life.augmented

Functional Safety in Electronic Systems: Principles and Applications

Alessandro Bastoni

Functional Safety Expert

STMicroelectronics

Lesson #3

Systematic Capability theory, including V model and preliminaries on SW and tools

Summary:

- Safety lifecycle
- V-model
- Notes on requirements
- Software development
- Tools assessment

life.augmented

How to deal with systematic failures

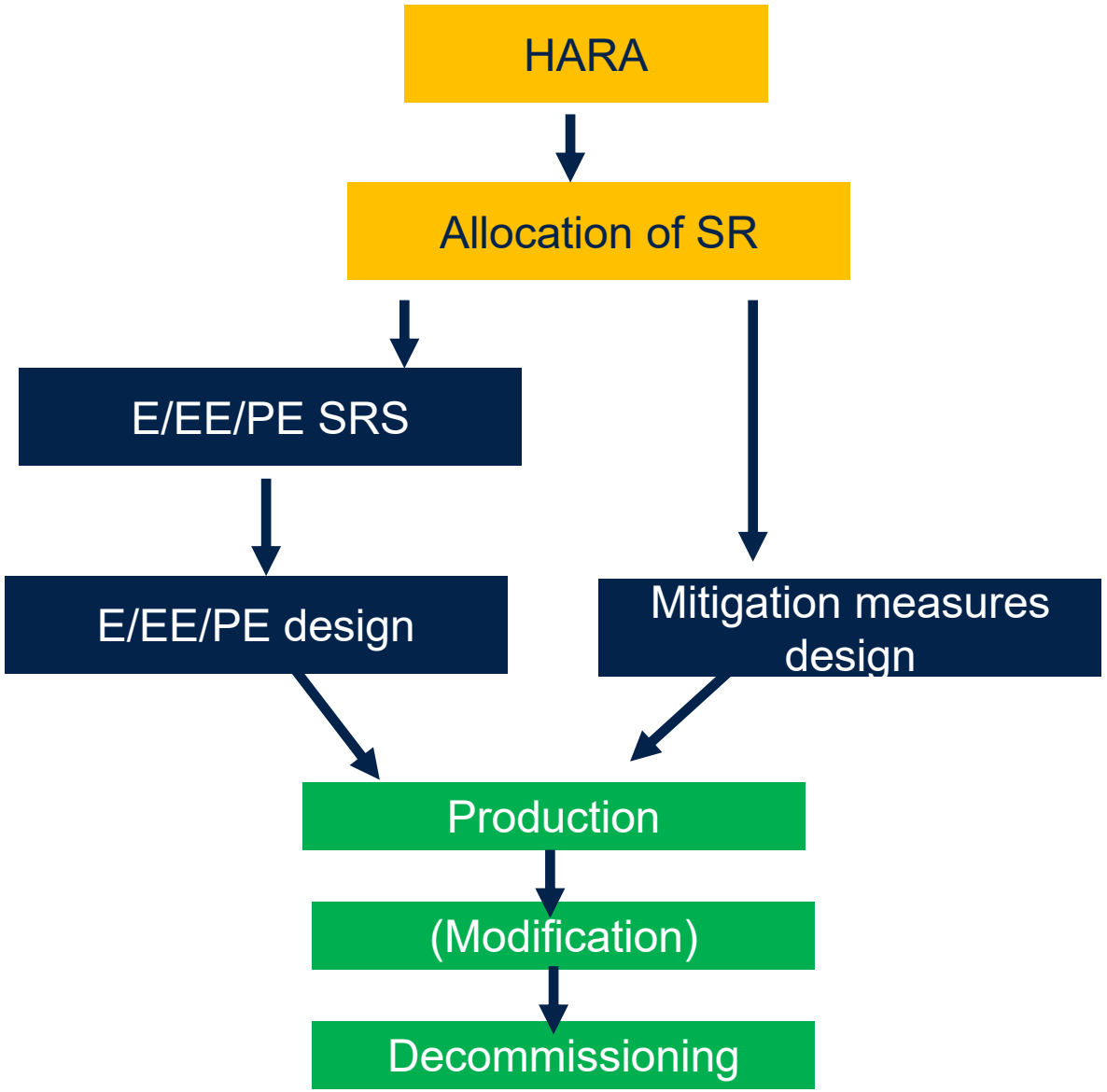
While most part of the safety standards requires to mitigate/consider systematic failures, related issue is more challenging for risk-based standards which set a specific safety integrity level also for systematic.

Two main patterns exist:

- Formal **safety lifecycle** (based on strict application of development rules tailored depending on the safety integrity level)
- **Proven in use argument**, based on evidence of stability/absence of systematic defects over time
- The two patterns can apply to embedded software and software tools as well.

