%	0
		Sensor #1 signal too low	f_C11,2	2	0	0	1	99%	0,02	0	0	1	100%	0
		Sensor #1 signal oscillates	f_C11,3	4	0	0	0	99%	0	0	0	0	100%	0
	Sensor #2 (C_{12})	Sensor #2 signal too high	f_C12,1	4	1	1	0	99%	0,04	1	1	0	100%	0
		Sensor #2 signal too low	f_C12,2	4	0	0	1	99%	0,04	0	0	1	100%	0
		Sensor #2 signal stuck at max	f_C12,3	2	1	1	0	99%	0,02	1	1	0	100%	0
		Sensor #2 signal stuck at min	f_C12,4	2	0	0	1	99%	0,02	0	0	1	100%	0
	Sensor #3 (C_{13})	Sensor #3 signal too high	f_C13,1	9	0	0	0	99%	0	0	0	0	100%	0
		Sensor #3 signal too low	f_C13,2	9	1	1	0	99%	0,09	1	1	0	100%	0
Single	ECU (C_{21})	Read sensor #1 signal too high	f_C21,1	10	1	1	0	0%	10	0	0	0	0%	0
		Read sensor #1 signal too low	f_C21,2	10	0	0	1	0%	10	0	0	0	0%	0
		Read sensor #2 signal too high	f_C21,3	8	1	1	0	0%	8	0	0	0	0%	0
		Read sensor #2 signal too low	f_C21,4	8	0	0	1	0%	8	0	0	0	0%	0
		Read sensor #3 signal too high	f_C21,5	15	0	0	0	0%	0	0	0	0	0%	0
		Read sensor #3 signal too low	f_C21,6	15	1	1	0	0%	15	0	0	0	0%	0
		Internal computation failure - safe	f_C21,7	250	0	0	0	0%	0	0	0	0	0%	0
		Internal computation failure - dang.	f_C21,8	250	1	1	1	0%	250	0	0	0	0%	0
		Actuator #1 control failure - active	f_C21,9	110	1	1	0	0%	110	0	0	0	0%	0
		Actuator #1 control failure - passive	f_C21,10	110	0	0	0	0%	0	0	0	0	0%	0
		Actuator #2 control failure - active	f_C21,11	80	1	1	0	0%	80	0	0	0	0%	0
		Actuator #2 control failure - passive	f_C21,12	80	0	0	0	0%	0	0	0	0	0%	0
Dyn. red. /w hot stby	Actuator #1 (C_{31})	Actuator #1 fails - active	f_C31,1	20	1	1	0	90%	2	0	0	0	100%	0
		Actuator #1 fails - passive	f_C31,2	20	0	0	0	90%	0	0	0	0	100%	0
	Actuator #2 (C_{32})	Actuator #2 fails - active	f_C32,1	30	0	0	0	90%	0	1	1	0	90%	3
		Actuator #2 fails - passive	f_C32,2	30	0	0	0	90%	0	0	0	0	90%	0
		Actuator #2 fails - intermittent	f_C32,3	10	0	0	0	90%	0	1	1	0	90%	1
				$\Sigma\lambda_{SR}$= 1094				$\Sigma\lambda_{SPF/RF}$= 93,25					$\Sigma\lambda_{MPF,L}$= 4,00	

Figure 5.9: Mapping of the failure model to an FMEDA table

from the probably very complicated fault tree of the example system. But even in this excerpt, it can be seen, that for certain complex architectures, the calculation of the system level PMHF needs detailed information about the subsystem failure combinations. But since the RAP involves a significantly high number of possible combinations, this may be very inefficient or even impossible. For complex systems, therefore, the PMHF is not always applicable.

To cope with this issue, the following solution alternatives shall be taken into consideration:

- Do not use the PMHF as optimisation criteria. In that case, only the SPFM and the LFM are applied, since the calculation of those is still possible for complex architectures, using the developed model.
- Calculate the PMHF of a complex system to its *parallel-serial equivalent*. This solution is explained in section 5.2.3.
- Use the same failure modes for each component choice of a subsystem. Since all component choices have similar functions, their malfunctions are also similar. Therefore, for some cases, this solution is easily applicable. This simplification allows the definition of the application specific z_S and z_{S_i} functions.

Mathematical modelling of the component selection

In order to support the solution of the RAP, the selection of components from the pool of component choices shall be modelled. Figure 5.11 shows its principle, based on the example shown in figure 5.6. As it can be seen in the referenced figure, the principles behind the component selection are the following:

- The component choices are represented by the set $A = \{A_i ... A_m\}$, consisting of the component choices for each subsystem S_i: $A_i = \{a_{i1}, a_{i2} ... a_{iq_i}\}$.

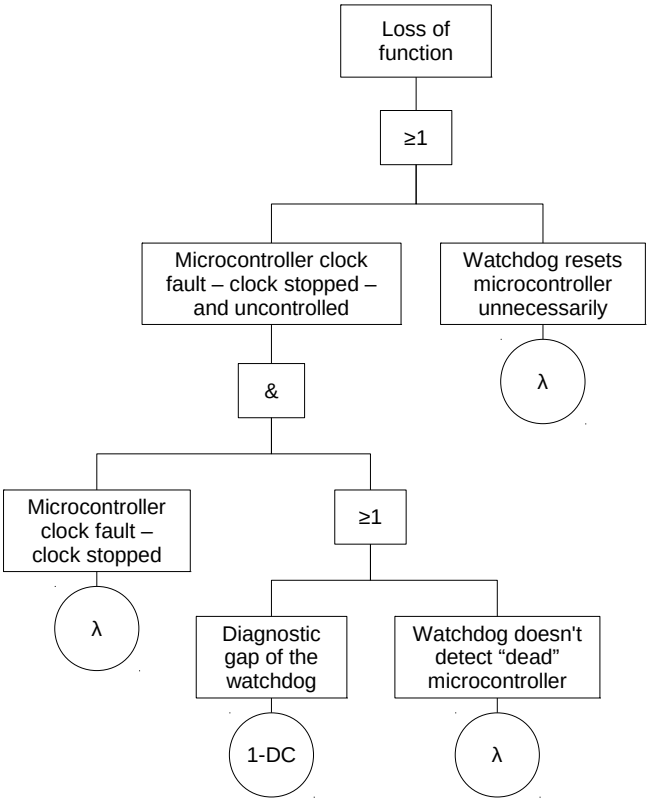


Figure 5.10: Excerpt from a fault tree of a complex system

- The actually chosen components are represented by C_{ij}, meaning the j-th component in S_i.
- The component selection is represented by the function Φ.

The component selection function Φ has multiple purpose. First of all, the components are *selected* by the values $\mathbf{Y}(i,j)$, showing if component a_{ij} is included in subsystem S_i:

$$\mathbf{Y}(i,j) = \begin{cases} 0 & \text{if } a_{ij} \text{ is not included,} \\ 1 & \text{if } a_{ij} \text{ is not included,} \\ -1 & \text{if } a_{ij} \text{ is not available.} \end{cases} \tag{5.29}$$

The intention of the latter value is to yield the matrix $\mathbf{Y}$, filling up the empty component choices with -1 values. In addition to that, there are two additional decisions related to the component selection that need to be modelled:

- *Architecture selection* in case of three components in a subsystem (represented by the vector $\boldsymbol{\kappa}$).
- *selection of the primary component*, in case of a dynamic redundant system (represented by the vector $\mathbf{b}$).

In the example shown in figure 5.11, the component selection is as follows:

- For S_1, the component choices A_{11}, A_{12} and A_{14} have been selected. A TMR architecture is chosen, as represented by $\boldsymbol{\kappa}(1) = 2$. Therefore, the primary unit is not applicable, $\mathbf{b}(1)$ is set to 1 per default.
- For S_2, the component choice A_{22} has been selected. There are no architectural choices available, $\boldsymbol{\kappa}(2)$ is set to 1 per default. Due to the same reason, the primary unit is not applicable, $\mathbf{b}(1)$ is set to 1 per default.

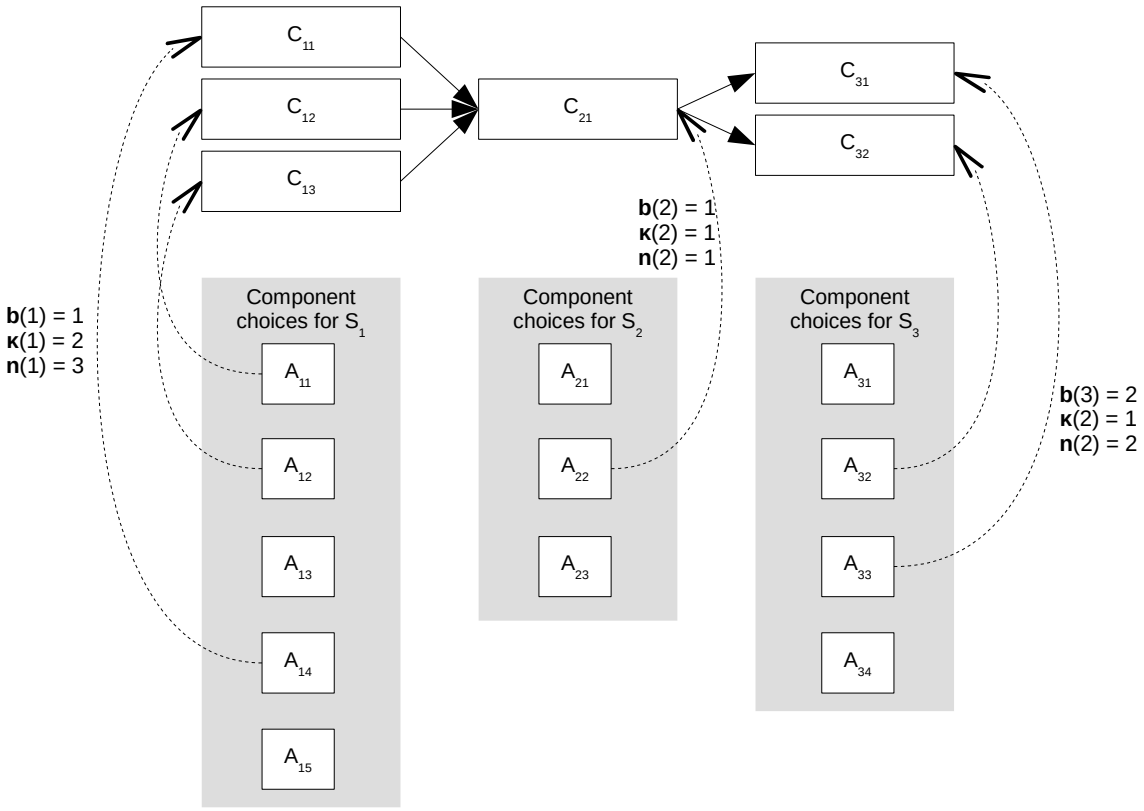


Figure 5.11: Graphical explanation of the component selection function

- For S_3, the component choices A_{32} and A_{33} have been selected. There are no architectural choices available, $\boldsymbol{\kappa}(3)$ is set to 1 per default. $\mathbf{b}(3) = 2$ represents that the second component choice (i.e. A_{33}) is meant to be the primary component in the dynamic redundant architecture.

Based on the procedure described above, the component selection is represented by the matrix $\mathbf{I}$:

$$\boldsymbol{I} = \begin{bmatrix} \mathbf{Y} & \boldsymbol{\kappa} & \mathbf{b} \end{bmatrix} = \begin{bmatrix} y_{11} & y_{12} & \dots & y_{1q^{max}} & \boldsymbol{\kappa}(1) & \mathbf{b}(1) \\ y_{21} & y_{22} & \dots & y_{2q^{max}} & \boldsymbol{\kappa}(2) & \mathbf{b}(2) \\ \dots & \dots & \dots & \dots & \dots & \dots \\ y_{m1} & y_{m2} & \dots & y_{mq^{max}} & \boldsymbol{\kappa}(m) & \mathbf{b}(m) \end{bmatrix} \tag{5.30}$$

where q^{max} represents the highest value in $\mathbf{q}$, i.e. the maximal number of available component choices in any subsystem.

Note, that the component selection is more than just the definition of the matrix $\mathbf{I}$, therefore the function Φ is introduced:

$$\Phi : \{A, \mathbf{I}\} \rightarrow \{C, \mathbf{n}, \boldsymbol{\kappa}\} \tag{5.31}$$

The rationale behind this function is the following:

- From the set of available components A and component selection matrix Y, the components C_{ij} can be generated.
- From the vector $\mathbf{b}$, the components C_{i1} can be allocated, representing the primary units in case of a dynamic redundant architecture. This way, the equations presented in section 5.2.3 can be applied without any adaptation.
- The vector $\mathbf{n}$ can be generated based on the number of selected components for each subsystem.

- The vector $\boldsymbol{\kappa}$ is simply inherited from the component selection definition to the actual subsystem representation.

Related to the matrix, vector and scalar values representing the failure model (introduced in section 5.2.3), the function Φ has to perform an appropriate mapping depending on the matrix $\mathbf{Y}$. This mapping is necessary, since the interpretation of the entries representing the failure model of the component choices (a_{ij}) has to be partially different. These differences are explained in the following paragraphs.

The interpretation of the failure rate vector $\boldsymbol{\lambda}_{a_{ij}}$ is identical to those of the chosen components.

The interpretation of the FEMM-s needs further considerations. In case of the chosen components (C_{ij}), the related matrices consider the actual subsystem architecture. In case of a component choice (a_{ij}), this information is not available yet. Therefore, the FEMM-s need to be treated differently:

- $\mathbf{M}^{S}_{a_{ij}\rightarrow S}$: shows if a certain failure mode of the component choice a_{ij} leads to a certain system level failure mode, on its own, assuming, that a_{ij} is the only component in subsystem S_i.
- $\mathbf{M}^{M}_{a_{ij}\rightarrow S}$: shows if a certain failure mode of the component choice a_{ij}, in combination with another independent failure, leads to a certain system level failure mode, assuming, that a_{ij} is the only component in subsystem S_i, and that $DC_{SPF/RF} = 0$. The latter assumption is necessary to ensure, that each failure mode can lead either on its own to a certain system level failure, or only in combination with another independent failure, or not at all.

Based on the FEMM-s and the vector $\boldsymbol{\gamma}_S$, the component choice criticality vectors $\boldsymbol{\gamma}^{S}_{a_{ij}}$ and $\boldsymbol{\gamma}^{M}_{a_{ij}}$ can be calculated, using the same equations as for the chosen components C_{ij}. These vectors have an important role in the component selection modelling, which will be presented later.

At this point, it is important to understand, which types of MPF can be in a redundant subsystem:

- MPF-s that would be MPF-s even if $n_i = 1$ (i.e. $\mathbf{M}^{S}_{a_{ij}}(a,b) = 0$ and $\mathbf{M}^{M}_{a_{ij}}(a,b) = 1$). For instance, the failure of a self-diagnostic function of a sensor would be such. Note, that by definition, these kind of MPF-s are non-existent in case of a parallel-serial architecture.
- MPF-s that would be SPF-s if $n_i = 1$, but become MPF-s due to the redundant subsystem architecture (i.e. $\mathbf{M}^{S}_{a_{ij}}(a,b) = 1$ and $\mathbf{M}^{M}_{a_{ij}}(a,b) = 0$). In the example shown in figure 5.6, the unavailability of the secondary (i.e. backup) actuator would be such.

This distinction is essential to understand how the DC information related to a component choice a_{ij} is modelled and interpreted. Instead of two vectors (as in case of the chosen components), a matrix and a vector are necessary:

- $\boldsymbol{\Delta}^{S}_{a_{ij}} = \left[\boldsymbol{\delta}^{S,\mathbf{n}(i)=1}_{a_{ij}} \boldsymbol{\delta}^{S,\mathbf{n}(i)>1,\boldsymbol{\kappa}(i)=1}_{a_{ij}} \boldsymbol{\delta}^{S,\mathbf{n}(i)>1,\boldsymbol{\kappa}(i)=2}_{a_{ij}}\right]$ defines the DC, with regards to single point faults, assuming that the component choice a_{ij} is included in subsystem S_i alone, or in a dynamic redundant -, or in a TMR architecture, respectively. This represents a major deviation to the representation of the diagnostic coverage of the chosen components. The reason is, that the diagnostic coverage with regards to single point faults may be significantly affected by the subsystem architecture. For instance, in case of the fail-operational safety goal, the $\boldsymbol{\delta}^{S,\mathbf{n}(i)=1}_{a_{ij}}$ may consist of 0-s only, since there is no redundant component that could step in, in case of a failure. In contrast, the same failure mode may be covered to 90% if being included in a dynamic redundant architecture (i.e. $\mathbf{n}(i) > 1$, $\boldsymbol{\kappa}(i) = 1$).
- $\boldsymbol{\delta}^{M}_{a_{ij}}$ defines the DC with regards to multiple point faults. Here, no separate architecture-dependent sub-vectors are incorporated. In case of primary MPF-s, which are MPF-s even in case of $\mathbf{n}(i) = 1$, the related vector value represents the failure mode coverage by diagnostic mechanisms. In case of SPF-s which become MPF-s in the process of adding redundancy, the related vector value represents the percentage of the failures reported to, or perceived by, the driver. Therefore, no architecture-dependent factors need to be modelled.

Due to these failure modelling differences, a component selection function Φ (see equation 5.31) is necessary, to perform the mapping between the failure model of the component choices a_{ij} and the selected components C_{ij}. The mapping has to be based on the subsystem architecture, distinguishing the following cases:

- Only one component is selected in a subsystem (i. e. $\mathbf{n}(i) = 1$)
- Dynamic redundant architecture (i.e. $\mathbf{n}(i) > 1$, $\boldsymbol{\kappa}(i) = 1$)
- TMR topology (i.e. $\mathbf{n}(i) = 3$, $\boldsymbol{\kappa}(i) = 2$)

As it can be seen from this list above, the factors that need to be considered are $\mathbf{n}(i)$, $\boldsymbol{\kappa}(i)$ and $\mathbf{b}(i)$. The first two show the architecture, the latter the number of the primary unit. All these parameters help to decide how the SPF- and MPF-related $\boldsymbol{\delta}$ vectors and FEMM-s of C_{ij} shall be determined.