Safety lifecycle

Each safety standard defines its specific safety lifecycle; the general structure is mainly the same:



Verification

Documentation

Safety Assessment

Verification & Validation

Verification is the process of confirming, through examination and objective evidence, that a product, system, or component meets its specified requirements. It answers the question: "Are we building the product right?:

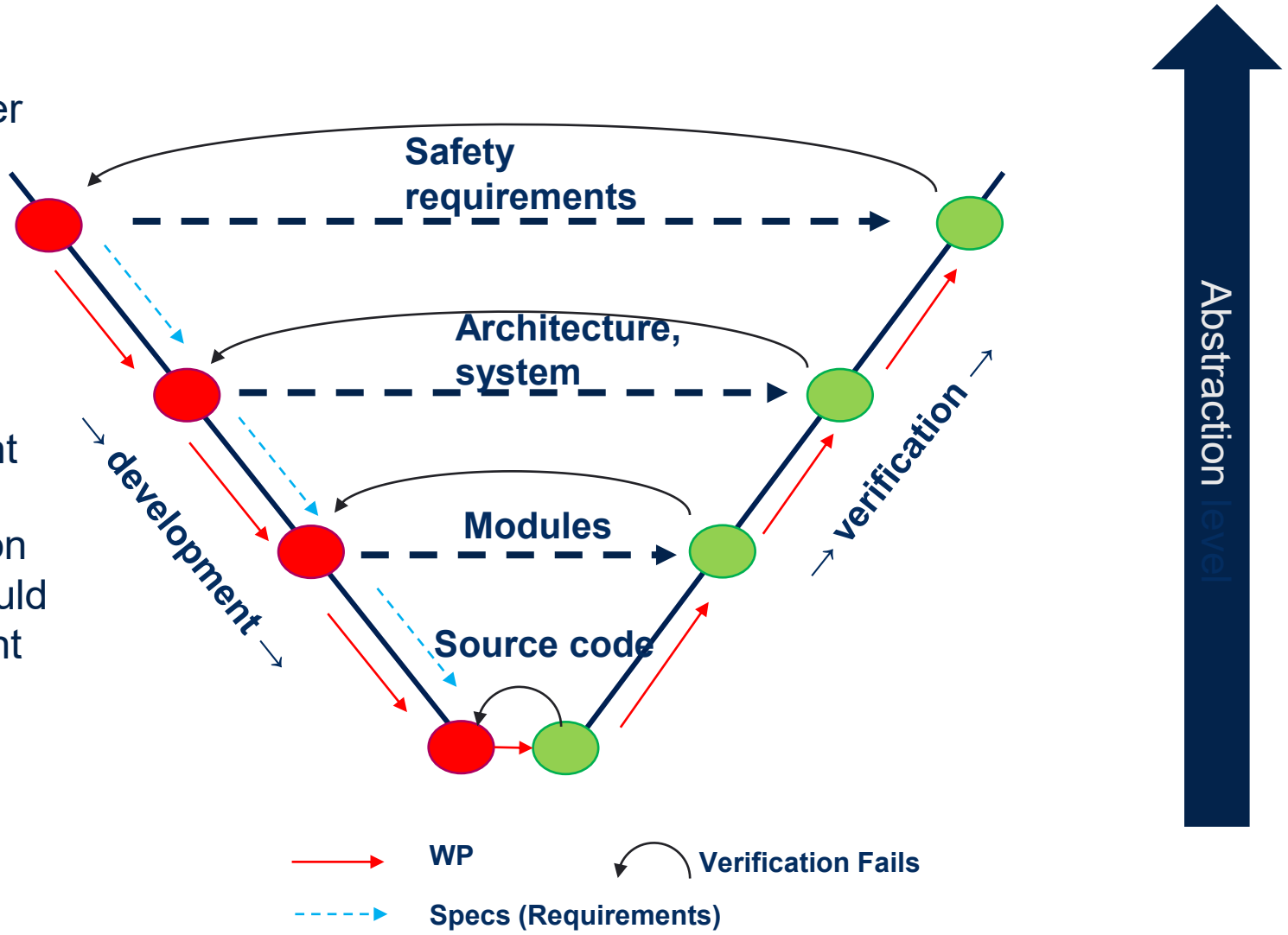
Validation is the process of confirming, through examination and objective evidence, that the product fulfills the requirements for its specific, intended use in the real world. It answers the question: "Are we building the right product?

Aspect	Verification	Validation
Purpose	Confirm requirements are correctly implemented	Confirm product meets user needs and intended use
Focus	Conformance to specifications	Fitness for purpose
Typical Activities	Reviews, inspections, unit/component testing	System-level testing, user acceptance testing

V-model characteristics (IEC 61508-3, ISO 26262-6)

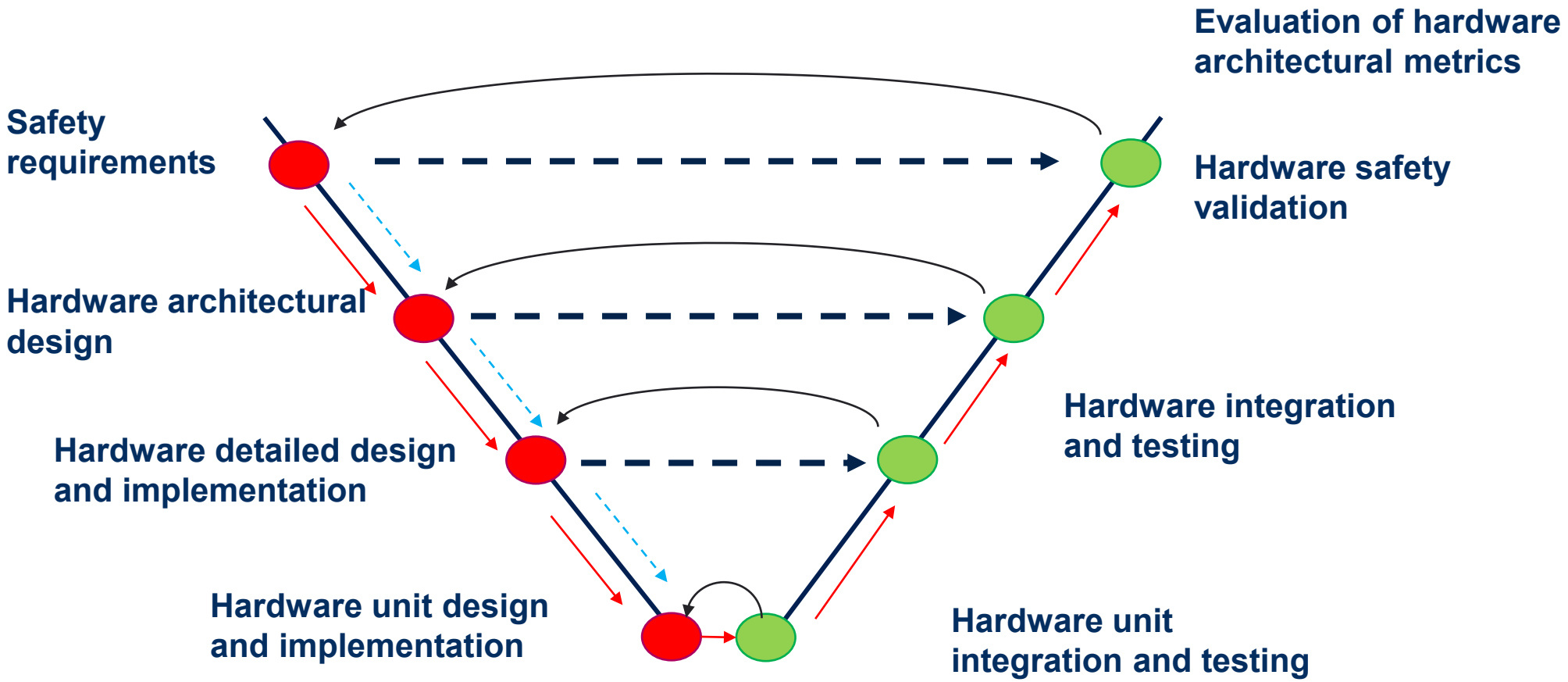
Advantages

- Top-down approach is forced
- Requirement-based client/server model between phases
- Inputs/outputs between phases (WP) well defined
- Testing specified at same abstraction level of development
- Non-compliance after verification is managed hierarchically (it could impacts the related development phases)
- Traceability bonded inside