In case of one single component included in a subsystem (i.e. $\mathbf{n}(i) = 1$), the mapping can be represented by the following equations:

$$\boldsymbol{\delta}^{S}_{C_{i1}} = \boldsymbol{\delta}^{S,\mathbf{n}(i)=1}_{a_{i^*j^*}} \qquad \boldsymbol{\delta}^{M}_{C_{i1}} = \boldsymbol{\delta}^{M}_{a_{i^*j^*}} \tag{5.32}$$

$$\mathbf{M}^{S}_{C_{i1}\to S} = \mathbf{M}^{S}_{a_{i^*j^*}\to S} \qquad \mathbf{M}^{M}_{C_{i1}\to S} = \mathbf{M}^{M}_{a_{i^*j^*}\to S} \tag{5.33}$$

The rationale for these equations is, that since only one component is included in the subsystem S_i, the SPF-s and MPF-s of $a_{i^*j^*}$ are mapped to the SPF-s and MPF-s of C_{ij}, respectively.

In case of a dynamic redundant architecture (i.e. $\mathbf{n}(i) > 1$, $\boldsymbol{\kappa}(i) = 1$), the mapping can be represented by the following equations:

$$\boldsymbol{\delta}^{S}_{C_{i1}} = \boldsymbol{\delta}^{S,\mathbf{n}(i)>1,\boldsymbol{\kappa}(i)=1}_{a_{i^*j^*}} \qquad \boldsymbol{\delta}^{M}_{C_{i1}} = \Psi_{\delta}\left(\boldsymbol{\delta}^{S,\mathbf{n}(i)>1,\boldsymbol{\kappa}(i)=1}_{a_{i^*j^*}}, \boldsymbol{\delta}^{M}_{a_{i^*j^*}}, \boldsymbol{\gamma}^{M}_{C_{ij}}, \boldsymbol{\gamma}^{S}_{a_{i^*j^*}}, \boldsymbol{\gamma}^{M}_{a_{i^*j^*}}\right) \tag{5.34}$$

$$\mathbf{M}^{S}_{C_{i1}\to S} = \mathbf{M}^{S}_{a_{i^*j^*}\to S} \qquad \mathbf{M}^{M}_{C_{i1}\to S} = \Psi_{M}\left(\mathbf{M}^{S}_{a_{i^*j^*}\to S}, \mathbf{M}^{M}_{a_{i^*j^*}\to S}, \boldsymbol{\delta}^{S,\mathbf{n}(i)>1,\boldsymbol{\kappa}(i)=1}_{a_{i^*j^*}}\right) \tag{5.35}$$

$$\boldsymbol{\delta}^{S}_{C_{i2}} = \mathbf{0} \qquad \boldsymbol{\delta}^{M}_{C_{i2}} = \Psi_{\delta}\left(\boldsymbol{\delta}^{S,\mathbf{n}(i)>1,\boldsymbol{\kappa}(i)=1}_{a_{i^*j^*}}, \boldsymbol{\delta}^{M}_{a_{i^*j^*}}, \boldsymbol{\gamma}^{M}_{C_{ij}}, \boldsymbol{\gamma}^{S}_{a_{i^*j^*}}, \boldsymbol{\gamma}^{M}_{a_{i^*j^*}}\right) \tag{5.36}$$

$$\mathbf{M}^{S}_{C_{i2}\to S} = \mathbf{0} \qquad \mathbf{M}^{M}_{C_{i2}\to S} = \Psi_{M}\left(\mathbf{M}^{S}_{a_{i^*j^*}\to S}, \mathbf{M}^{M}_{a_{i^*j^*}\to S}, \boldsymbol{\delta}^{S,\mathbf{n}(i)>1,\boldsymbol{\kappa}(i)=1}_{a_{i^*j^*}}\right) \tag{5.37}$$

The rationale behind these equations is the following:

- The SPF-s and RF-s of C_{i1} can be carried over from the SPF-s and RF-s of $a_{i^*j^*}$, since those would lead to a safety goal violation in the absence of the safety mechanisms.
- C_{i2} has no SPF-s and RF-s, since none of its faults can lead to a safety goal violation on its own, even in the absence of safety mechanisms.
- The MPF-s of C_{ij} are on one hand the SPF-s of $a_{i^*j^*}$ (as MPF,D) and the MPF-s of $a_{i^*j^*}$. This is represented by the combination function Ψ.

The equations related to C_{i3} are structurally identical to those of C_{i2}, if $\mathbf{n}(i) = 3$ and $\boldsymbol{\kappa}(i) = 1$.

In case of a TMR topology (i.e. $\mathbf{n}(i) = 3$, $\boldsymbol{\kappa}(i) = 2$), the mapping can be represented by the following equations:

$$\boldsymbol{\delta}^{S}_{C_{ij}} = \boldsymbol{\delta}^{S,\mathbf{n}(i)>1,\boldsymbol{\kappa}(i)=2}_{a_{i^*j^*}} \qquad \boldsymbol{\delta}^{M}_{C_{ij}} = \Psi_{\delta}\left(\boldsymbol{\delta}^{S,\mathbf{n}(i)>1,\boldsymbol{\kappa}(i)=2}_{a_{i^*j^*}}, \boldsymbol{\delta}^{M}_{a_{i^*j^*}}, \boldsymbol{\gamma}^{M}_{C_{ij}}, \boldsymbol{\gamma}^{S}_{a_{i^*j^*}}, \boldsymbol{\gamma}^{M}_{a_{i^*j^*}}\right) \tag{5.38}$$

$$\mathbf{M}^{S}_{C_{ij}\to S} = \mathbf{M}^{S}_{a_{i^*j^*}\to S} \qquad \mathbf{M}^{M}_{C_{ij}\to S} = \Psi_{M}\left(\mathbf{M}^{S}_{a_{i^*j^*}\to S}, \mathbf{M}^{M}_{a_{i^*j^*}\to S}\right) \tag{5.39}$$

Note, that the equations are structurally identical for C_{i1}, C_{i2} and C_{i3}. The rationale behind these equations is the following:

- The SPF-s and RF-s of C_{ij} can be carried over from the SPF-s and RF-s of $a_{i^*j^*}$, since those would lead to a safety goal violation in the absence of the safety mechanisms. This applies for all three components of S_i, since they are equally handled by the voter.
- The MPF-s of C_{ij} are on one hand the SPF-s of $a_{i^*j^*}$ (as MPF,D) and the MPF-s of $a_{i^*j^*}$. This is represented by the combination function Ψ.

Note, that the indices i and j for the selected components C_{ij}; and i* and j* for the component selection $a_{i^*j^*}$ intend to represent that i is not necessarily equal to i^* (and j is not necessarily equal to j^*), since these represent different indices.

Based on the equations above, the most important question arising is, how the combination functions Ψ_M and Ψ_δ are specified. As already shown above, these functions are intended to create the MPF-related $\boldsymbol{\delta}$ vectors and FEMM-s of C_{ij}. The combination functions are based on the following basic

principle: For each failure of a component choice a_{ij}, if $\mathbf{M}^S_{a_{ij}}(a,b)=1$ then $\mathbf{M}^M_{a_{ij}}(a,b)=0$, and vice versa. This means that a failure mode $\mathbf{f}_{a_{ij}}(b)$ can lead to $\mathbf{f}_S(a)$ either as SPF or as MPF, but not both.

The functions Ψ_M and Ψ_δ create the matrix $\mathbf{M}^M_{C_{ij}\to S}$ and the vector $\boldsymbol{\delta}^M_{C_{ij}}$ in two steps:

1. Function Ψ_M: Create $\mathbf{M}^M_{C_{ij}\to S}$ by sweeping through all its elements one by one.
2. Function Ψ_δ: Based on the $\mathbf{M}^M_{C_{ij}\to S}$, the vector $\boldsymbol{\gamma}^M_{C_{ij}}$ derived from it, and the component choice criticality vectors $\left(\boldsymbol{\gamma}^S_{a_{i^*j^*}}\text{ and }\boldsymbol{\gamma}^M_{a_{i^*j^*}}\right)$ determine the elements of $\boldsymbol{\delta}^M_{C_{ij}}$ one by one.

In case of a dynamic redundant architecture, i.e. $\mathbf{n}(i)>1$, $\boldsymbol{\kappa}(i)=1$, the function Ψ_M determines the elements of the matrix $\mathbf{M}^M_{C_{i1}\to S}$ using the following equation:

$$\mathbf{M}^M_{C_{i1}\to S}(a,b)=\begin{cases}0 & \text{if } \left(\mathbf{M}^S_{a_{ij}}(a,b)=0\wedge\mathbf{M}^M_{a_{ij}}(a,b)=0\right)\vee\left(\mathbf{M}^S_{a_{ij}}(a,b)=1\wedge\boldsymbol{\delta}^{S,\mathbf{n}>1,\boldsymbol{\kappa}(i)=1}_{a_{i^*j^*}}(b)=0\right),\\ 1 & \text{if } \left(\mathbf{M}^S_{a_{ij}}(a,b)=1\wedge\boldsymbol{\delta}^{S,\mathbf{n}>1,\boldsymbol{\kappa}(i)=1}_{a_{i^*j^*}}(b)>0\right)\vee\mathbf{M}^M_{a_{ij}}(a,b)=1.\end{cases} \tag{5.40}$$

The rationale behind this equation is the following:

- If $\mathbf{M}^S_{a_{ij}}(a,b)=0\wedge\mathbf{M}^M_{a_{ij}}(a,b)=0$, then the failure mode $\mathbf{f}_{\mathbf{a_{ij}}}(b)$ cannot lead to the system level failure mode $\mathbf{f_S}(a)$ at all.
- If $\mathbf{M}^S_{a_{ij}}(a,b)=1\wedge\boldsymbol{\delta}^{S,\mathbf{n}(i)>1,\boldsymbol{\kappa}(i)=1}_{a_{i^*j^*}}(b)=0$, then failure mode $\mathbf{f}_{a_{ij}}(b)$ is potentially an SPF, hence there is no detected portion, that could be MPF,L.
- If $\mathbf{M}^S_{a_{ij}}(a,b)=1\wedge\boldsymbol{\delta}^{S,\mathbf{n}(i)>1,\boldsymbol{\kappa}(i)=1}_{a_{i^*j^*}}(b)>0$, the detected portion of failure mode $\mathbf{f}_{a_{ij}}(b)$ could become MPF,L.
- If $\mathbf{M}^M_{a_{ij}}(a,b)=1$, failure mode $\mathbf{f}_{a_{ij}}(b)$ leads to the system level failure mode $\mathbf{f}_S(a)$ as MPF in the first place.

Based on the determined $\mathbf{M}^M_{C_{i1}\to S}$ matrix, each k-th element of the vector $\boldsymbol{\delta}^M_{C_{i1}}$ can be calculated using the following equation:

$$\boldsymbol{\delta}^M_{C_{i1}}(k)=\begin{cases}0 & \text{if } \boldsymbol{\gamma}^M_{C_{ij}}(k)=0,\\ \boldsymbol{\delta}^M_{a_{i^*j^*}}(k) & \boldsymbol{\gamma}^M_{C_{ij}}(k)=1.\end{cases} \tag{5.41}$$

The rationale behind this equation is the following:

- The $\boldsymbol{\delta}^M_{a_{i^*j^*}}(k)$ shall be taken over if the failure mode $\mathbf{f}_{a_{ij}}(b)$ is critical as MPF.
- o shall be entered if the failure mode $\mathbf{f}_{a_{ij}}(b)$ is not critical as MPF, because in that case the DC value is irrelevant.

In case of a dynamic redundant architecture, i.e. $\mathbf{n}(i)>1$, $\boldsymbol{\kappa}(i)=1$, the function Ψ_M determines the elements of the matrix $\mathbf{M}^M_{C_{i2}\to S}$ using the following equation:

$$\mathbf{M}^M_{C_{i2}\to S}(a,b)=\begin{cases}0 & \text{if } \mathbf{M}^S_{a_{ij}}(a,b)=0\wedge\mathbf{M}^M_{a_{ij}}(a,b)=0,\\ 1 & \text{if } \mathbf{M}^S_{a_{ij}}(a,b)=1\vee\mathbf{M}^M_{a_{ij}}(a,b)=1.\end{cases} \tag{5.42}$$

The rationale behind this equation is, that since this is a backup module, all of the failure modes of the component choice's (a_{ij}) failure modes, that can lead to system level failure mode $\mathbf{f_S}(a)$ as SPF/RF or as MPF, become potential MPF-s of the component C_{ij}.

Based on the determined $\mathbf{M}^M_{C_{i2}\to S}$ matrix, each k-th element of the vector $\boldsymbol{\delta}^M_{C_{i2}}$ can be calculated by the function Ψ_δ using the following equation:

$$\boldsymbol{\delta}^M_{C_{i2}}(k)=\begin{cases}0 & \text{if } \boldsymbol{\gamma}^M_{C_{ij}}(k)=0,\\ \boldsymbol{\delta}^{S,\mathbf{n}(i)>1,\boldsymbol{\kappa}(i)=1}_{a_{i^*j^*}}(k) & \text{if } \boldsymbol{\gamma}^S_{a_{i^*j^*}}(k)=1,\\ \boldsymbol{\delta}^M_{a_{i^*j^*}}(k) & \text{if } \boldsymbol{\gamma}^S_{a_{i^*j^*}}(k)=0\wedge\boldsymbol{\gamma}^M_{a_{i^*j^*}}(k)=1.\end{cases} \tag{5.43}$$

The rationale behind these equations is the following:

- If $\gamma^M_{C_{ij}}(k) = 0$, then the failure mode $\mathbf{f}_{C_{i2}}(k)$ is not critical as MPF, therefore the DC value is practically irrelevant.
- If $\gamma^S_{a_{i^*j^*}}(k) = 1$, the failure mode $\mathbf{f}_{C_{i2}}(k)$ will become MPF-s, because C_{i2} is a backup module. Note that the $\gamma^M_{a_{i^*j^*}}(k) = 1$ can be valid at the same time. But in this case, the SPF/RF has higher priority, the MPF is negligible.
- If $\gamma^S_{a_{i^*j^*}}(k) = 0 \wedge \gamma^M_{a_{i^*j^*}}(k) = 1$, the failure mode $\mathbf{f}_{C_{i2}}(k)$ is potentially an MPF,L in the first place.

Note, that the same rules apply for C_{i3}.

In case of a TMR architecture, i.e. $\mathbf{n}(i) = 3$, $\boldsymbol{\kappa}(i) = 2$, the function Ψ_M determines the elements of the matrix $\mathbf{M}^M_{C_{ij}\to S}(a,b)$ using the following equation:

$$\mathbf{M}^M_{C_{ij}\to S}(a,b) = \begin{cases} 0 & \text{if } \left(M^S_{a_{ij}}(a,b) = 0 \wedge M^M_{a_{ij}}(a,b) = 0\right) \vee \left(M^S_{a_{ij}}(a,b) = 1 \wedge \delta^{S,\mathbf{n}(i)=3,\boldsymbol{\kappa}(i)=2}_{a_{i^*j^*}}(b) = 0\right), \\ 1 & \text{if } \left(M^S_{a_{ij}}(a,b) = 1 \wedge \delta^{S,\mathbf{n}(i)=3,\boldsymbol{\kappa}(i)=2}_{a_{i^*j^*}}(b) > 0\right) \vee M^M_{a_{ij}}(a,b) = 1. \end{cases} \tag{5.44}$$

Based on the determined $\mathbf{M}^M_{C_{ij}\to S}$ matrix, each k-th element of the vector $\delta^M_{C_{ij}}$ can be calculated using the following equation:

$$\delta^M_{C_{ij}}(k) = \begin{cases} 0 & \text{if } \gamma^S_{C_{ij}}(k) = 0, \\ \delta^M_{a_{i^*j^*}}(k)1 & \gamma^S_{C_{ij}}(k) = 1. \end{cases} \tag{5.45}$$

The rationale behind these equation is the same as for the dynamic redundant architecture's primary component C_{i1}.

Possible model simplifications in case of parallel-serial architectures

As already stated in section 5.2.3, the problem solution can be basically split up in two cases: parallel-serial - and complex architectures. Since complex topologies are generic, parallel-serial ones represent a subset of them. Hence, model simplifications may allow a more straightforward modelling and solution of the automotive RAP. Since – as stated before – many complex automotive physical architectures can be translated into logical parallel-serial topologies. This downgraded model may be beneficial for many practical problems.

In order to identify which model simplifications are possible and appropriate, the characteristics of a parallel-serial architecture need to be clarified, using the mathematical model defined in section 5.2.3:

- Components of parallel-serial architecture do not have MPF-s, if they are "alone" in their respective subsystem ($\mathbf{n}(i) = 1$): $\mathbf{M}^M_{a_{ij}} = \mathbf{0}$. The rationale for this is, that a subsystem of a parallel-serial architecture fails, if a certain number of its subsystems fail.
- Since the calculation formula for the PMHF is specific to the architecture, it can be derived for this particular topology. Since the basic principle behind these architectures is simple, the PMHF calculation will be manageable as well.

The first point is easy to handle, by setting the $\mathbf{M}^M_{a_{ij}}$ to $\mathbf{0}$.

In case of the latter point, the functions z_S and z_{S_i} (equations 5.27 and 5.28) need to be specified for this specific architecture. The advantage of this particular case is, that the structure of the fault tree, used for the calculation of the PMHF is predictable. This fault tree has to consider the $\mathbf{n}(i)$ and $\boldsymbol{\kappa}(i)$ of each subsystem. Note, that due to the assumption (and constraint) that $\mathbf{n}(i) \leq 3$, the complexity of the related fault trees and PMHF calculation equations is limited. To support the definition of these equations, figures 5.12 and 5.13 show the fault trees of the static- and the dynamic redundant case, respectively. Note, that the events in these fault trees are not exact elements of the failure mode vectors, but represent logical categories of those, to limit the size of the fault trees. To simplify the equations, only an upper estimation is determined, by calculating the OR gates using the formula $P(A+B) = P(A) + P(B)$ (instead of $P(A+B) = P(A) + P(B) - P(A)P(B)$). This upper estimation is slightly higher than the actual PMHF. But due to the related constraint, this yields a more conservative system design.