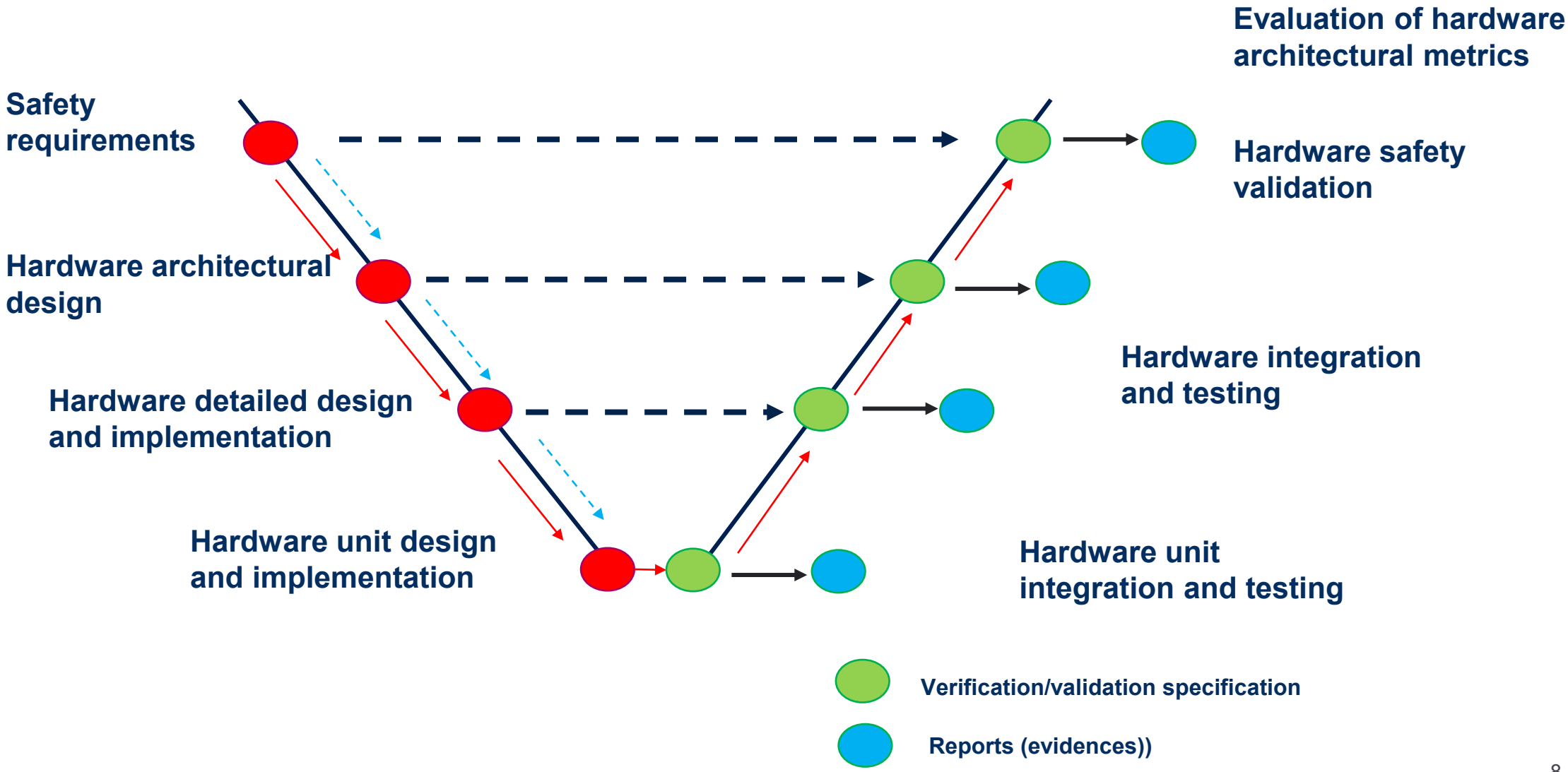


life.augmented

V-model for hardware development (ISO 26262-5) - phases



V-model for hardware development (ISO 26262-5) - documents



About traceability

IEC61508 V-model requires Forward and Backward traceability among several specification and verification requirements sets. HR for SIL3/4, just R for SIL1/2. Among other techniques, traceability is a very good asset for safety and quality in the development of the software:

Forward traceability: checking that a requirement is adequately addressed in later lifecycle stages.

→ Pro: key asset to proper evaluate a Change Impact depending on high level requirements update/change

Backward traceability: checking that every implementation decision is clearly justified by some requirement.