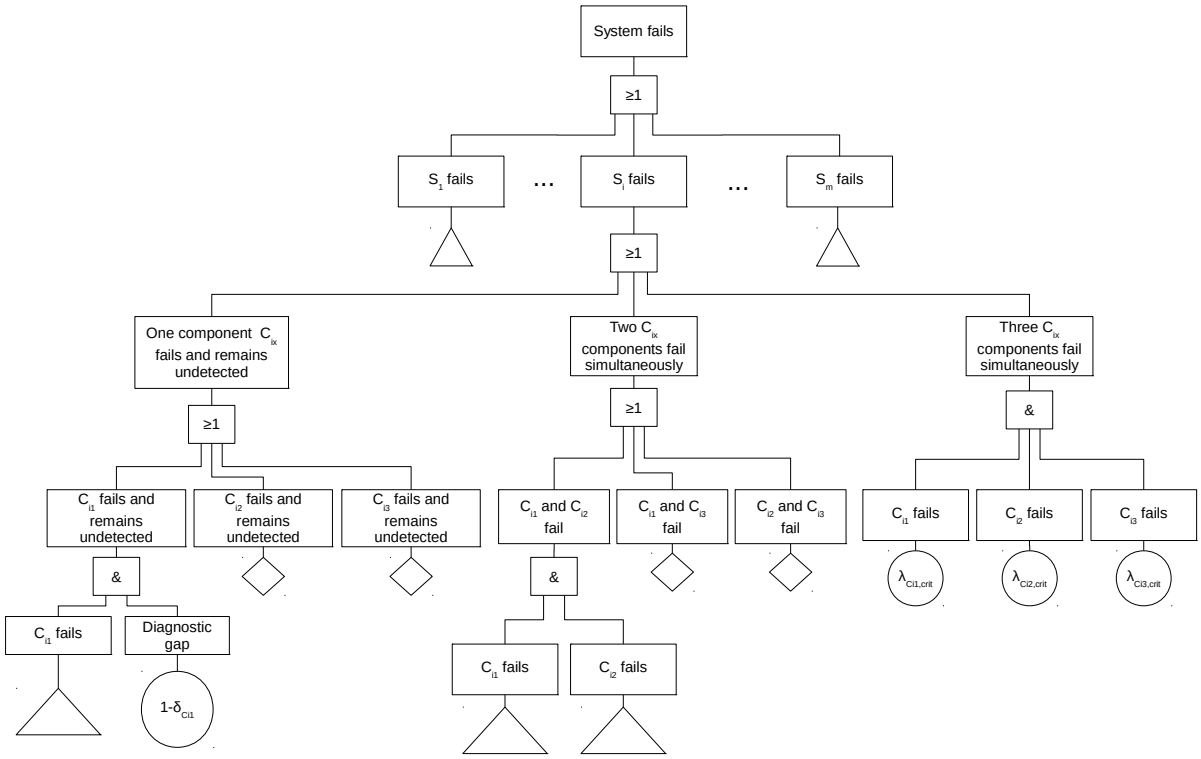


Figure 5.12: Fault tree - Static redundant architecture

Based on these assumptions and simplifications, the system level PMHF is then calculated as:

$$PMHF \leq \sum_{i=1}^{m} \frac{Q_{S_i}}{t} \tag{5.46}$$

Whereas, the failure probability Q_{S_i} of the subsystems S_i is calculated:

$$Q_{S_i} = \begin{cases} \lambda_{C_{i1},crit} t & \text{if } \mathbf{n}(i) = 1, \\ \lambda_{C_{i1},crit,ud} t + \prod_{j=1}^{2} \lambda_{C_{ij},crit} t & \text{if } \mathbf{n}(i) = 2 \text{ and } \boldsymbol{\kappa}(i) = 1, \\ \lambda_{C_{i1},crit,ud} t + \lambda_{C_{i1},crit,d} \lambda_{C_{i2},crit,ud} t^2 + \prod_{j=1}^{3} \lambda_{C_{ij},crit} t & \text{if } \mathbf{n}(i) = 3 \text{ and } \boldsymbol{\kappa}(i) = 1, \\ \sum_{j=1}^{n_i} \lambda_{C_{ij},crit,ud} t + \left(\lambda_{C_{i1},crit} \lambda_{C_{i2},crit} + \lambda_{C_{i1},crit} \lambda_{C_{i3},crit} + \lambda_{C_{i2},crit} \lambda_{C_{i3},crit} \right) t^2 + & \\ + \prod_{j=1}^{3} \lambda_{C_{ij},crit} t & \text{if } \mathbf{n}(i) = 3 \text{ and } \boldsymbol{\kappa}(i) = 2. \end{cases} \tag{5.47}$$

The rationale for this is the following:

- In case of a subsystem without any redundancy, the entire critical failure rate of the sole component of that subsystem will contribute to the PMHF.
- In case of the dynamic redundant architecture, the PMHF can be calculated using the fault tree shown in figure 5.13. Note, that the sub-cases where $\mathbf{n}(i) = 2$ or $\mathbf{n}(i) = 3$ are also shown in the figure.
- In case of the static redundant architecture, the PMHF can be calculated using the fault tree shown in figure 5.13.

The input values for equation 5.47, i.e. the $\lambda_{C_{ij},crit}$, $\lambda_{C_{ij},crit,ud}$ and $\lambda_{C_{i1},crit,d}$ are calculated using the following equations:

$$\lambda_{C_{i1},crit} = \boldsymbol{\gamma}_{C_{i1}}^{ST} \cdot \boldsymbol{\lambda}_{C_{i1}} \tag{5.48}$$

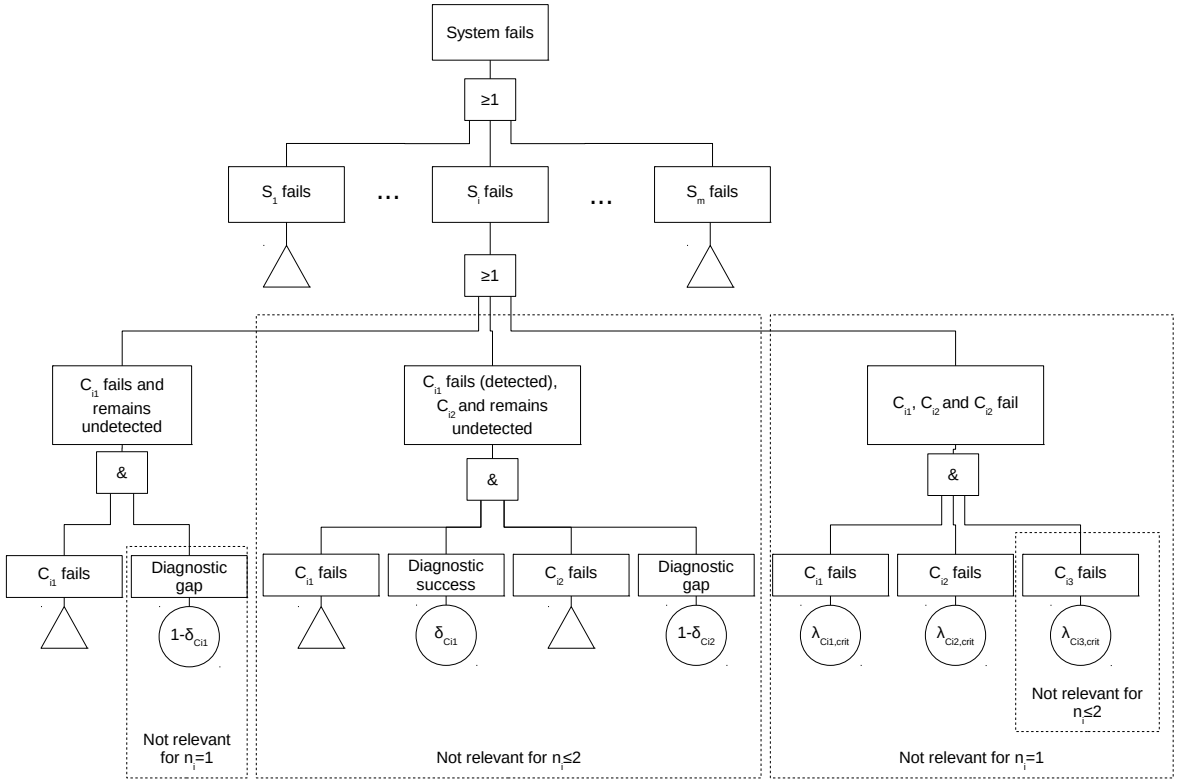


Figure 5.13: Fault tree - Dynamic redundant architecture

$$\lambda_{C_{i1},crit,ud} = \boldsymbol{\gamma}_{C_{i1}}^{ST} \cdot \left(\boldsymbol{\lambda}_{C_{i1}} \circ \left(\mathbf{u} - \boldsymbol{\delta}_{\boldsymbol{C_{i1}}}^{\boldsymbol{S}} \right) \right) \tag{5.49}$$

$$\lambda_{C_{i1},crit,d} = \boldsymbol{\gamma}_{C_{i1}}^{ST} \cdot \left(\boldsymbol{\lambda}_{C_{i1}} \circ \boldsymbol{\delta}_{\boldsymbol{C_{i1}}}^{\boldsymbol{S}} \right) \tag{5.50}$$

$$\lambda_{C_{i2},crit} = \begin{cases} \boldsymbol{\gamma}_{C_{i2}}^{MT} \cdot \boldsymbol{\lambda}_{C_{i2}} & \text{if } \boldsymbol{\kappa}(i) = 1 \\ \boldsymbol{\gamma}_{C_{i2}}^{ST} \cdot \boldsymbol{\lambda}_{C_{i2}} & \text{if } \boldsymbol{\kappa}(i) = 2. \end{cases} \tag{5.51}$$

$$\lambda_{C_{i2},crit,ud} = \begin{cases} \boldsymbol{\gamma}_{C_{i2}}^{MT} \cdot \left(\boldsymbol{\lambda}_{C_{i2}} \circ \left(\mathbf{u} - \boldsymbol{\delta}_{\boldsymbol{C_{i2}}}^{\boldsymbol{M}} \right) \right) & \text{if } \boldsymbol{\kappa}(i) = 1 \\ \boldsymbol{\gamma}_{C_{i2}}^{ST} \cdot \left(\boldsymbol{\lambda}_{C_{i2}} \circ \left(\mathbf{u} - \boldsymbol{\delta}_{\boldsymbol{C_{i2}}}^{\boldsymbol{S}} \right) \right) & \text{if } \boldsymbol{\kappa}(i) = 2. \end{cases} \tag{5.52}$$

$$\lambda_{C_{i3},crit} = \begin{cases} \boldsymbol{\gamma}_{C_{i3}}^{MT} \cdot \boldsymbol{\lambda}_{C_{i3}} & \text{if } \boldsymbol{\kappa}(i) = 1 \\ \boldsymbol{\gamma}_{C_{i3}}^{ST} \cdot \boldsymbol{\lambda}_{C_{i3}} & \text{if } \boldsymbol{\kappa}(i) = 2. \end{cases} \tag{5.53}$$

$$\lambda_{C_{i3},crit,ud} = \begin{cases} \boldsymbol{\gamma}_{C_{i3}}^{MT} \cdot \left(\boldsymbol{\lambda}_{C_{i3}} \circ \left(\mathbf{u} - \boldsymbol{\delta}_{\boldsymbol{C_{i3}}}^{\boldsymbol{M}} \right) \right) & \text{if } \boldsymbol{\kappa}(i) = 1 \\ \boldsymbol{\gamma}_{C_{i3}}^{ST} \cdot \left(\boldsymbol{\lambda}_{C_{i3}} \circ \left(\mathbf{u} - \boldsymbol{\delta}_{\boldsymbol{C_{i3}}}^{\boldsymbol{S}} \right) \right) & \text{if } \boldsymbol{\kappa}(i) = 2. \end{cases} \tag{5.54}$$

where $\circ$ denotes the element-wise vector multiplication, and $\mathbf{u} = \begin{pmatrix} 1 & 1 & ... & 1 \end{pmatrix}$ represents a vector of the appropriate length, where each value is 1. Note, that these equations simply consider where the relevant $\boldsymbol{\delta}_{\boldsymbol{C_{ij}}}$ and $\boldsymbol{\gamma}_{\boldsymbol{C_{ij}}}$ vectors are: on the SPF or the MPF analysis side of the FMEDA. And since this depends on the architecture (i. e. TMR or dynamic redundant), the $\mathbf{n}(i)$ and $\boldsymbol{\kappa}(i)$ provide guidance.

As already told in section 5.2.3, one of the solutions to cope with the definition of the application specific z_S and z_{S_i} functions, is to calculate the PMHF to the *parallel-serial equivalent* of a complex system. As already stated at the start of the section, a serial system is characterised by the following equation: $\mathbf{M}^M_{a_{ij}} = \mathbf{0}$. Therefore, the serial equivalent of a complex system is created by:

$$\mathbf{M}^M_{a_{ij}} := \mathbf{0} \qquad \text{where } i = 1...m,\ j = 1...\mathbf{q}(i) \tag{5.55}$$

This will lead to two effects, depending on the architectural role of the subsystems:

- If subsystem has only MPF-s (i.e. it is part of a parallel path of the system architecture), it will be completely omitted from the parallel-serial equivalent. This can therefore be only the case if $\mathbf{M}^S_{a_{ij}} = \mathbf{0}$ for all $j = 1...\mathbf{q}(i)$ in subsystem S_i.
- If a subsystem has SPF-s and RF-s as well (i.e. if $\mathbf{M}^S_{a_{ij}} \neq \mathbf{0}$ for all $j = 1...\mathbf{q}(i)$ in subsystem S_i), that subsystem will be included in the parallel-serial equivalent. But its MPF-s will not be considered in the PMHF calculation.

It is important to investigate, how the PMHF of the complex system ($PMHF_{comp}$) is related to the PMHF of its parallel-serial equivalent ($PMHF_{pse}$). First of all, it is obvious, that

$$PMHF_{comp} = PMHF_{pse} + \Delta_{PMHF} \tag{5.56}$$

where $\Delta_{PMHF} > 0$. Therefore the optimisation may lead to a solution that does not fulfil the PMHF constraint set for the RAP. On the other hand, it has to be investigated, how high this Δ_{PMHF} may be.

The contribution of MPF-s compared to SPF-s and RF-s depends mainly on the proportion of their related failure rates, as already shown in section 4.4.3 for the compared HW architectures. For *nearly* serial architectures (i.e. where the amount of parallel paths in the architecture is sufficiently low), this contribution may even be negligible. To support an appropriate evaluation, the following values can be used as guidance:

First, the ratio calculated for the component choices of a subsystem:

$$r^{S_i}_{\lambda} = \frac{\sum_{j=1}^{\mathbf{q}(i)} \gamma^{MT}_{a_{ij}} \cdot \boldsymbol{\lambda}_{a_{ij}}}{\sum_{j=1}^{\mathbf{q}(i)} \gamma^{ST}_{a_{ij}} \cdot \boldsymbol{\lambda}_{a_{ij}}} \tag{5.57}$$

This value is relevant, because certain subsystems tend to have a higher $r^{S_i}_{\lambda}$ than others. In the example shown in figure 5.5, S_6 (actuator output voltage monitoring) will definitely have more MPF-s than SPF-s and RF-s, due to its function: It is intended to support monitoring based safety mechanisms.

Then, this ratio can be extended for all component choices.

$$r^{A}_{\lambda} = \frac{\sum_{i=1}^{m} \sum_{j=1}^{\mathbf{q}(i)} \gamma^{MT}_{a_{ij}} \cdot \boldsymbol{\lambda}_{a_{ij}}}{\sum_{i=1}^{m} \sum_{j=1}^{\mathbf{q}(i)} \gamma^{ST}_{a_{ij}} \cdot \boldsymbol{\lambda}_{a_{ij}}} \tag{5.58}$$

This value is relevant, but may be biased, depending on the number of component choices in each subsystem. Therefore, great care shall be taken if applying this ratio as metric. Note, that both the $r^{S_i}_{\lambda}$ and the r^{A}_{λ} cannot take the DC values (i.e. the $\boldsymbol{\delta}^{\boldsymbol{S}}_{\boldsymbol{a_{ij}}}$ and the $\boldsymbol{\delta}^{\boldsymbol{M}}_{\boldsymbol{a_{ij}}}$ matrices) into account, since they can be only interpreted for a certain system configuration.

Further research shall clarify, using a sufficient set of practical examples, which thresholds can be applied to use these values as hard criteria.

Outlook - Extension of the model to cope with m-to-n redundancy allocation

As stated before, due to the broad use of integrated components in the automotive industry, the redundancy allocation cannot be always represented as a 1-to-1 relationship. Figure 5.14 shows the different types that need to be considered. In this figure, the base architecture consists of a sensor, an ECU and an actuator. This is then extended into a dynamic redundant architecture, in three distinctive ways:

(a) *1-to-n:* In this case, one physical component represents the redundant counterpart of multiple other ones. In the example figure, the smart actuator in the backup path serves as redundant path to the ECU and the actuator of the primary path.
(b) *n-to-1:* In this case, several physical components represent the redundant counterpart of another one. In the example figure, the single ECU of the primary path has a split ECU, consisting of two components, as redundant counterpart in the backup path.
(c) *m-to-n:* This case is the combination of the 1-to-n and n-to-1 types described above. In the example figure, the boundaries of the backup path components are completely different from the primary path.

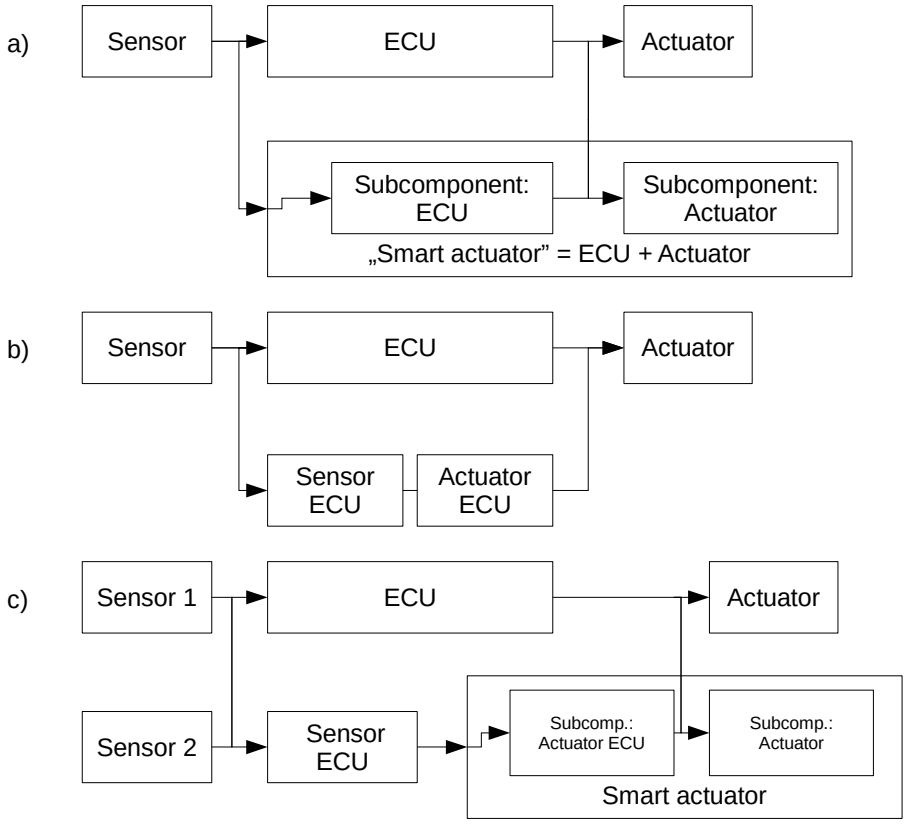


Figure 5.14: Types of multiple redundancy allocation, a) 1-to-n, b) n-to-1, c) m-to-n

To cope with these types, the term *virtual components* is introduced. Virtual components may be of two different types, as shown in figure 5.15:

- *Type 1* virtual components consist of several physical elements. These are used to handle n-to-1 redundancy allocation. A virtual component can be therefore the redundant counterpart of a physical component. In figure 5.15 a), virtual component V_1 contains physical components $P_{1.1}$ and $P_{1.2}$.
- *Type 2* virtual components are created by splitting up a physical component. These are intended to cope with 1-to-n redundancy allocation. The goal is the same, as in case of type 1 virtual components: a virtual component can be the redundant complement of a physical one. In figure 5.15 b), physical component P_1 is split up into virtual components $V_{1.1}$ and $V_{1.2}$.

Using the notation of the virtual elements, a possible parallel serial architecture is shown in figure 5.16:

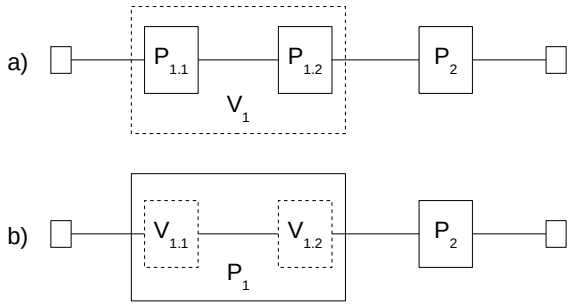


Figure 5.15: Virtual component types, a) Type 1, b) Type 2

- Virtual components C_{21} and C_{31} form a physical component. Both are therefore type 2 virtual components
- Virtual components C_{12} and C_{22} form a physical component. Hence, both are type 2 virtual components
- Physical components $C_{42.1}$ and $C_{42.1}$ assemble a type 1 virtual component.

As it can be seen, one physical element can be therefore part of several subsystems. Hence, the split-up of the system into subsystems shall consider all virtual elements.

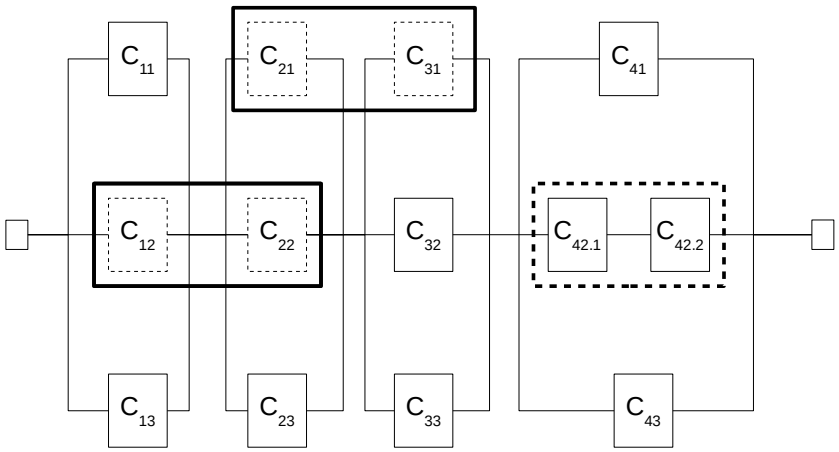


Figure 5.16: Parallel serial architecture with virtual components

Type 1 virtual components can be handled in the same way as physical ones, throughout the whole model and the optimisation algorithm. The reason for this is, that *components* have not been considered by the model as being atomic. Therefore, they *may* consist of several subcomponents, without any effect on the mathematical model.

Type 2 virtual components, on the other hand need to be investigated more in detail. This investigation focuses on the three main parts of the mathematical model:

- Generic aspects of architecture modelling
- Calculation of the cost, packaging space need and weight.
- Failure model.

The calculation of cost, packaging space need and weight follows the same scheme. It needs to be considered, that if a type 2 virtual component is chosen to be included in the architecture, the entire physical component it belongs to, has to be included. Therefore, even if other virtual subcomponents

belonging to it are not used, the weight, packaging space and cost of the entire physical component has to be included in the calculations. In order to cope with this, a mathematical model has to be found to model the connections between type 2 virtual components, showing which one of them form a physical component. This can be represented by the matrix **V**, completing the component choice matrix **A**. The structure of this matrix can be described with the following equation:

$$\mathbf{V}(i,j) = \begin{cases} 0 & \text{if } a_{ij} \text{ is a physical component or a type 1 virtual component,} \\ x & \text{if } a_{ij} \text{ is a type 2 virtual component,} \\ -1 & \text{if } a_{ij} \text{ is not available as component choice.} \end{cases} \tag{5.59}$$

where the number $x \in \mathbb{N}^+$ denotes the identifier of the physical component the type 2 virtual components belong to. For the example shown in figure 5.16, the matrix **V** would be (assuming that all component choices are included in the architecture):

$$\mathbf{V} = \begin{bmatrix} 0 & 2 & 0 \\ 1 & 2 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{5.60}$$

Note that the element $\mathbf{V}(i,j)$ of the matrix belongs to the j-th component choice of the i-th subsystem.

Using the matrix **V**, the component choice matrix **Y** can be extended to show which other virtual components are directly or indirectly included. This new matrix, the $\mathbf{Y}^*$ can be calculated using the following equation:

$$\mathbf{Y}^*(i,j) = \begin{cases} 0 & \text{if } a_{ij} \text{ is chosen neither directly or indirectly,} \\ 1 & \text{if } \mathbf{Y}(i,j) = 1, \text{ or if } \exists \quad a_{i'j'}, \text{ where } \mathbf{Y}(i',j') = 1 \text{ and } \mathbf{V}(i,j) = \mathbf{V}(i'j') \neq 0. \end{cases} \tag{5.61}$$

Based on the matrix $\mathbf{Y}^*$, the cost, weight and packaging space need can be calculated using the following equations:

$$G = \sum_{i=1}^{m} \sum_{j=1}^{q_i} \mathbf{Y}^*(i,j) \cdot g_{ij} \tag{5.62}$$

$$W = \sum_{i=1}^{m} \sum_{j=1}^{q_i} \mathbf{Y}^*(i,j) \cdot w_{ij} \tag{5.63}$$

$$P = \sum_{i=1}^{m} \sum_{j=1}^{q_i} \mathbf{Y}^*(i,j) \cdot p_{ij} \tag{5.64}$$

As already mentioned in the list of assumptions applied for the mathematical failure model (see section 5.2.3), the failure modes of the components are considered being independent. The introduction of the m-to-n redundancy allocation imposes a new challenge related to this premise. According to the dependency model presented in section 4.5 (figure 4.53), type 2 virtual components are, by definition, prone to the dependency types *common resource* and *overlapping*. Therefore, the failure model and the component selection, presented in section 5.2.3 can be carried over without modifications, only if the following assumptions apply:

- Type 2 virtual components of the same physical component are independent from each other. This implies that failure modes of a type 2 virtual component have no effect on the other virtual components of the same physical element.
- Failure modes of the type 2 virtual components not directly chosen (i.e. not used in the architecture) have no effect on the system level.

Both assumptions can be proven by a DFA, as presented in section 4.5. If the result of the DFA is that the assumed independence is non-existent, the failure model needs to be extended by the modelling of common cause- and cascading failures. The aim of this model extension is to represent the following:

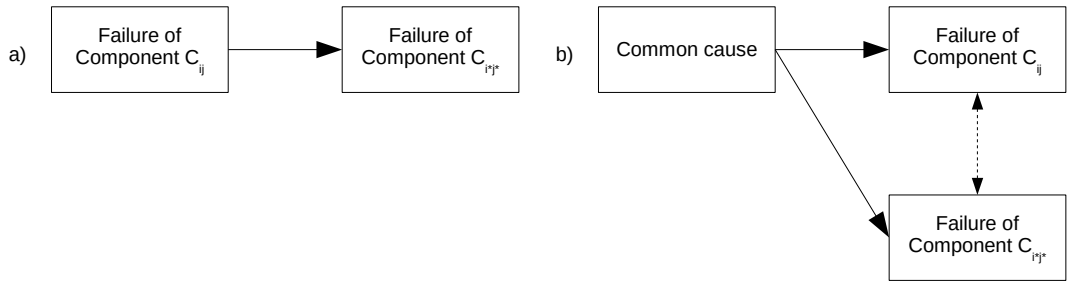


Figure 5.17: Principle behind the FEMM-s representing dependent failures

- Component level failures caused by failures of other components (i.e. cascading failures)
- Simultaneous failures of two components caused by a single common cause (i.e. common cause failures)

To represent these, the FEMM-s can be employed, as shown in figure 5.17. As it can be seen in the referred figure, both in the cascading- and in the common cause failure case, the relation the FEMM has to represent is the following: Are the two failures of two separate physical or virtual components dependent? The type of the dependent failures is practically irrelevant. Based on this assumption, the FEMM between two components can be modelled as:

$$\mathbf{M}_{C_{ij}\rightarrow C_{i^*j^*}}(a,b) = \begin{cases} 0 & \text{if } \mathbf{f}_{C_{ij}}(b) \text{ does not lead to } \mathbf{f}_{C_{i^*j^*}}(a), \\ 1 & \text{if } \mathbf{f}_{C_{ij}}(b) \text{ leads to } \mathbf{f}_{C_{i^*j^*}}(a). \end{cases} \tag{5.65}$$

Note, that this dependent failure modelling approach can be used not only if m-to-n reliability allocation is applied, but in any case if the independence assumption is invalid.

The mathematical modelling of the failure propagation (section 5.2.3) can be extended with the FEMM representing dependent failures. This can be the subject of further research.

5.2.4 Selection of an appropriate optimisation algorithm

The modern overview papers mentioned in section 5.1 ([Pra00], [Wan07] and [Ibr13]) offer good guidance on the selection of an appropriate optimisation algorithm. [Pra00] states that meta-heuristic methods are one of the key developments on RAP, in the 21st century. It also states that heuristic and exact solutions are very difficult to realise. Therefore, attention is focused towards these methods. Based on the reviewed papers mentioned above, the following can be stated:

- EA and in particular GA can be effectively adopted for complex optimisation problems, which the automotive RAP certainly is [Pra00] [Wan07]. There are many papers utilising EA. Its effectiveness is proven in many applications. Recent developments tend towards hybrid GA-s. These solutions combine classical GA-s with heuristic methods to avoid pre-mature convergence [Wan07].
- SA offers also good results but needs more computation power [Pra00].
- Tabu Search Method (TS) can be also recommended, although the available literature is very limited [Pra00] [Wan07].
- A very important recent development is the utilisation of Ant Colony Optimisation (ACO) methods. Its effectiveness has been proven in several applications [Wan07], and can be compared with that of an EA.
- PSO is also a recent development. First results (e.g. [Li12]) are promising [Ibr13]. [Val08] states that PSO is superior to GA, mainly due to the fact that it tends less to pre-mature convergence.

The purpose of this section is to show that the automotive RAP can be solved by an appropriately selected, generally accepted meta-heuristic optimisation method. Furthermore, the goal is to prove

that the failure - and component selection models, developed in section 5.2.3 can be incorporated in this optimisation algorithm. To fulfil this objective, the algorithm is implemented in a straightforward, efficient but effective way. Fine-tuning can be the subject of further research.

Hence, based on the results of the literature study, GA is selected. The main reason for this is, that it is widely use to solve RAP-s, and has been proven in various applications. It can deal with complex system modelling, which is essential, given the requirements of an automotive RAP.

Based on [Oli99], [Yao97], [Smi96], [Ham14] and [Hua08], the GA can be described by algorithm 1.

Algorithm 1 Genetic algorithm for the parallel-serial system with 1-to-n redundancy allocation

```
Initialise parameters of GA
Set number of actual population ρ = 1
Initialise population Ω(ρ)
Evaluate fitness of the initial population
while ρ < ρ_max do
  Select parents
  Create offspring
  Mutate
  Extend population with mutated offspring
  Evaluate fitness of extended population
  Survival of the fittest
  ρ = ρ + 1
end while
```

5.2.5 Genetic algorithm for the optimisation of automotive systems with 1-to-n redundancy allocation

This section presents the steps of the GA as shown by algorithm 1, and explains how the failure model and the component selection (section 5.2.3) can be integrated.

Initialise population

In order to be able to set up the initial population, first the *chromosome definition* shall be clarified. As already explained in section 5.2.3, the component selection is represented by the matrix $\mathbf{I}$, as shown in equation 5.30. The GA will apply this matrix $\mathbf{I}$ as the chromosome of an individual in the population.

Since each individual is represented by a 2-dimensional matrix, the population itself is represented by the 3-dimensional matrix $\mathbf{\Omega}$, where $\mathbf{I}_k = \mathbf{\Omega}(k)$ is the k-th individual of that population. Note, that $1 \leq k \leq N_\Omega$.

The initial matrix $\mathbf{\Omega}$ is created randomly. This random function needs to be intelligent, due to the constraints that apply for the matrix $\mathbf{I}_k$.

For each row of the matrix $\mathbf{Y}$ (which is a sub-matrix of $\mathbf{I}_k$), the number of 1-s (i.e. the $\mathbf{n}(i)$) is limited:

$$1 \leq \mathbf{n}(i) \leq 3 \tag{5.66}$$

as stated as one of the assumptions in section 5.2.1.

The vectors $\boldsymbol{\kappa}$ and $\mathbf{b}$ shall fulfil the following constraints:

$$\boldsymbol{\kappa}(i) = 1 \text{ if } \mathbf{n}(i) \leq 2 \tag{5.67}$$

$$1 \leq \boldsymbol{\kappa}(i) \leq 2 \tag{5.68}$$

$$\mathbf{b}(i) \leq \mathbf{n}(i) \tag{5.69}$$

Fitness evaluation

The fitness evaluation can be derived from the problem statement given in 5.2.3. The fitness function is represented by the function F_{fit} (based on [Smi96]):

$$F_{fit} = G + \Theta(SPFM, LFM, PMHF, W, P) \tag{5.70}$$

As it can be seen, the following factors contribute to the fitness of an individual:

- *Cost*, as the factor to be optimised
- *Penalty function*, depending on if the constraints are met.

As it can be seen from equation 5.70, the better the fitness of the individual, the lower the value of the Ψ function.

The penalty function is defined as:

$$\begin{aligned}\Theta = max(0, \pi_{SPFM}(SPFM_{min} - SPFM)) + max(0, \pi_{LFM}(LFM_{min} - LFM)) + \\ + max(0, \pi_{PMHF}(PMHF - PMHF_{max})) + max(0, \pi_W(W - W_{max})) + \\ + max(0, \pi_P(P - P_{max}))\end{aligned} \tag{5.71}$$

The maximum functions ($max(0, ...)$) are used to avoid negative penalty terms in the equation. The penalty factors need to be adjusted based on experimenting with a specific problem. The lower the penalty factors, the less the algorithm yields to premature convergence. Besides that, the relation of the penalty factors to each other defines the *emphasis* on certain constraints. The higher the penalty factor, the higher the importance of the particular constraint.

The calculation of the SPFM, LFM and PMHF has been already derived from the failure model in section 5.2.3. The calculation of the cost, weight and packaging space need can be calculated using the following equations (based on [Smi96]):

$$G = \sum_{i=1}^{m} \sum_{j=1}^{q_i} \mathbf{\Upsilon}(i,j) \cdot g_{ij} \tag{5.72}$$

$$W = \sum_{i=1}^{m} \sum_{j=1}^{q_i} \mathbf{\Upsilon}(i,j) \cdot w_{ij} \tag{5.73}$$

$$P = \sum_{i=1}^{m} \sum_{j=1}^{q_i} \mathbf{\Upsilon}(i,j) \cdot p_{ij} \tag{5.74}$$

As it can be seen in the equations above, cost, weight and packaging space need are calculated as sum of the related factors of the chosen components. This simple calculation method is applied by e.g. [Smi96], [Ash93] and [Red97]. There are other, more complicated calculation methods (e.g. the one applied by [Yao09]), where the non-linearity of the problem is addressed. Nonetheless, since the goal of this section is to prove that the mathematical model extensions needed to cope with the specifics of an automotive RAP fit into an EA, the simplified models are used.

Parent selection

The parent selection is based on the roulette-wheel method [Oli99]. In this method, the probability of being selected as a parent is proportional to the fitness of the individual. But, compared to the survival of the fittest method, it also allows the selection of weaker individuals, preventing premature convergence. $0.5 \cdot N_\Omega$ parents are selected for breeding in each cycle.

Create offspring

For creating the offspring, a modified single point crossover is applied (see figure 5.18). A slight modification is necessary due to the constraints regarding the vectors $\boldsymbol{\kappa}$ and $\mathbf{b}$. The crossover point is selected randomly for each line of the matrix $\mathbf{Y}$. Note, that in figure 5.18 the grey parts of the matrix $\mathbf{Y}$ represent the -1 values, i.e. the entries where no component choice is available for a certain subsystem.

The handling of the vectors $\boldsymbol{\kappa}$ and $\mathbf{b}$ needs some explanation. Since both of them have to fulfil their related constraints (see equations 5.66, 5.67, 5.68 and 5.69), if a crossover is be applied to them, a subsequent repair would be necessary. To cope with this, the crossover of these vectors is performed in the following steps:

1. Child 1 inherits $\boldsymbol{\kappa}$ of parent 1 and $\mathbf{b}$ of parent 2.
2. Child 2 inherits $\boldsymbol{\kappa}$ of parent 2 and $\mathbf{b}$ of parent 1.
3. $\boldsymbol{\kappa}$ and $\mathbf{b}$ are repaired in accordance with algorithms 3 and 4.