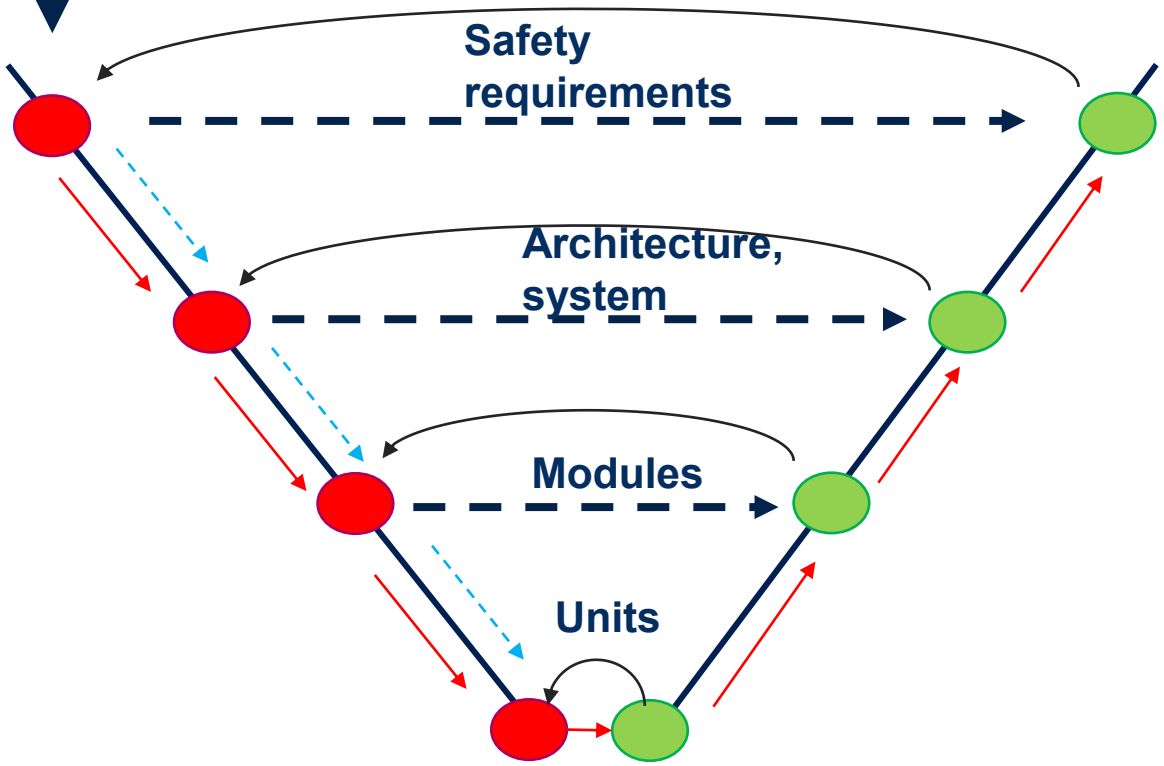
→ Pro: decrease probability of unused or unjustified functions/hw parts

WARNING: adding traceability lately in a project will turn out in a nightmare!

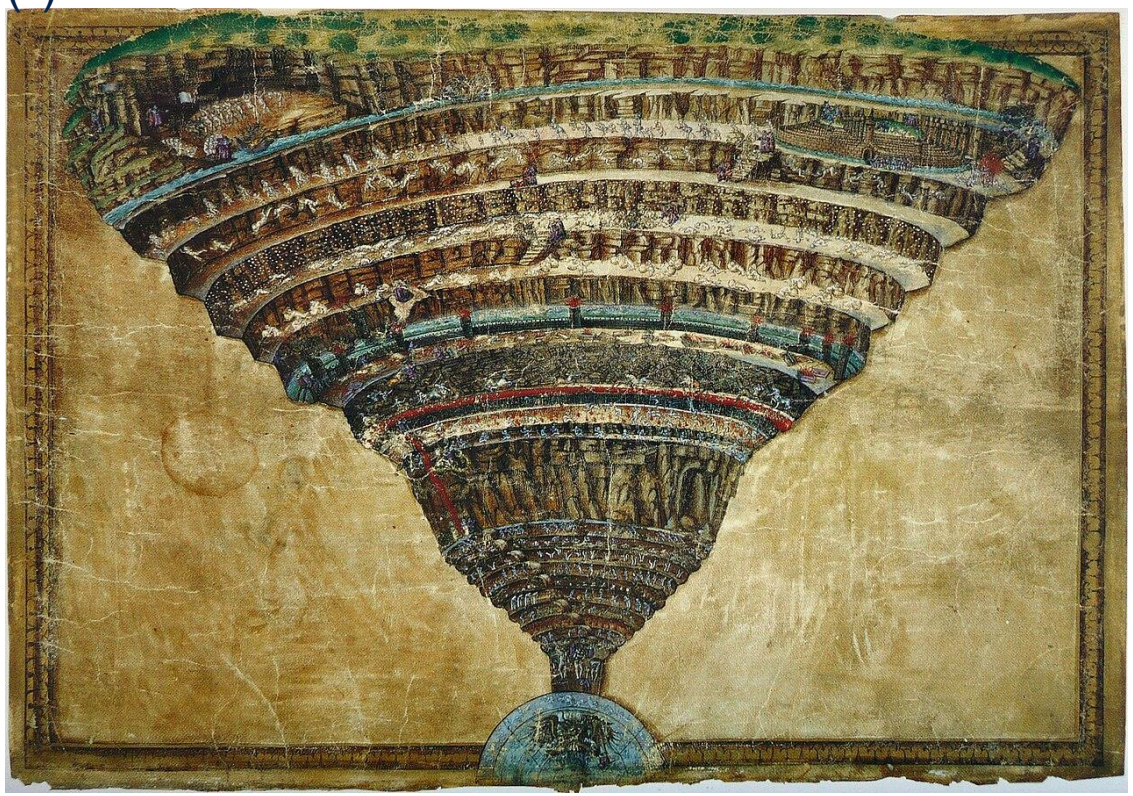
life.augmented

The V-model trap

V-model Devil is here – see reference [R3]