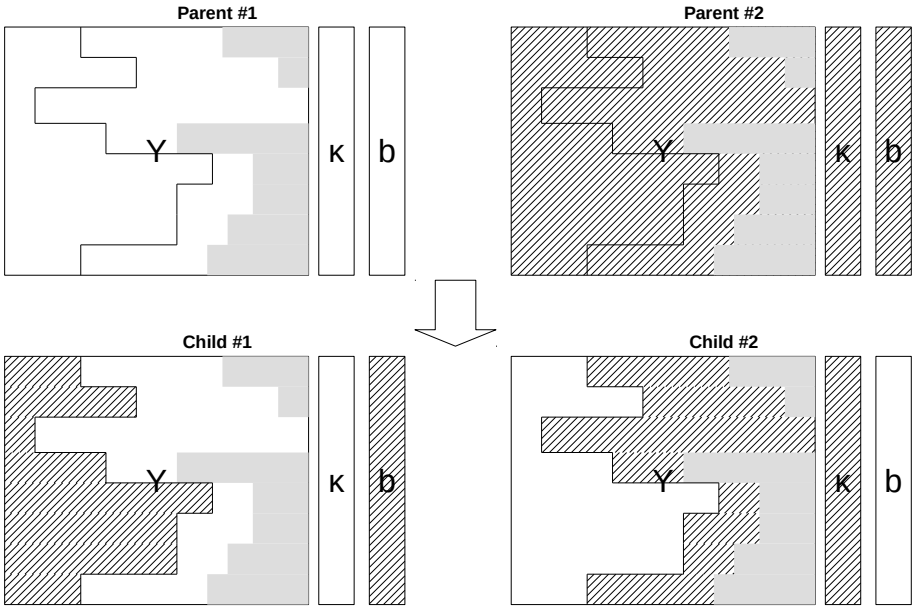


Figure 5.18: Modified single-point crossover

Since the constraints related to the genome of each individual (see equations 5.66, 5.67, 5.68 and 5.69) need to be complied with, a repair algorithm shall be performed for each child after crossover. This repair method is shown in algorithms 2, 3 and 4 . Note that in these algorithms $random(a, b)$ denotes a random number, where $random(a, b) \in \mathbb{N}$ and $a \leq random(a, b) \leq b$.

Algorithm 2 Repair algorithm for the matrix $\mathbf{Y}$

```
for i = 1 : m do
  if n(i) == 0 then
    Set randomly chosen bit to 1
  else if n(i) > 3 then
    for j = 1 : (n(i) − 3) do
      Set randomly chosen bit to 0, where original entry was 1
    end for
  end if
end for
```

Algorithm 3 Repair algorithm for the vector $\boldsymbol{\kappa}$

```
for i = 1 : m do
  if n(i) ≤ 2 then
    κ(i) = 1
  end if
end for
```

Algorithm 4 Repair algorithm for the vector $\mathbf{b}$

```
for i = 1 : m do
  if b_i > n(i) then
    b(i) = random(1, n(i))
  end if
  if κ(i) == 2 then
    b(i) = 1
  end if
end for
```

Mutate offspring

Offspring mutation is applied separately for the matrix $\mathbf{Y}$ and the vectors $\boldsymbol{\kappa}$ and $\mathbf{b}$, in the following order:

- In each line of $\mathbf{Y}$ (i.e. in each subsystem of that individual), $\mathbf{n}(i)$ is changed with 50% probability to a new, random number. Based on the new $\mathbf{n}(i)$, components are selected or de-selected randomly.
- In each line of $\mathbf{Y}$ (i.e. in each subsystem of that individual), two randomly chosen bits are swapped with 50% probability. This way the value $\mathbf{n}(i)$ is not changed, therefore the related constraint (see equation 5.66) is still fulfilled.
- In vector $\boldsymbol{\kappa}$ a randomly chosen element, that can be changed (i.e. where $\mathbf{n}(i) = 3$) is set to a random value $random(1, 2)$, with 50% probability.
- In vector $\mathbf{b}$ a randomly chosen element, that can be changed (i.e. where $\mathbf{n}(i) > 1$ and $\boldsymbol{\kappa}(i) = 1$) is set to a random value $random(1, \mathbf{n}(i))$, with 50% probability.

Due to the first mutation step, the matrix $\mathbf{I}$ needs to be repaired after the mutation, using the algorithms 2, 3 and 4.

Create extended population

After the offspring has been created from the genome of the parents, and altered by mutation, the extended population is created by merging the offspring into the original pool of individuals. Note, that the size of the extended population is $1.5 \cdot N_\Omega$.

Survival of the fittest

In this step, the fitness of the extended population is evaluated, and N_Ω survivors are chosen. The selected individuals will form the next generation. In order to improve the convergence of the algorithm towards a (nearly) optimal solution, for this selection, a purely elitist algorithm has been applied. The best N_Ω individuals are allowed to survive.

5.2.6 Extension of the genetic algorithm to cope with m-to-n redundancy allocation

The genetic algorithm presented in section 5.2.5 needs minor adjustments, to introduce virtual components. The component choice description is extended by the matrix $\mathbf{V}$, showing which components are type 2 virtual ones. Due to the introduction, the split of the system into subsystems (i.e. the number m) may be subject to a re-adjustment.

The fitness calculation is affected by the different calculation methods required for cost, weight and packaging space. The equations 5.62, 5.63 and 5.64 shall be therefore applied.

5.3 Verification of the developed model by practical examples

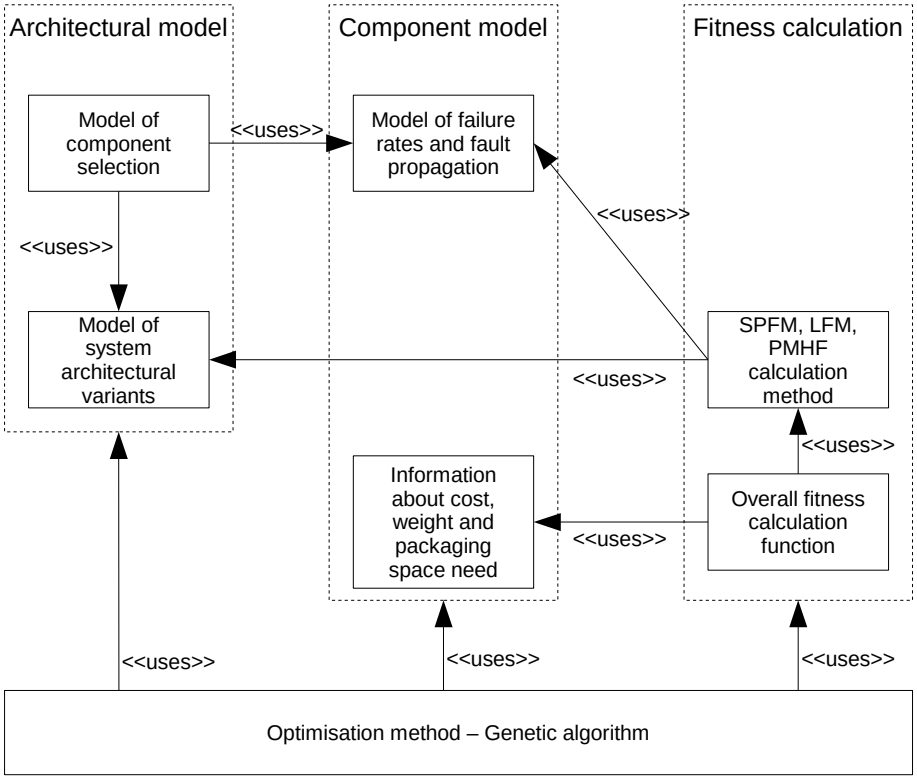


Figure 5.19: Architecture of the developed model

The developed model (figure 5.19) will be verified in two steps:

1. The verification of the mathematical model to calculate the SPFM, LFM and PMHF using the architecture shown in figure 5.5. In this case, the fitness of the "winner" of an optimisation session is calculated manually (using Microsoft Excel), and compared with the results from MatLab [Mat16].
2. The verification of the GA design itself, by using a simple parallel-serial architecture. In this case, the component choices and RAP parameters are chosen that way, that the optimum can be obtained manually. The verification criteria is that the GA has to find this optimum.

Both the mathematical model and the GA have been implemented and verified in MatLab.

5.3.1 Verification of the mathematical model

Introduction of the architecture

The high level system architecture shown in figure 5.5 represents a typical automotive ECU, that can control an electromechanical actuator, based on signals received via Controller Area Network (CAN).

An exemplary optimisation session produced the logical architecture shown in figure 5.20. This architecture is optimal to verify the mathematical model, because all architectural variants are included:

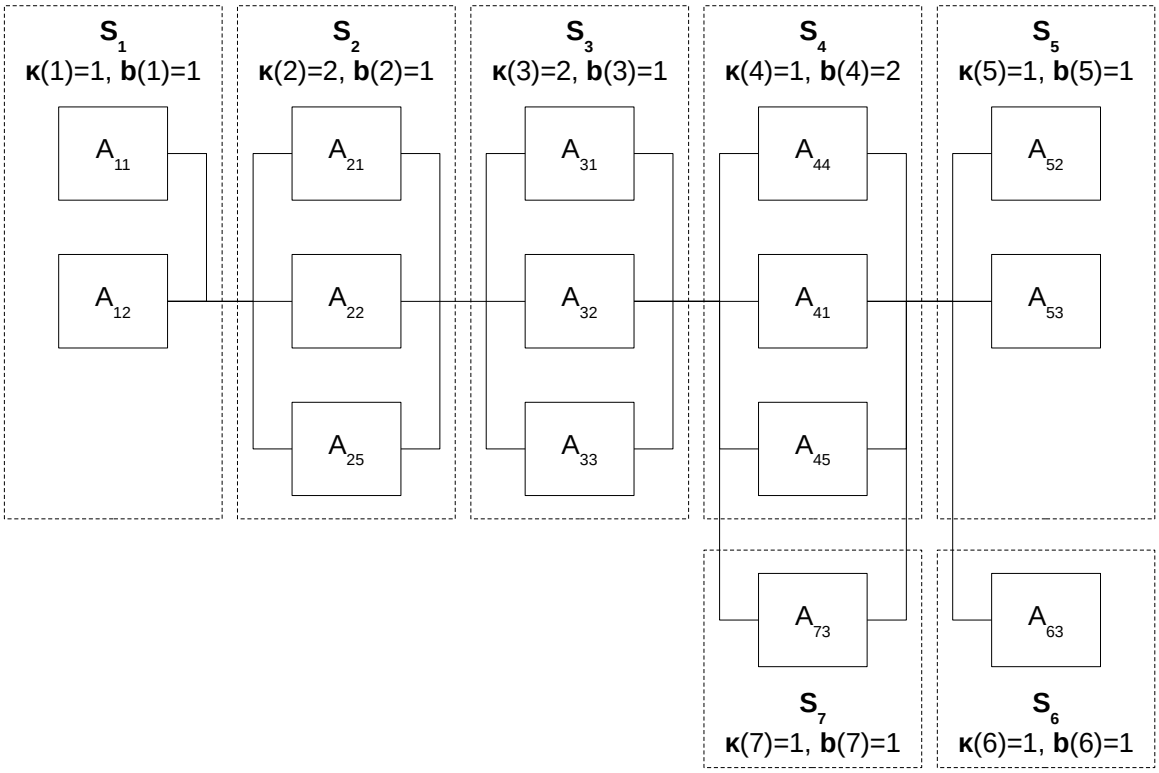


Figure 5.20: Verification of the mathematical model - Logical architectural example

- Single component in a subsystem (i.e. $\mathbf{n}(i) = 1$): S_6, S_7
- Dynamic redundant architecture with 2 components (i.e. $\mathbf{n}(i) = 2$, $\boldsymbol{\kappa}(i) = 1$): S_1, S_5
- Dynamic redundant architecture with 3 components (i.e. $\mathbf{n}(i) = 3$, $\boldsymbol{\kappa}(i) = 1$): S_4
- As special case, dynamic redundant architecture with $\mathbf{b}(i) \neq 1$: S_4
- TMR architecture (i.e. $\mathbf{n}(i) = 3$, $\boldsymbol{\kappa}(i) = 2$): S_2, S_3

Manual calculation of the individual's fitness

All factors of the individual's fitness have been manually calculated using Microsoft Excel. The FMEDA (an excerpt shown in figure 5.21) has been performed in accordance with the [ISO11] [part 5, clause 8]. In order to support an appropriate verification, the failure modes of the component choices have been defined in a comprehensible manner. Therefore, the analysis could be carried over in an ordinary, realistic way.

For the PMHF calculation, the parallel-serial equivalent method is used, to prove the correctness of the equations 5.47 and 5.46. The parallel-serial equivalent of this ECU is shown in figure 5.22. As it can be seen, in this particular example all component choices had failure modes that can lead to at least one system level failure mode as SPF or RF. Therefore, all subsystems are included.

The Excel table to verify the cost, weight and packaging space calculation is shown in figure 5.24.

The results of this verification step were positive. The results from the two diverse sources were identical.

				SPF evaluation				MPF evaluation			
				1	0			1	0		
Arch.	Component (C_{ij})		Failure rate [FIT] (λ_{Cij})	$\vec{\omega}_{ij}$	$\vec{\omega}_{ij}$	DC_{SPF} (δ_{aij})	$\lambda_{SPF/RF}$	$\vec{\omega}_{ij}$	$\vec{\omega}_{ij}$	DC_{MPF}	$\lambda_{MPF\perp}$
S_1 - Dyn. Red.	A_{11}	% f1: UDC 0V	300	1	0	99%	3,00	1	0	100%	0,00
		% f2: UDC voltage drop too high	20	0	0	99%	0,00	0	0	100%	0,00
		% f3: RPP not working	10	0	0	0%	0,00	0	0	0%	0,00
		% f4: UDC not buffered sufficiently	200	1	1	90%	20,00	1	1	100%	0,00
	A_{12}	% f1: UDC 0V	400	0	0	0%	0,00	1	0	90%	40,00
		% f2: UDC voltage drop too high	40	0	0	0%	0,00	0	0	90%	0,00
		% f3: RPP not working	20	0	0	0%	0,00	0	0	0%	0,00
		% f4: UDC not buffered sufficiently	200	0	0	0%	0,00	1	1	90%	20,00
S2 - TMR	A_{21}	% f1: 3V3 out of range	30	1	1	99%	0,30	1	1	100%	0,00
		% f2: 3V3 in-range failure	50	0	0	0%	0,00	0	0	0%	0,00
		% f3: 3V3 stuck at 0V	80	1	0	99%	0,80	1	0	100%	0,00
		% f4: 5V out of range	40	1	1	99%	0,40	1	1	100%	0,00
		% f5: 5V in-range failure	50	0	0	0%	0,00	0	0	0%	0,00
		% f6: 5V stuck at 0V	90	1	0	99%	0,90	1	0	100%	0,00
	A_{22}	% f1: 3V3 out of range	20	1	1	99%	0,20	1	1	100%	0,00
		% f2: 3V3 in-range failure	30	0	0	0%	0,00	0	0	0%	0,00
		% f3: 3V3 stuck at 0V	50	1	0	99%	0,50	1	0	100%	0,00
		% f4: 5V out of range	30	1	1	99%	0,30	1	1	100%	0,00
		% f5: 5V in-range failure	40	0	0	0%	0,00	0	0	0%	0,00
		% f6: 5V stuck at 0V	60	1	0	99%	0,60	1	0	100%	0,00
	A_{23}	% f1: 3V3 out of range	35	1	1	99%	0,35	1	1	100%	0,00
		% f2: 3V3 in-range failure	60	0	0	0%	0,00	0	0	0%	0,00
		% f3: 3V3 stuck at 0V	70	1	0	99%	0,70	1	0	100%	0,00
		% f4: 5V out of range	50	1	1	99%	0,50	1	1	100%	0,00
		% f5: 5V in-range failure	40	0	0	0%	0,00	0	0	0%	0,00
		% f6: 5V stuck at 0V	5	1	0	99%	0,05	1	0	100%	0,00
		% f7: UDC shorted to GND	40	1	0	99%	0,40	1	0	100%	0,00

Figure 5.21: Verification of the mathematical model - excerpt from the FMEDA

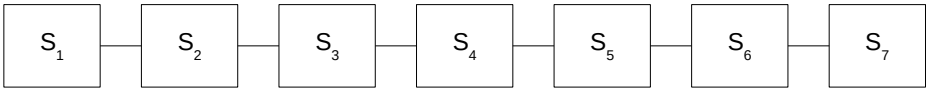


Figure 5.22: Parallel-serial equivalent of the complex architecture example

5.3.2 Verification of the genetic algorithm

Introduction of the architecture

A simple parallel-serial automotive system architecture (figure 5.25 a)) is used to prove the effectiveness of the GA designed in section 5.2.5.

In order to fulfil the goal of the verification, the component choices needed to be designed that way, that the optimum can be determined manually. In order to do so, following design principles have been applied on the component choices and on the parameters of the RAP:

- Simple failure model is applied: Each component choice has only two failure modes, and each one leads to only one of the two system level failure modes. One system level failure is critical, the other is uncritical. The failure rate split is 50% to 50%.
- The component choices' failure rates, cost, weight and packaging space are chosen that way, that the components of the optimal solution are significantly better (table 5.1).
- $\delta_{a_{2j}}^{S,\mathbf{n}(i)>1,\boldsymbol{\kappa}(i)=1} = 0.6$ to ensure, that only a TMR architecture can fulfil the SPFM target value.
- The limit values for the SPFM, LFM and PMHF constraints (i.e. $SPFM_{min}$, LFM_{min} and $PMHF_{max}$) are chosen that way, that they cannot be reached without redundancy in the subsystems S_2 and S_3.
- The limit values for the weight and packaging space constraints (i.e. W_{max} and P_{max}) are chosen that way, that further, unnecessary redundancy would lead to penalty in the fitness calculation.

The parameters of the EA (table 5.2) have been determined by a pre-design and trial-and-error-based

t= 8000 h

Arch.	Component (C_{ij})		Failure rate [FIT] (λ_{Cij})	FTA supporting calculations			
				Q_{crit}	$Q_{crit,d}$	$Q_{crit,ud}$	Q_S_i
S_1 - Dyn. Red.	A_{11}	% f1: UDC 0V	300	4,00E-03	3,82E-03	1,84E-04	2,0320E-04
		% f2: UDC voltage drop too high	20				
		% f3: RPP not working	10				
		% f4: UDC not buffered sufficiently	200				
	A_{12}	% f1: UDC 0V	400	4,80E-03	4,32E-03	4,80E-04	
		% f2: UDC voltage drop too high	40				
		% f3: RPP not working	20				
		% f4: UDC not buffered sufficiently	200				
S2 - TMR	A_{21}	% f1: 3V3 out of range	30	1,92E-03	1,90E-03	1,92E-05	5,5582E-05
		% f2: 3V3 in-range failure	50				
		% f3: 3V3 stuck at 0V	80				
		% f4: 5V out of range	40				
		% f5: 5V in-range failure	50				
		% f6: 5V stuck at 0V	90				
	A_{22}	% f1: 3V3 out of range	20	1,28E-03	1,27E-03	1,28E-05	
		% f2: 3V3 in-range failure	30				
		% f3: 3V3 stuck at 0V	50				
		% f4: 5V out of range	30				
		% f5: 5V in-range failure	40				
		% f6: 5V stuck at 0V	60				
	A_{23}	% f1: 3V3 out of range	35	1,60E-03	1,58E-03	1,60E-05	
		% f2: 3V3 in-range failure	60				
		% f3: 3V3 stuck at 0V	70				
		% f4: 5V out of range	50				
		% f5: 5V in-range failure	40				
		% f6: 5V stuck at 0V	5				
		% f7: UDC shorted to GND	40				

Figure 5.23: Verification of the mathematical model - excerpt from the manually calculated FTA

refinement. The pre-design principles were the following:

- The constraints for the SPFM, LFM and PMHF (i.e. $SPFM_{min}$, LFM_{min} and $PMHF_{max}$) are chosen to represent an ASIL C system, where a fair amount of redundancy is required. See section 4.4.3 for further details.
- The constraints for the weight and packaging space (i.e. W_{max} and P_{max}) have been set exactly to the values of the manually obtained optimum, to "punish" any further, unnecessary redundancy.
- The penalty factors of the fitness function (i.e. π_{SPFM}, π_{LFM}, π_{PMHF}, π_W and π_P) have been set to sufficiently low values, to avoid premature convergence.