(*)



Dante's Devil is here

(*) *La voragine infernale, Sandro Botticelli (Source: Wikipedia)*

life.augmented

SOTIF (Safety Of The Intended Functionality)

ISO/PAS 21448 — provides guidelines for ensuring safety in advanced driver assistance systems (ADAS) and autonomous vehicles. SOTIF (Safety Of The Intended Functionality)

SOTIF addresses safety risks arising from the intended functionalities of a system, especially when no faults or failures are present.

SOTIF focuses on hazards caused by performance limitations, environmental conditions, or misuse, beyond traditional fault-based safety. It move the focus on incomplete system specifications, which by nature are not intercepted by the V-model.

SOTIF purpose is to identify and mitigate risks related to correct system behavior that can still lead to unsafe situations. It complements ISO26262 by covering scenarios where the system behaves as designed but still poses safety risks.

.

Methods tailoring in V model

The formal V model prescribes for each phase lists of recommended methods.
Recommendations are included in tables, and ranked this way:

HR/++ the technique or measure is highly recommended for related safety integrity level; If not used, then the rationale behind not using it should be detailed and agreed with the assessor.

R/+: the technique or measure is recommended for this safety integrity level as a lower

recommendation to a HR/++ recommendation or as an additional safety margin measure.

NR: the technique or measure is positively not recommended for this safety integrity level.

Methods tailoring in V model

Table notional example.

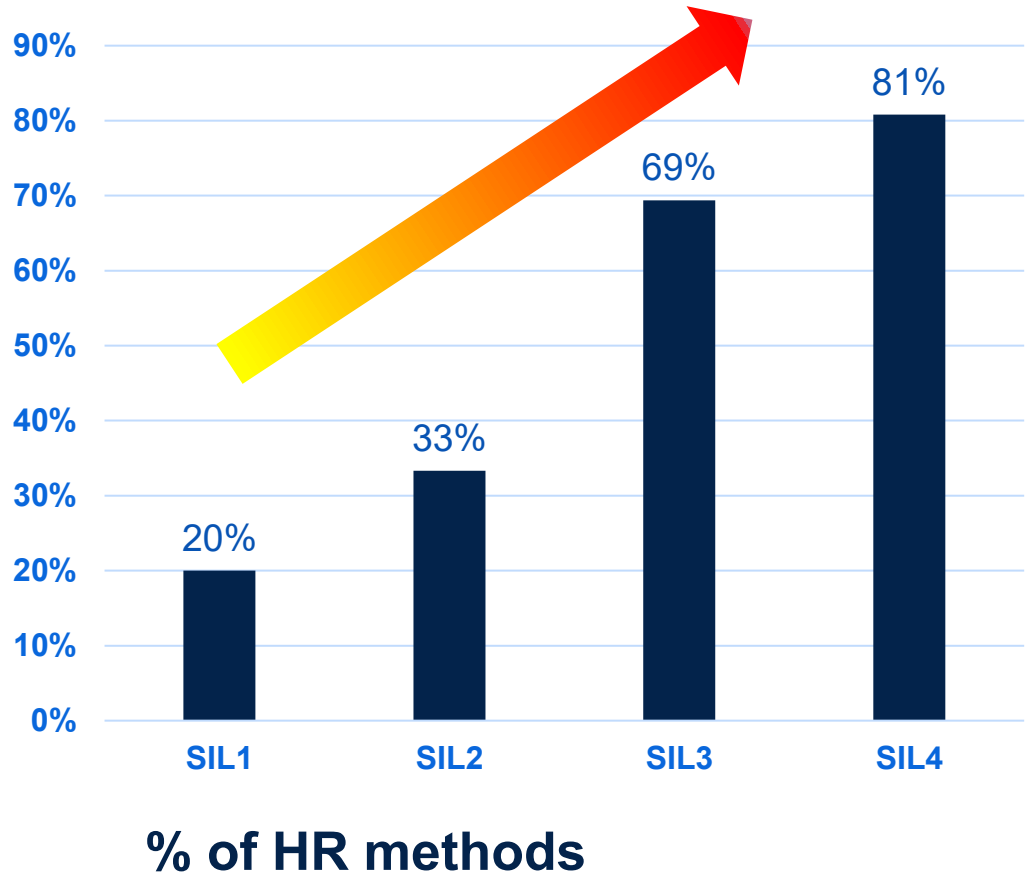
		SIL 1	SIL 2	SIL 3	SIL 4
1	Measures against voltage breakdown, voltage variations, overvoltage, low voltage	R	HR	HR	HR
2	Increase of interference immunity	R	HR	HR	HR
3	Spatial separation of multiple lines	HR	HR	HR	HR
4	…	…	…	…	…
5	…	…	…	…	…

life.augmented

IEC 61508-3 tables contents vs SIL/SC

Request of method application strongly depends on SIL level:

“Generic recommendations” turn into “detailed prescriptions” on low side of the V (e.g. software metrics coverages, numeric approach)



Requirements

A **requirement** is formally defined as:

a documented statement

describing a condition or capability

that a system, product, or service must satisfy

to meet stakeholder needs or constraints.

This definition is consistent across major standards in systems and software engineering.

Requirements are the basic pillar of the formal V-model.

What is really required?

life.augmented

Requirements qualities

ISO/IEC/IEEE 29148:2018 (Requirements Engineering) explicitly describes quality attributes that requirements should satisfy, including:

Atomicity: Requirement expresses a single need or capability (to avoid ambiguity and complexity).

Unambiguity: Requirement has only one interpretation.

Completeness: Requirement includes all necessary information.

Consistency: Requirement does not conflict with others.

Verifiability: Requirement can be verified by inspection, analysis, test, or demonstration.

Modifiability: Requirement can be changed without introducing errors.

Traceability: Requirement can be traced to its origin and related artifacts.

Correctness: Requirement accurately reflects the stakeholder need.

Assured by the V-model

life.augmented

How to write (good) requirements

Semi-formal methods:

- Structured Natural Language (SNL) – ("The system shall [action] [object] [under conditions].“)
- State diagrams
- Decision Tables
- UML use cases,

Formal methods:

- Petri nets
- Alloy
- …

life.augmented

Structured Natural Language (SNL) - example

Based on consistent template e.g.: The [actor] shall [action] [object] [under conditions].

Guidelines:

- Use Clear and Precise Language
- Use Active Voice
- Use Consistent Terminology (glossary)
- Avoid Negative Statements
- Use Singular Nouns and Consistent Numbering
- Limit Use of Pronouns

Proven in use (*) argument

Based on the demonstration, supported by operational experience over a specific, extended period of time, that the probability of unknown systematic failures is low enough for the target safety integrity level.

Main issues related to this approach are:

- ❑ Requires the presence of a credible monitoring procedure for on-field failures
- ❑ In some cases, hardly linkable to a specific systematic integrity levels
- ❑ Usually, suitable just for very simple components, because configuration changes can invalidate the argument
- ❑ Dependency on multiple factors including component manufacturing process

(*) different names can be found on safety standards ecosystem

Tools assessment

IEC 61508 and ISO 26262 defines a structured approach to assess the confidence in software tools used in the development and verification of safety-related systems. This ensures that tools do not introduce or fail to detect errors that could compromise functional safety.

The structure is similar, tools are assessed according to their capability to (negatively) affect the implementation or verification on the safety function. Then, an evaluation is done on the possibility to later identify those introduced issues.

The result of the assessment is the prescription of specific requirements on the tools, possibly requiring:

- Adoption of “certified” tools explicitly developed according a safety lifecycle
- Adoption of additional mitigation measures (e.g. tools output comparison, proven in use argument, etc.)

Tools assessment – IEC61508

A software off-line support tool is a software application that assists with one or more phases of the software development lifecycle but does not have any direct influence on the safety-related system during its runtime. These tools are categorized into three classes based on their interaction with the safety-related system:

T1 Tools: these tools do not produce any outputs that directly or indirectly affect the executable code (including data) of the safety-related system.

T2 Tools: these tools support the testing or verification of the design or executable code. While errors in these tools may cause defects to go undetected, they cannot introduce errors into the executable software itself.

T3 Tools: these tools generate outputs that directly or indirectly contribute to the executable code of the :safety-related system.

Tools assessment – IEC61508

The selection of the tools shall be justified. Then, following requirements apply:

- . **Documentation:**
  All T2 and T3 tools must have clear specifications or documentation detailing their behavior and usage constraints.
- **Assessment:**
  Evaluate T2 and T3 tools to understand how much they are relied upon and identify possible failure modes that could impact the executable software. Apply mitigation if needed.
- **Conformance Evidence (T3 only):**
  Provide evidence that T3 tools meet their specifications, based on successful past use and/or formal validation.