Results of the optimisation run

The verification of the GA has been performed on two levels: First, by performing one optimisation run, and analysing its results. In the second step, 100 subsequent optimisation runs have been done, to

Component choice	λ_{crit}	λ_{uncrit}	Cost	Weight	P. space
A_{11}	60	60	10	200	100
A_{12}	60	60	20	300	200
A_{13}	200	200	10	200	100
A_{14}	120	120	30	400	250
A_{21}	400	400	45	1200	1000
A_{22}	400	400	45	1200	1000
A_{22}	800	800	60	2000	1500
A_{24}	400	400	45	1200	1000
A_{31}	500	500	100	800	700
A_{32}	100	100	30	400	500
A_{33}	300	300	30	400	500
A_{34}	100	100	200	1500	800

Table 5.1: Verification of the GA - Parameters of the component choices

		Cost	Weight	P. space
S_1	A_{11}	20	10	25
	A_{12}	15	15	40
S_2	A_{21}	5	2	4
	A_{22}	7	2	3
	A_{25}	5	3	6
S_3	A_{31}	8	5	20
	A_{32}	4	10	25
	A_{33}	20	5	10
S_4	A_{44}	10	10	10
	A_{41}	5	10	10
	A_{45}	8	20	20
S_5	A_{52}	20	50	50
	A_{53}	40	80	80
S_6	A_{63}	1	5	7
S_7	A_{73}	10	3	3
Sum		178	230	313

Figure 5.24: Verification of the mathematical model - calculation of cost, weight and packaging space

compensate the non-determinism of the GA. But since the GA is a non-exact solution of the RAP, both are to be handled accordingly.

The results of the *single optimisation run* have been evaluated, using the following criteria:

- Final result – if the manually obtained minimum has been found.
- Trend of the fitness of the population.
- Coverage of the possible architectural alternatives.

The investigated single optimisation run found the manually determined optimum. As shown in figure 5.26, both the maximal and the minimal fitness improves over the 150 generations. Obviously, the average fitness improves continuously as well. Note, that considering the given problem (see section 5.2.3) lower fitness value means a better individual. The reason for this is that the goal is to minimise cost by fulfilling the given constraints.

The developed representation of the component selection (equation 5.30) leads to a two-dimensional problem domain. The first dimension can be represented by the selected component themselves. The second dimension is the selected topology. In order to find a simplified representation of the individuals covered by the GA, the following representation is introduced:

- The bit matrix $\mathbf{Y}$ is translated into a bit-string: $\mathbf{Y}(1,1), \mathbf{Y}(1,2)...\mathbf{Y}(m, \mathbf{q}(m))$.
- In second step, this bit-string is transformed into a decimal number.
- This decimal representation of the component selection is the x-axis of figure 5.27. On its y-axis, the fitness of the individuals evaluated during the single optimisation run is represented.

Two important findings are shown by figure 5.27:

- The coverage of the optimisation run in focus is acceptable: The crossover and mutation functions prevented pre-mature convergence sufficiently. But a significant number of individuals needed to be evaluated. Further improvement of the parameters of the algorithm, and of the algorithm itself are necessary to decrease the number of generations necessary to reach a stable result.
- The solution space has several local optimums. The algorithm with its current parameters successfully avoids these and heads to the global minimum.

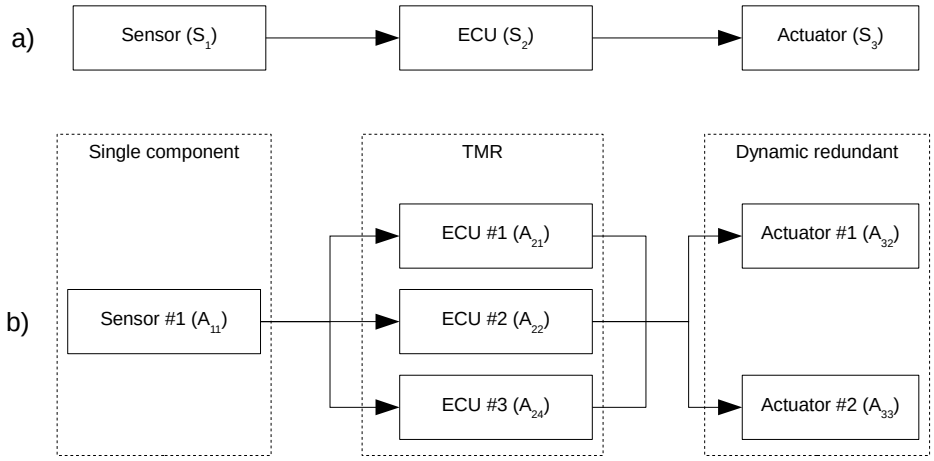


Figure 5.25: Example parallel-serial architecture used for the verification of the GA - a) High-level system architecture, b) Manually obtained optimum

Parameter	Value
N_Ω	20
ρ_{max}	150
W_{max}	4600
P_{max}	4100
$SPFM_{min}$	97 %
LFM_{min}	70 %
$PMHF_{max}$	100
π_{SPFM}	100
π_{LFM}	100
π_{PMHF}	1
π_W	0,001
π_P	0,001

Table 5.2: Verification of the GA - Parameters of the GA

This lack of determinism made it necessary to analyse 100 subsequent optimisation runs, to obtain statistic data about the quality of the algorithm.

The results of the *100 subsequent optimisation runs* have been analysed in accordance with one simple criteria: If the optimum has been found, and if not, then what was the result's fitness. To visualise this, the minimal fitness of the 100 optimisation runs is shown in figure 5.28. As it can be seen the optimum has been found in 94% of the cases. In the other 6 cases, the fitness of the result is approximately 1% higher, than that of the optimum. This indicates, that these results are nearly optimal.

The verification is therefore considered as positive. The developed mathematical model and the straightforward GA are functional and possess the expected performance. Nonetheless, further research is necessary, to optimise the GA, and to improve its computation time.

It is important to emphasise that the algorithm and parameters of the implemented GA may need problem specific fine-tuning to obtain the nearly optimal result, and to avoid pre-mature convergence.

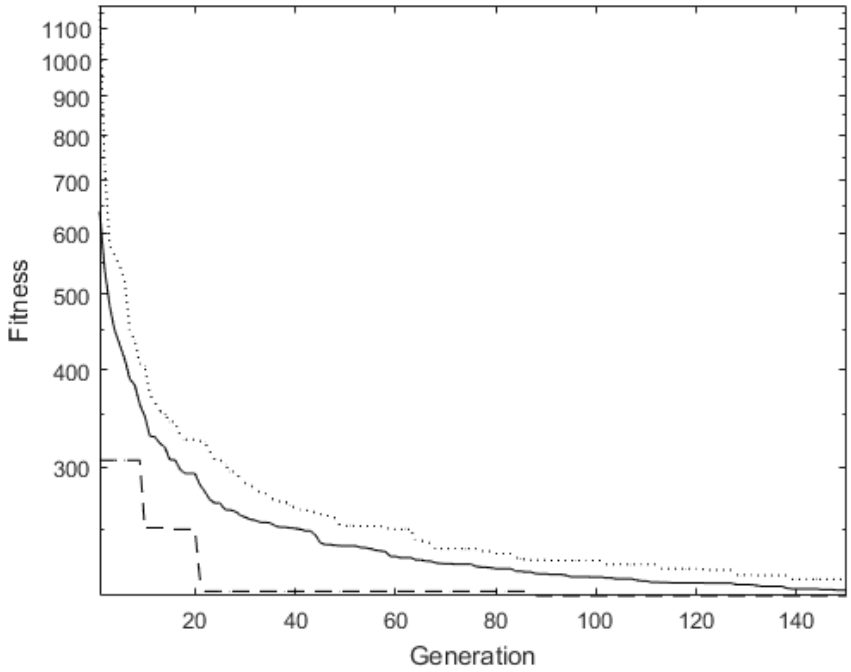


Figure 5.26: Trend of the population's maximal, minimal and average fitness

5.4 Integration of the architecture optimisation into common automotive development processes

As already stated above, the automotive RAP can be applied for optimising system and HW architectures. This section gives guidance on the integration of the architecture optimisation into an automotive development process.

Figure 5.29 shows how the V-model of a system or HW design process is altered when using redundancy optimisation. As it can be seen, the development process still starts with the requirement definition. In this case, this includes the definition of targets and constraints necessary for the RAP.

The subsequent design steps need to change significantly. The architectural- and detailed design phases have to interact in a different way. First, a high level architecture (e.g. Figure 5.25 a)) needs to be defined. Then, without knowing the final design (which will be the result of the optimisation), the detailed design of the component choices needs to be developed. This is necessary, since the mathematical model developed in section 5.2.3 uses the failure model (including failure modes and failure rates), and contains the component choices' weight, cost, etc. Therefore, the optimisation step requires inputs from the high level architectural- and preliminary detailed design specifications. Based on these inputs, the optimisation algorithm determines the low level architectural design (e.g. Figure 5.25 b)), which can be used as input for the final detailed design step, allowing the designers to make final adjustments.

The right side of the V-model is not significantly affected. The integration testing (verifying that the architectural design is implemented as specified) used inputs from both the high- and low level architectural design steps.

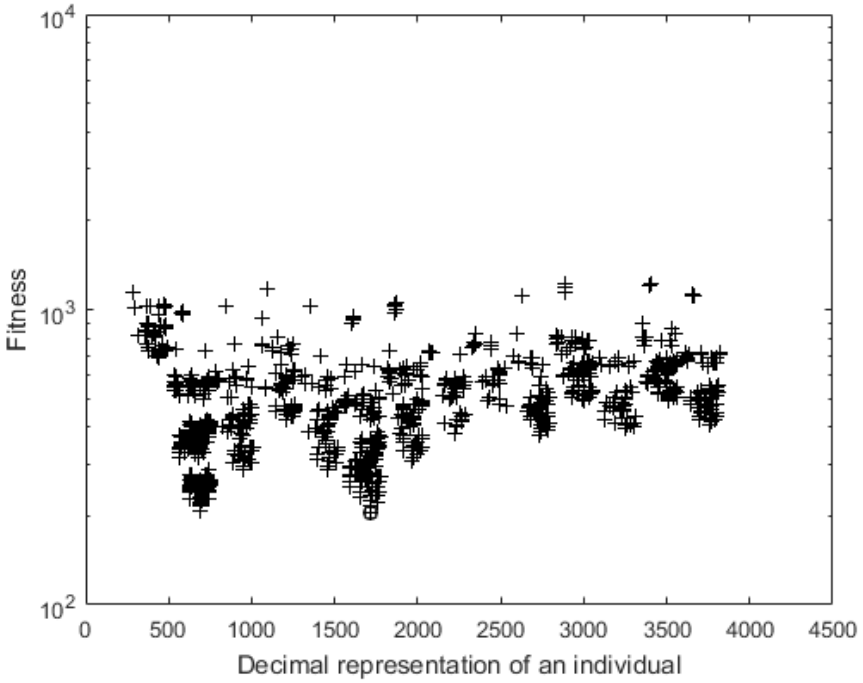


Figure 5.27: Decimal coverage diagram of an optimisation run

5.5 Summary of the automotive RAP

The importance of the optimisation of redundant system and HW will increase in the next decade, mainly due to the increasing importance of fail-operational behaviour. Derived from the results of section 4, it can be clearly stated, that full redundancy is seldom necessary. Therefore, finding the optimal redundant architecture for a certain product will be one of the keys to market success.

The RAP, as currently known in the literature, sets goals very similar to the one described above. Nonetheless, the particularities of automotive products need a different approach. The mathematical model developed in section 5.2.3 offers solutions for these particularities, and provides a solid basis for typical meta-heuristic optimisation methods.

The developed model and the exemplary GA have been verified with practical examples, with positive results.

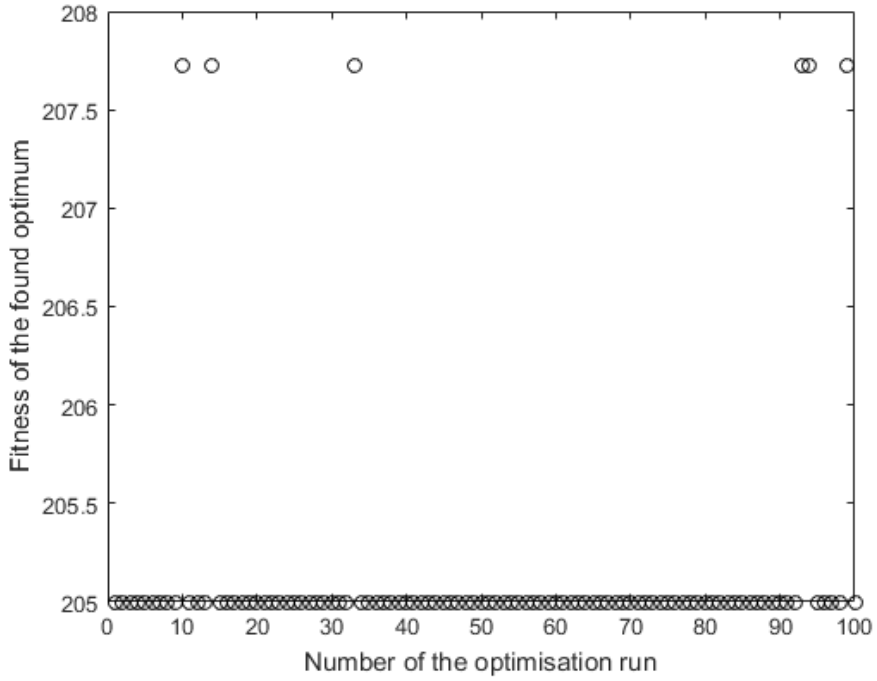


Figure 5.28: Fitness of the found optimum – 100 optimisation runs

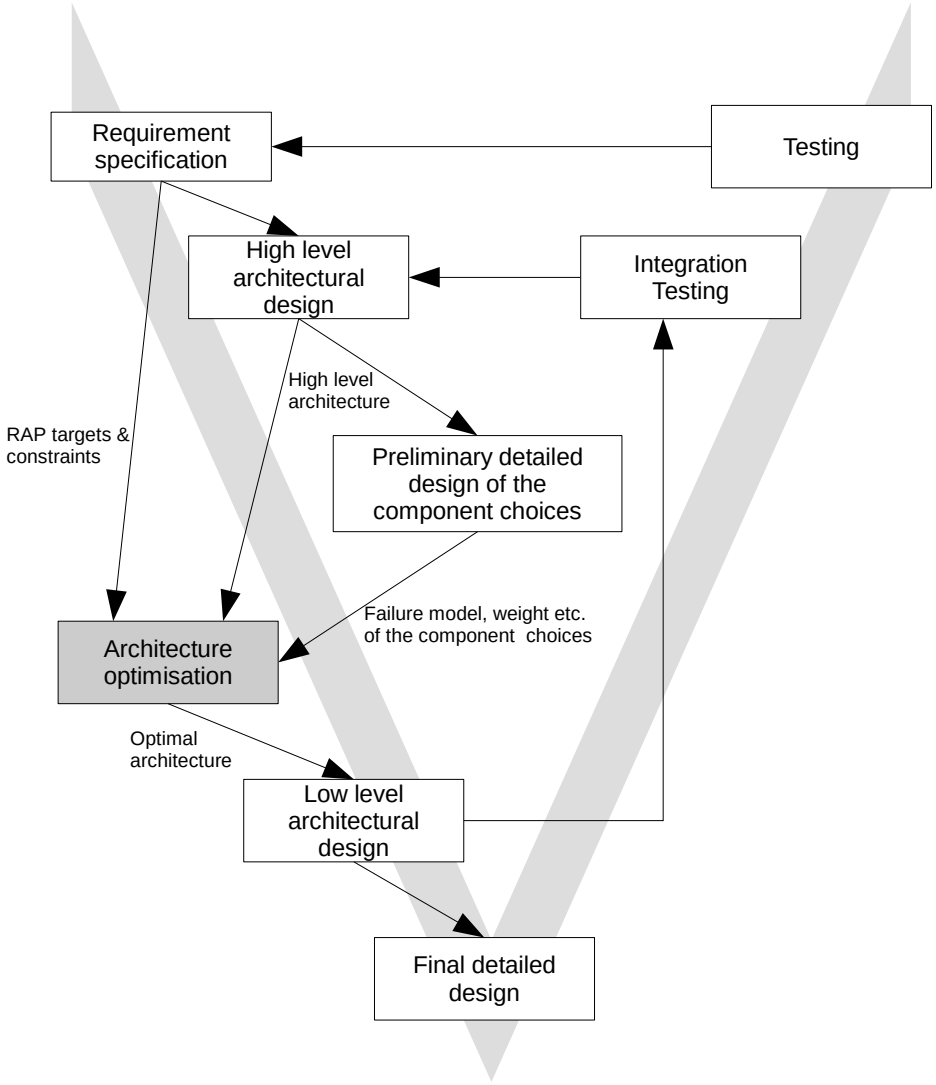


Figure 5.29: Integration of architectural optimisation into common automotive development processes

6 Summary and Outlook

The automotive industry is facing a new challenge. Many of our future vehicle systems will cope with higher availability requirements, due to various reasons. The main reason is definitely the trend towards autonomous and automated driving, the biggest challenge this industry has ever faced. It is not just a next evolutionary step, but a completely new paradigm. Therefore, understanding fail-operational systems used in this domain, is of paramount importance.

Fail-operational behaviour may be almost unknown territory to the automotive domain, but has been standard for decades in others. The reason for this is, that the availability of systems can be critical due to safety or financial reasons. It would be pointless, not to rely on the experiences of other industrial domains. In order to do so, chapter 3 investigates the generic literature on fault tolerance, and standards and literature from the avionics, railway, agricultural and automotive domain. As it can be seen from this list, this thesis concentrates both on related standards and literature. It has been investigated how other domains address the issue of fault tolerance in their related standards. Domain-specific and generic literature (i.e. technical papers and books) was investigated, to get insight into the particular technical solutions of domains having more experience with fail-operational behaviour.

Proven by the *literature research*, fault tolerance has been in the focus of research and development in the past six decades. Starting from the early computers, evolved in the avionic domain, and in availability-critical industrial applications, there are many system, HW and SW topologies available. These are well documented and analysed using the classic methods of reliability engineering (e.g. using Markov analyses). Whether, and how these can be carried over into automotive applications, is clarified in a later chapter (4). There is a significant amount of literature on *fault tolerant avionic systems*. Since high availability is essential for systems in air- and spacecraft, their technical solutions have been evolved over four decades. This is exactly visible in the related literature. The concepts of sensors, controllers, actuators, communication network and even the power supply concepts show, that their availability is regarded as essential. The technical solutions presented in the found papers and books show a very high level of maturity, as product of an evolutionary process over several decades. The literature found on *agricultural- and railway applications* is very scarce, offering insight mainly into research projects. The currently available papers on *fail-operational automotive systems* show the start of this new trend, and present the current concerns about the applicability of typical fault tolerant topologies. But these concerns are not based on a detailed investigation of these architectures from the viewpoint of the [ISO11]. This gap is closed by chapter 4. Another missing link, even if seemingly very basic is the definition of basic, conceptual parameters of fail-operational behaviour. This is also addressed by chapter 4.

The analysis showed, that *functional safety standards* offer very little guidance for fail-operational systems. The [IEC10] (i.e. the generic, domain-independent functional safety standard) offers a framework applicable both for fail-safe and fail-operational systems. The latter is not addressed in particular, but its main concepts can be applied. The standards and guidebooks of the *avionic domain* ([EUR99], [SAE10], [SAE96] and [NAS04]) do not define specific technical requirements, but offer a process and method framework for the entire lifecycle of avionic systems. The *railway* standards ([ÖVE99], [ÖVE12]) explicitly demand fail-safe behaviour. Since railway vehicles are in their ultimate safe state if standing still, this is completely comprehensible. The functional safety standard ([ASI14]) related to *agricultural vehicles* addresses certain fault tolerant HW architectures, but does not define requirements related to this. Regarding the *automotive industry*, the first, brief analysis of the [ISO11] showed, that the [ISO11] is applicable in general, but further extensions may be necessary. This is in focus of chapter 4.

Another aspect that was in the focus of the literature research, is the topic of *dependent failure analyses* (also known as common cause- or common mode analysis). The domains being confronted with this problem (i.e. nuclear industry, avionics) have developed their own methods in the past, integrated into their standard analysis and assessment processes. However, the automotive domain has only the method defined roughly in the [ISO11][part 9]. This issue is addressed in chapter 4, and approaches for

solutions are presented.

One of the key findings of the literature research is, that all of the investigated functional safety standards offer little or no guidance on fail-operational systems. Each standard has to be interpreted in the domain it is applied in. Already available technical solutions for systems, HW and SW need to be investigated from the viewpoint of the particular application, and tailored if necessary. This is exactly the goal of chapter 4: To investigate the [ISO11] *from the viewpoint of fail-operational behaviour*; and to *analyse typical fault tolerant* system/HW and SW *architectures* from the viewpoint of an automotive application. Therefore, in first step, the [ISO11] is analysed in detail, focusing on key concepts and even on detailed requirements. This step is grounded by the author's 8 years of experience from working with the standard, and my activities as Austrian delegate in the ISO standardisation committee. The result is, that the [ISO11] is basically applicable for fail-operational systems, HW and SW. Its core concepts are generic enough to support this. But it is still obvious, that the [ISO11] has been written with keeping fail-safe systems in mind. Thus, chapter 4 defines improvement proposals – mainly additional notes and examples – to make the task of functional safety experts easier, when coping with this new challenge. Since the standard can be applied for fail-operational automotive systems, HW and SW, it is up to the functional safety engineers to tailor and apply already existing design patterns. In order to support this, sections 4.4.3 and 4.4.4 investigate typical fault tolerant topologies from the viewpoint of an automotive application.

In section 4.4.3, typical fault tolerant HW topologies are analysed using state-of-the-art quantitative automotive analysis techniques (i.e. FMEDA and FTA), supported by a simplified failure model. Using these analyses, the HW metrics of the [ISO11] are calculated for these architectures. The key finding of this section is, that for lower ASIL-s, the required level of redundancy is fairly low. It has also been identified, that the two most common fail-operational HW architectures have very similar performance, and their advantages and disadvantages can be judged only from the perspective of a specific application. Another important result of this section is the newly derived HW metric, the RRM, that can be used as simple design guidance instead of the SPFM, LFM and PMHF for ASIL A and B HW.

In section 4.4.4, typical fault tolerant SW architectures are analysed using a structured methodology (SACAM). The SACAM method has been developed to compare architectures of already existing SW, where detailed design information is available. Hence, the first task was to tailor this process to conduct a high level architectural comparison. Using state-of-the-art architectural modelling method, and using criteria specific to automotive SW development, this tailoring was successful. The key result of this comparison was, that there are multiple solutions with similar performance. Therefore, also here (as in case of the HW), it is essential to apply the right topology, depending on the specific problem. Another finding was, that in time critical applications, topologies using checkpoint and restart, are not applicable. It also has to be emphasised, that since SW is prone to systematic faults only, the key factor here is to find the right spots in the architecture, where fault tolerance is *necessary* and *reasonable*.

Section 4.5 addresses the issue identified during the literature study: There is no structured method for automotive dependent failure analyses available. Other domains (e.g. avionics, nuclear plants) have developed their own methods, fitting into their specific analysis landscapes. In this section of the thesis, these methods are investigated in detail, and their core has been transferred into an automotive DFA. The most important result is a generic dependency model, applicable on all abstraction levels (e.g. vehicle, system, HW, SW). Using this dependency model, potential coupling factors between elements, intended to be independent, can be identified in a very structured way. Another advantage of the model is, that it can be applied independently of the other analyses performed on the product (e.g. FTA, FMEA). Hence, this DFA is applicable for all ASIL-s. The dependency model will be taken over in the 2^{nd} edition of the [ISO11][part 9].

As already mentioned before, one of the key findings of chapter 4 is, that complete redundancy is seldom necessary, and that the right architecture depends on the specific application. Due to these reasons, it is of paramount importance, to be able to optimise redundant architectures by selecting the right spots to apply redundancy, to select the right components, and to choose the right topology. The so called RAP, known since the '60-s, addresses this very problem. Invented for the robustness optimisation of computers of the time, the RAP has been in the focus of intense research in the past five decades. Since current automotive products try to avoid redundant architectures (due to the impact on weight, cost and packaging space), the RAP has not been applied for this industrial domain.

In the last chapter (5), a complex mathematical model is defined to support the solution of an *automotive RAP*. In order to do so, the mathematical model deals with the specifics of the automotive RAP: Reliability is modelled using the HW metrics of the [ISO11]; complex failure-effect mapping is required instead of the typical binary state modelling; and the redundant elements are in an m-to-n relation.

The developed *mathematical model* uses a failure model based on linear algebra to dynamically model state-of-the-art automotive analyses techniques (FMEDA and FTA, in particular), and to calculate the SPFM, LFM and PMHF. The defined mathematical modelling of the component selection allows the optimisation algorithm to calculate these three metrics for various architectural candidates: For each subsystem, from one up to three components can be chosen. In case of redundant architectures, the components can be arranged into a dynamic redundant- or a TMR architecture. These two topologies will be the two most wide-spread ones in the automotive industry, based on the results of section 4.4.3. The mathematical model has been verified using a complex ECU architecture, comparing the results calculated by the model implemented in Matlab and manually performed analyses, using Microsoft Excel.

Based on this mathematical model, a straightforward meta-heuristic optimisation algorithm has been defined. This GA is intended to prove the applicability of the mathematical model in a meta-heuristic optimisation algorithm. The developed GA has been verified using a practical system example, consisting of a sensor, an ECU and an actuator. The component choices and constraints have been designed that way, that the optimum can be obtained manually. This way, the performance of the GA can be evaluated. A long term verification run has shown, that the developed optimisation method finds the optimum in more than 90% of the cases, and coming very close in the remaining 10%.

Due to the novelty of the topic in the automotive industry, and due to the very broad application field, there is plenty of room for *further research*. First of all, related to the key attributes of fail-operational behaviour, further research has to derive qualitative and quantitative criteria for the appropriateness of warning and degradation concepts. Especially the allowed performance degradation, and the time to remain operational after the first fault (as derived in chapter 4.3) need further attention. The applicability of the [ISO11] needs to be proven in practical projects. Technical solutions, taken over from other domains need to be evaluated using more complex failure models, and real-life architectural variants. For this, the comparisons shown in section 4.4 provide a solid basis. In particular, the application of fault tolerant SW architecture would be of great interest. Since it is even doubted by some experts, that fault tolerance is necessary for automotive SW. The most promising architectures (i.e. N-version programming and N-self-checking programming) are debated, due to the existing dependencies between the different implementations. Here, further detailing of the new DFA method defined in this thesis can be very helpful. For the DFA methodology, further, application-specific research can derive an exhaustive list of examples for coupling factors. This investigation can be based on field returns, even from other domains.

Since this thesis laid only the basics for an automotive RAP, there are many directions further research can head to. Related to the mathematical model, the dynamic FTA modelling for complex architectures is definitely one of the key aspects. Since not all complex architectures can be reasonably transformed into their parallel-serial equivalents, this can improve the calculation accuracy of the PMHF. The mathematical modelling of dependent failures, especially in case of m-to-n redundancy allocation would be a great step forward towards realistic failure behaviour modelling. Using these extensions, the calculation of the SPFM, LFM and PMHF can be more accurate. Section 5.2.3 provides ideas, how these dependent failures can be modelled. The optimisation algorithm can be optimised and tailored further. The design shown in section 5.2.5 is only intended to verify the applicability of the mathematical model for a typical optimisation algorithm. Further refinement of the algorithm, or even a completely new approach will definitely lead to a more stable solution. From my point of view, PSO would be one of the best candidates. Its higher complexity, and the lower amount of available experience with the method lead to the selection of the GA for this thesis. But its advantages over the GA are promising.

Due to the importance and novelty of the application of fail-operational systems in the automotive domain, it is essential to understand the related key challenges and concepts. Since there is a vast amount of literature from other industrial domains, it is crucial to know how already existing solutions can be carried over. This thesis provides a detailed and thorough investigation of methods, means and

concepts applied in other domains, and extends these to lay the groundwork for the development and optimisation of fail-operational automotive systems.

Glossary

ABS Anti-Lock Brake System. 2, 46
ACO Ant Colony Optimisation. 142
AgPL Agricultural Performance Level. 8, 40
ALMA Architecture Level Modifiability Analysis. 94
ALPSM Architectural Level Prediction of Software Maintenance. 94
APU Auxiliary Power Unit. 38
ASIL Automotive Safety Integrity Level. 7–13, 16, 21, 44–46, 50, 54–57, 59–63, 74, 76, 78, 80, 83, 85–89, 91–93, 96, 97, 99, 101, 103, 104, 107–109, 113, 120, 150, 158
ATAM Architecture Trade-Off Analysis Method. 94, 95

CAN Controller Area Network. 147
COTS Commercial Off The Shelf. 117
CRC Cyclic Redundant Code. 22, 28, 44

DC Diagnostic Coverage. 6, 19, 24, 40, 41, 44, 68, 71–74, 85, 125–128, 132, 134, 135, 138
DFA Dependent Failure Analysis. 107–109, 112, 113, 141, 158, 159

E/E Electric-Electronic. 1, 4, 7, 15, 19, 56, 66, 122
EA Evolutionary Algorithm. 117, 122, 142, 144, 149
ECU Electronic Control Unit. ix, 2, 39, 42, 48, 117, 120, 124, 125, 128, 138, 139, 147, 148, 159
EPS Electric Power Steering. 4, 13, 46, 54–56, 59, 66
ESAAMI Extending SAAM by Integration in the Domain. 94
ESP Electronic Stability Program. 2, 48, 49, 61

FEMM Failure Effect Mapping Matrix. x, 121, 126, 127, 132, 133, 142
FIT Failure In Time. 4, 14
FMEA Failure Mode and Effects Analysis. 16–21, 112, 126, 158
FMEDA Failure Mode, Effects and Diagnostic Analysis. x, 16–19, 21, 65, 66, 68, 69, 71–73, 76, 122, 124, 127–129, 137, 148, 149, 158, 159
FTA Fault Tree Analysis. x, 15–17, 20, 21, 33, 39, 40, 53, 65, 66, 68, 71, 73, 76, 108, 112, 122, 124, 127, 128, 150, 158, 159
FTTI Fault Tolerant Time Interval. 4, 44, 46

GA Genetic Algorithm. vii, x, 142, 143, 147, 149–152, 154, 159

HARA Hazard Analysis and Risk Assessment. 9, 11, 55, 59–63
HFT Hardware Fault Tolerance. 22, 48
HSI Hardware-Software Interface. 63, 64
HW Hardware. vii, 1–4, 6, 7, 9, 10, 13, 14, 17–19, 21–26, 28–30, 32, 33, 38–45, 50, 52–55, 57, 63–66, 68, 85–88, 91–94, 106–109, 113, 114, 117, 119, 120, 122, 125, 138, 153, 154, 157–159

IC Integrated Circuit. 120

k-o-o-n k-out-of-n. 124–126

LFM Latent Fault Metric. ix, 10, 14, 15, 19, 21, 44, 53, 56, 66, 68, 71, 74, 76–81, 83, 84, 86, 87, 113, 121, 122, 124, 127, 129, 144, 147, 149, 150, 158, 159

MPF Multiple Point Fault. 14–16, 50, 57, 66, 68–71, 73, 74, 108, 109, 112, 124, 126–128, 132–135, 137, 138
MPF,D Multiple Point Fault, Detected. 15, 133

MPF,L Multiple Point Fault, Latent. 15, 19, 68, 127, 134, 135
MPF,P Multiple Point Fault, Perceived. 15
MTTF Mean Time to Failure. 40, 52, 53, 116

NMR N-Modular Redundancy. 24, 26, 31, 43
NVRAM Non-Volatile Random Access Memory. 105

PMHF Probabilistic Metric for Random Hardware Failures. ix, 10, 15, 16, 21, 56, 57, 66, 69, 71, 73–78, 80, 81, 83, 84, 86, 87, 121, 122, 124, 127–129, 135, 136, 138, 144, 147–150, 158, 159
PSO Particle Swarm Optimisation. 117, 142, 159

RAM Random Access Memory. 99, 105, 110, 111
RAP Redundancy Allocation Problem. vii, 114, 117–121, 123, 124, 126, 129, 135, 138, 142–144, 147, 149, 151, 153, 154, 158, 159
RAT Ram Air Turbine. 38
RF Residual Fault. 14, 19, 66, 68, 73, 127, 133–135, 138, 148
RPN Risk Priority Number. 17
RRM Replication Rate Metric. 87, 119, 158

SA Simulated Annealing. 117, 142
SAAM Scenario-based Architecture Analysis Method. 94, 95
SAAMCS Scenario-based Architecture Analysis Method for Complex Scenarios. 94
SACAM Software Architecture Comparison Method. 94–96, 158
SACEM Systeme d'aide a la conduite, a l'exploitation et a la maintenance. 42
SBAR Software-Based Architecture Re-engineering. 94
SbW Steer-by-Wire. 2, 3
SIL Safety Integrity Level. 7, 22, 42
SPF Single Point Fault. 14, 19, 50, 57, 66, 68, 73, 74, 76, 85, 121, 124, 126–128, 132–135, 137, 138, 148
SPFM Singe Point Fault Metric. ix, 10, 14, 15, 19, 21, 53, 56, 66, 68, 69, 71, 73–81, 83–87, 113, 121, 122, 124, 127, 129, 144, 147, 149, 150, 158, 159
SPICE Software Process Improvement and Capability Determination. 10, 89
SR Safety Related. 14
SRL Software Reliability Level. 40
SW Software. vii, 1–3, 6, 7, 9, 10, 13, 17, 18, 23, 28–35, 39–45, 48, 52–55, 57, 63, 64, 88–99, 104–110, 113, 114, 117, 120, 157–159
SWC Software Component. 92, 94–107, 111, 120

TMR Triple Modular Redundancy. ix, 24, 26, 33, 34, 42, 46–48, 50, 51, 63, 65, 66, 72–77, 81, 82, 84, 85, 119, 125, 130, 132, 133, 135, 137, 148, 149, 159
TRU Transformer Rectifier Unit. 38
TS Tabu Search Method. 142
TTP Time Triggered Protocol. 36
TTR Triple-Triple Redundancy. 34

UML Unified Modelling Language. 96

VDA Verband der Automobilindustrie. 17

Bibliography

[Aga09] Vikas K. Sharma; Manju Agarwal. "Ant Colony Optimization Approach to Heterogeneous Redundancy in Multi-state Systems with Multi-state Components". In: (2009) (cit. on p. 119).

[Agr11] P. Sinha; V. Agrawal. "Evaluation of Electric-Vehicle Architecture Alternatives". In: (2011) (cit. on pp. 47, 48, 50, 52, 54).

[Ale15] Aldeida Aleti. "Designing Automotive Embedded Systems with Adaptive Genetic Algorithms". In: (2015) (cit. on pp. 120, 123).

[Armar] Klaus Becker; Bernhard Schätz; Christian Buckl; Michael Armbruster. *Deployment Calculation and Analysis for a Fail-Operational Automotive Platform*. unknown year. URL: `http://www6.in.tum.de/Main/Publications/becker2014deployment.pdf` (cit. on p. 49).

[Ash93] Oded Berman; Noushin Ashrafi. "Optimization Models for Reliability of Modular Software Systems". In: (1993) (cit. on pp. 120, 123, 147).

[ASI14] ASI. *Traktoren und Maschinen für die Land- und Forstwirtschaft - Sicherheitsbezogene Teile von Steuerungen*. 2014 (cit. on pp. 9, 41, 42, 161).

[Bal07] Zdzislaw H. Klim; Marek Balazinski. "Methodology for the Common Mode Analysis". In: (2007) (cit. on pp. 39, 40, 110, 111, 114).

[Ber15] Adela Beres. *Impact of increasing availability demands on EPS development*. 2015 (cit. on pp. 47, 50, 51, 53, 54, 62).

[Bla09] H. Blair-Smith. "Space shuttle fault tolerance: Analog and digital teamwork". In: *IEEE* (2009) (cit. on pp. 35, 37, 38).

[Bor08] Kai Borgeest. *Automobilelektronik: Eine Einführung für Ingenieure*. 2008 (cit. on pp. 37, 47, 51, 52).

[Buh09] Indika Meedeniya; Aldeida Aleti; Barbora Buhnova. "Redundancy Allocation in Automotive Systems using Multi-objective Optimisation". In: (2009) (cit. on p. 120).

[Car98] Rick Kazman; Mark Klein; Mario Barbacci; Tom Longstaff; Howard Lipson; Jeromy Carriere. "The Architecture Tradeoff Analysis Method". In: (1998) (cit. on p. 97).

[Cha09] Michel Leeman; Paul Degoul; Pascal Chaussis. "Independence and Non-interference: Two Cardinal Concepts to Develop EE Architectures Hosting Safety-Critical Systems". In: (2009) (cit. on pp. 32, 53, 109, 111).

[Cha12] Tipwimol Sooktip; Naruemon Wattanapongsakorn;David W. Coit; Nida Chatwattanasiri. "Multi-Objective Optimization for k-out-of-n Redundancy Allocation Problem". In: (2012) (cit. on pp. 118, 119).

[Cle96] Rick Kazman; Gregory Abowd; Len Bass; Paul Clemens. "Scenario-Based Analysis of Software Architecture". In: (1996) (cit. on p. 97).

[Cos01] A. Manzone; A. Pincetti; D. De Costantini. "Fault Tolerant Automotive Systems: An Overview". In: (2001) (cit. on pp. 47, 53).

[Dub13] Elena Dubrova. *Fault-Tolerant Design*. 2013 (cit. on pp. 3, 5–7, 23, 24, 27–31, 64).

[Ech07] Klaus Echtle. "Fault-Tolerant Communication in Safety-Relevant Automotive Applications". In: *On-Line Testing Workshop, 2001. Proceedings. Seventh International* (2007) (cit. on pp. 52, 54).

[Ede15] Jan Edel. *Automated cars meet ISO 26262? How can HFT and diversified redundancies make human control areas safe?* 2015 (cit. on pp. 44, 47, 49, 51, 54, 101).

[Ern15] Mischa Moestl; Rolf Ernst. "Cross-Layer Dependency Analysis for Safety-Critical Systems Design". In: (2015) (cit. on pp. 33, 110, 111).

[EUR99] EUROCAE. *Software considerations in airborne*. 1999 (cit. on pp. 34, 161).

[Fab16] Adam Schnellbach; Mario Hirz; Juergen Fabian. "Comparison of Fail-Operational Software Architectures from the Viewpoint of an Automotive Application". In: (2016) (cit. on p. 96).

[Filo4] Zarko Filipovic. *Elektrische Bahnen: Grundlagen, Triebfahrzeuge, Stromversorgung*. 2004 (cit. on p. 43).

[Flü10] Holger Flühr. *Avionik und Flugsicherungstechnik*. 2010 (cit. on pp. 35, 37, 38).

[Gan74] W. C. Gangloff. "Common Mode Failure Analysis". In: (1974) (cit. on pp. 33, 111).

[Gob07] John C. Grebe; Dr. William M. Goble. *FMEDA – Accurate Product Failure Metrics*. 2007 (cit. on p. 20).

[Gui93] C. Hennebert; G. Guiho. "SACEM: A fault tolerant system for train speed control". In: (1993) (cit. on p. 43).

[Ham14] Mostafa Abouei Ardakan; Ali Zeinal Hamadani. ""Reliability optimization of series–parallel systems with mixed redundancy strategy in subsystems"". In: (2014) (cit. on pp. 119, 146).

[Han11] Francisco Rovira Más; Qin Zhang; Alan C. Hansen. *Mechatronics and Intelligent Systems for Off-road Vehicles*. 2011 (cit. on p. 42).

[Has93] A. Hachiga.; K. Akita; Y. Hasegawa. "The design concepts and operational results of fault-tolerant computer systems for the Shinkansen train control". In: (1993) (cit. on p. 43).

[Hireda] Adam Schnellbach; Mario Hirz. "Comparison of hardware metrics of incompletely redundant fail-operational automotive hardware architectures". In: (Submitted) (cit. on p. 66).

[Hiredb] Adam Schnellbach; Mario Hirz. "Mathematical model to support the optimisation of redundant system and hardware architectures for automotive applications". In: (submitted) (cit. on p. 117).

[Hua08] Zhigang Tian; Ming J. Zuo; Hongzhong Huang. "Reliability-Redundancy Allocation for Multi-State Series-Parallel Systems". In: (2008) (cit. on pp. 119, 146).

[Ibr13] H. A. Khorshidi; I. Gunawan; M. Yousef Ibrahim. "Investigation on System Reliability Optimization Based on Classification of Criteria". In: (2013) (cit. on pp. 117, 118, 120, 145).

[IEC10] IEC. *Functional safety of electrical/electronic/programmable electronic safety-related systems*. 2010 (cit. on pp. 3, 7–9, 12, 19–21, 23, 24, 41, 42, 44, 49, 51, 86, 87, 161).

[img16] img-fotki.yandex.ru. *Ram air turbine*. 2016. URL: `https://img-fotki.yandex.ru/get/3606/61625582.77/0_1b2c38_787413fa_orig.jpg` (cit. on p. 39).

[Ise08] R. Isermann. *Mechatronische Systeme*. 2008 (cit. on pp. 6, 23–25, 46, 49).

[ISO11] ISO. *ISO 26262: Road vehicles - Functional Safety*. 2011 (cit. on pp. 1, 3–5, 7–17, 19–23, 34, 35, 42, 44–47, 51, 53–58, 60–64, 67, 69, 75, 77, 85, 87–95, 108–111, 115, 122, 123, 125, 128–130, 151, 161–163).

[Joh09] Roger Johansson. *A fault tolerant architecture for brake-by-wire in railway cars*. 2009 (cit. on pp. 43, 44).

[Joro0] J. Scobie; M. Maiolani; M. Jordan. "A Cost Efficient Fault Tolerant Brake By Wire Architecture". In: (2000) (cit. on pp. 47, 49).

[Jur12] Karol Rastony Juraj Ilavsky. "Comprehensive Technical Safety Analysis Approach Including Common-Cause Failures". In: (2012) (cit. on p. 32).

[Kno12] Jia Huang; Kai Huang; Andreas Raabe; Christian Buckl; Alois Knoll. "Towards FaultTolerant Embedded Systems with Imperfect Fault Detection". In: (2012) (cit. on pp. 119, 128).

[Kru95] Philippe Kruchten. "Architectural Blueprints—The "4+1" View Model of Software Architecture". In: (1995) (cit. on pp. 97, 98).

[Kum13] P. Babul Saheb; K. Subbarao; S.Phani Kumar. "A Survey on Voting Algorithms Used In Safety Critical Systems". In: (2013) (cit. on p. 25).

[Kuooo] V. Rajendra Prasad; Way Kuo. “Reliability Optimization of Coherent Systems”. In: (2000) (cit. on pp. 118, 130, 131).

[Kuo77] Frank A. Tillman; Ching-Lai Hwang; Way Kuo. “Optimization Techniques for System Reliability with Redundancy-A Review”. In: (1977) (cit. on pp. 117, 118).

[Levo1] Gregory Levitin. “Redundancy Optimization for Multi-State System with Fixed Resource-Requirements and Unreliable Sources”. In: (2001) (cit. on p. 119).

[Li12] Yong Wang; Lin Li. “Heterogeneous Redundancy Allocation for Series-Parallel Multi-State Systems Using Hybrid Particle Swarm Optimization and Local Search”. In: (2012) (cit. on pp. 119, 145).

[Lisoo] G. Levitin; A. Lisnianski. “Survivability maximization for vulnerable multi-state systems with bridge topology”. In: (2000) (cit. on p. 119).

[Liso1] G. Levitin; A. Lisnianski. “Structure optimization of multi-state system with two failure modes”. In: (2001) (cit. on p. 119).

[Lyo92] Tom Sadeghi; Arthur Lyons. “Fault Tolerant EHA Architectures”. In: (1992) (cit. on p. 67).

[Mal14] M. Ruggeri; C. Ferraresi; L. Dariz; G. Malaguti. “A High Functional Safety Performance Level Machine Controller for a Medium Size Agricultural Tractor”. In: *SAE* (2014) (cit. on p. 43).

[Mat16] Mathworks. *MatLab homepage*. 2016. URL: `https://de.mathworks.com/products/matlab/?requestedDomain=www.mathworks.com` (cit. on p. 150).

[Mis99] Suprasad V. Amari; Joanne Bechta Dugan; Ravindra B. Misra. “Optimal Reliability off Systems Subject To Imperfect Fault-Coverage”. In: (1999) (cit. on pp. 119, 128).

[Mos95] Peter J. Rutledge; Ali Mosleh. “Dependent-Failures in Spacecraft: Root Causes, Coupling Factors, Defenses, and Design Implications”. In: (1995) (cit. on pp. 40, 111).

[Nako7] Junichi Onishi; Sakuo Kimura; Ross J. W. James; Yuji Nakagawa. “Solving the Redundancy Allocation Problem With a Mix of Components Using the Improved Surrogate Constraint Method”. In: (2007) (cit. on p. 118).

[NASo4] NASA. *NASA Software Safety Guidebook*. 2004 (cit. on pp. 34, 35, 161).

[Neh10] Malakondaiah Naidu; Suresh Gopalakrishnan; Thomas W. Nehl. “Fault-Tolerant Permanent Magnet Motor Drive Topologies for Automotive X-By-Wire Systems”. In: (2010) (cit. on pp. 49–51, 54).

[Neno7] Philipp Nenninger. “Vernetzung verteilter sicherheitsrelevanter Systeme im Kraftfahrzeug”. Doctoral Thesis. 2007 (cit. on pp. 6, 27, 41, 52, 54).

[Ng11] O. Tannous; L. Xing; P. Rui; M. Xie; S. H. Ng. “Redundancy Allocation for Series-Parallel Warm-Standby Systems”. In: (2011) (cit. on p. 117).

[Oli99] Rodrigo de Oliveira Chaves. “Genetic Algorithms Applied on Route Optimization”. In: (1999) (cit. on pp. 146, 147).

[Ort15] Mauricio Matamoros; Jesus Savage; Jorge Luis Ortega-Arjona. “A comparison of two Software Architectures for General Purpose Mobile Service Robots”. In: (2015) (cit. on p. 97).

[ÖVE12] ÖVE. *EN 50128 - Telekommunikationstechnik, Signaltechnik und Datenverarbeitungssysteme - Software für Eisenbahnsteuerungs- und -Überwachungssysteme*. 2012 (cit. on pp. 43, 161).

[ÖVE99] ÖVE. *EN 50126 - Bahnanwendungen - Spezifikation und Nachweis der Zuverlässigkeit, Verfügbarkeit, Instandhaltbarkeit und Sicherheit (RAMS)*. 1999 (cit. on pp. 43, 161).

[Pec15] Carlos Henrique Mariano; Carlo Alessandro Zanetti Pece. “Simulation Optimization Approach to Solve a Complex Multi-objective Redundancy Allocation Problem”. In: (2015) (cit. on pp. 117, 119).

[Pen12] Hang Zhou; Yuanjian Yang; Hong-Zhong Huang; Yu Liu; Weiwen Peng. “Reliability Analysis of A Satellite System Considering Common Cause Failures”. In: (2012) (cit. on pp. 40, 111).

[Potwn] Frédéric Pothon. *DO-178C/ED-12C versus DO-178B/ED-12B – Changes and Improvements*. year unknown (cit. on p. 34).

[Praoo] Way Kuo; V. Rajendra Prasad. "An Annotated Overview of System-Reliability Optimization". In: (2000) (cit. on pp. 117–120, 123, 145).

[QMC15] VDA QMC. *Automotive SPICE - Process Reference Model, Process Assessment Model, Version 3.0*. 2015. URL: http://www.automotivespice.com/fileadmin/software-download/Automotive_SPICE_PAM_30.pdf (cit. on p. 91).

[Red97] V. Ravi; B.S.N. Murty; P.J. Reddya. "Nonequilibrium Simulated Annealing-Algorithm Applied to Reliability Optimization of Complex Systems". In: (1997) (cit. on pp. 118, 130, 131, 147).

[Rei14] Konrad Reif. *Elektronik in der Fahrzeugtechnik. Hardware, Software, Systeme und Projektmanagement*. 2014 (cit. on pp. 47, 49).

[SAE10] SAE. *Guidelines for Development of Civil Aircraft and Systems*. 2010 (cit. on pp. 33, 34, 161).

[SAE96] SAE. *GUIDELINES AND METHODS FOR CONDUCTING THE SAFETY ASSESSMENT PROCESS ON CIVIL AIRBORNE SYSTEMS AND EQUIPMENT*. 1996 (cit. on pp. 33, 34, 39, 161).

[Sch13] Adam Schnellbach. *Latent Faults in the ISO 26262*. 2013 (cit. on pp. 69, 74).

[Sch14] Adam Schnellbach. *Dependent Failure Analysis - An abstract approach*. 2014 (cit. on p. 108).

[Sch15] Adam Schnellbach. "Basics of fail-operational systems". In: *CTI ISO 26262 conference, Rochester, MI* (2015) (cit. on pp. 55, 56).

[Sha09] Soureh Latif Shabgahi. "Modelling and Analysis of Decision makers in Fault -Tolerant Systems". Bachelor Thesis. 2009 (cit. on p. 25).

[Sha13] Harish Garg; S. P. Sharma. "Reliability-redundancy Allocation Problem of a Pharmeceutical Plant". In: (2013) (cit. on p. 120).

[Sin11] Purnendu Sinha. "Architectural design and reliability analysis of a fail-operational brake-by-wire system from ISO 26262 perspectives". In: *Reliability Engineering and System Safety 96* (2011) (cit. on pp. 47, 49, 51–54).

[Smi04] Yun-Chia Liang; Alice E. Smith. "Ant Colony Optimization Algorithm for the Redundancy Allocation Problem (RAP)". In: (2004) (cit. on pp. 119, 120, 123).

[Smi96] David W. Coit; Alice E. Smith. "Reliability Optimization of Series-Parallel Systems Using a Genetic Algorithm". In: (1996) (cit. on pp. 146, 147).

[Stö02] Rolf Isermann; Ralf Schwarz; Stefan Stölzl. "Fault-tolerant drive-by-wire systems". In: *IEEE Control Systems Magazine* (2002) (cit. on pp. 46, 48, 51, 52).

[Sum15] Anil Patidar; Ugrasen Suman. "A Survey on Software Architecture Evaluation Methods". In: (2015) (cit. on pp. 96, 97).

[Sup04] Member; Hoang Pham; Glenn Dill Suprasad V. Amari. "Optimal Design of k-out-of-n:G Subsystems Subjected to Imperfect Fault-Coverage". In: (2004) (cit. on pp. 119, 128).

[Tha05] G. Zayaraz; P. Thambidurai. "Software Architecture Selection Framework Based on Quality Attributes". In: (2005) (cit. on p. 97).

[Too09] David Wyatt; Mike Tooley. *Aircraft Electrical and Electronic Systems: Principles, Maintenance and Operation*. 2009 (cit. on pp. 35, 38, 39).

[Ulb14] Peter Matthias Ulbrich. "Ganzheitliche Fehlertoleranz in eingebetteten Softwaresystemen". 2014 (cit. on p. 28).

[Utk95] Sergey V. Gurov; Lev V. Utkin. "Cold Standby Systems with Imperfect and Noninstantaneous Switch-Over Mechanism". In: (1995) (cit. on p. 119).

[Val08] Yamille del Valle; Ganesh Kumar Venayagamoorthy; Salman Mohagheghi; Jean-Carlos Hernandez; Ronald G. Harley. "Particle Swarm Optimization: Basic Concepts, Variants and Applications in Power Systems". In: (2008) (cit. on p. 145).

[VDA14] VDA. *Automotive SPICE*. 2014. URL: http://vda-qmc.de/en/certification/automotive-spice/ (cit. on p. 11).

[VDI04] VDI. *Design methodology for mechatronic systems*. 2004 (cit. on pp. 65, 90).

[Ver03] Christoph Stoermer; Felix Bachmann; Chris Verhoef. "SACAM: The Software Architecture Comparison Analysis Method". In: (2003) (cit. on pp. 96, 97).

[Vil14] Christopher Temple; Antonio Vilela. *Fehlertolerante Systeme im Fahrzeug – von "fail-safe" zu "fail-operational"*. 2014. URL: http://www.elektroniknet.de/automotive/assistenzsysteme/artikel/110612/1/ (cit. on pp. 46–48, 54).

[Wan07] Way Kuo; Rui Wan. "Recent Advances in Optimal Reliability Allocation". In: (2007) (cit. on pp. 117–119, 145).

[Wan12] Yantao Song; Bingsen Wang. "A Hybrid Electric Vehicle Powertrain with Fault-Tolerant Capability". In: *IEEE* (2012) (cit. on p. 50).

[Wer12] Martin Werdich. *FMEA - Einführung und Moderation*. 2012 (cit. on pp. 17, 21).

[Win86] Christopher J. Dale; Alan Winterbottom. "Optimal Allocation of Effort to Improve System Reliability". In: (1986) (cit. on pp. 118, 130, 131).

[Wys03] R. Debouk; T. Fuhrman; J. Wysocki. "Architecture of By-Wire Systems Design Elements and Comparative Methodology". In: (2003) (cit. on pp. 47–52, 54).

[Xie13] Wei Wang; Junlin Xiong; Min Xie. "The Robust Redundancy Allocation Problem of Series-Parallel Systems". In: (2013) (cit. on p. 119).

[Yao09] Zai Wang; Tianshi Chen; Ke Tang; Xin Yao. "Multi-objective Approach to Redundancy Allocation Problem in Parallel-series Systems". In: (2009) (cit. on pp. 120, 147).

[Yao97] Xin Yao. "Global Optimisation by Evolutionary Algorithms". In: (1997) (cit. on p. 146).

[Yeh96] Y. C. Yeh. "Triple-triple redundant Boeing 777 primary flight computer". In: *IEEE* (1996) (cit. on pp. 35, 38, 58).

[Yeo02] Heeseok Choi; Keunhyuk Yeom. "An Approach to Software Architecture Evaluation with the 4+1 View Model of Architecture". In: (2002) (cit. on pp. 97, 98).

[Yum93] Jae-Hwan Kim; Bong-Jin Yum. "A Heuristic Method for Solving Redundancy Optimization Problems in Complex Systems". In: (1993) (cit. on pp. 118, 130, 131).