Lesson #4

Safety analysis methods (FMEDA/FTA/DFA/ETA/Markov)

Summar\y:

- **FMEA**
- **FMEDA**
- **FTA**
- **DFA**
- **Markov analysis**

life.augmented

Principles of FMEA (Failure Modes and Effects Analysis)

Proactively identify potential failure modes in products, processes, or systems to improve reliability and safety.

Key Elements:

- Failure Mode: Specific way a part or process can fail (e.g., crack, short circuit).
- Effect: Impact of the failure on the system or user (e.g., loss of function, safety hazard).
- Cause: Root cause or trigger of the failure mode (e.g., material defect, human error).

Use Risk Priority Number (RPN) or similar metrics based on:

- Severity (S): How serious the effect is.
- Occurrence (O): Likelihood of failure happening.
- Detection (D): Probability of detecting the failure before it reaches the customer

Iterative Process: update FMEA regularly during design changes, production, and field feedback to maintain effectiveness.

Example of FMEA

Notional example (process)

.

Process Step	Potential Failure Mode	Potential Effects	Potential Causes	(S)	(O)	(D)	RPN	Recommended Actions
Soldering components	Poor solder joint	Device malfunction or failure	Insufficient solder, operator error	9	4	3	108	Train operators, improve soldering process controls
Component placement	Misaligned components	Short circuit or open circuit	Misalignment during placement	8	3	4	96	Use automated placement machines, add visual inspection
Testing final assembly	Incomplete functional testing	Defective devices shipped to customer	Test procedure incomplete	10	2	5	100	Standardize test procedures, add test checklist

- **Severity (S):** Impact on system or user (1-10 scale, 10=most severe).
- **Occurrence (O):** Likelihood of failure (1-10 scale, 10= most frequent).
- **Detection (D):** Likelihood failure will be detected before release (1-10 scale, 1=highly detected).
- **RPN:** RPN=S×O×D*RPN=S×O×D*; higher values indicate higher priority

.

life.augmented

Principles of FMEDA (Failure Modes, Effects and Diagnostics Analysis)

Similar to FMEA with following specifics:

- Includes indication of Diagnostics (aimed to mitigate/detect the failures)
- Quantitative: get rid of RPN in favor of failure rates computations.
- It provides an overall result in terms of DC and SFF (SPF)
- For each failure mode line, includes information on associated fault model (to correctly compute failure distribution)
- Hw only: it cannot be applied to processes and software

Key topic is the failure distribution problem (how to associate individual failure mode to the system failure rate)

Principles of Fault Tree Analysis (FTA)

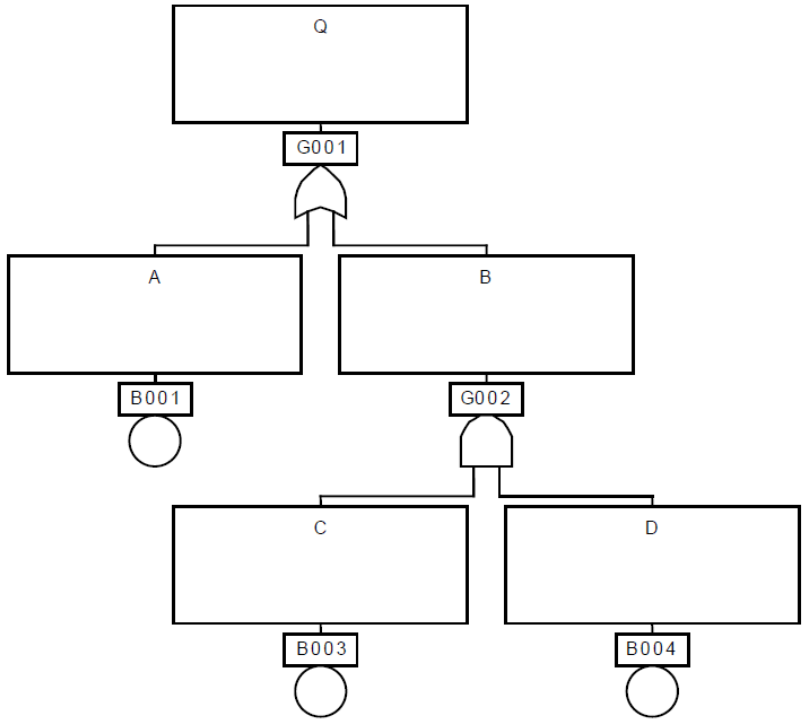
FTA is a top-down, deductive analytical method used to identify the causes of system-level failures.

Purpose: to systematically analyze how combinations of basic faults can lead to a critical undesired event (the top event).

Key Features:

- Starts with a defined top event (system failure or hazard).
- Uses logical gates (AND, OR) to map relationships between faults.
- Top-down analysis flows down until a basic event (root cause) is found
- Helps visualize failure pathways and their interdependencies

Principles of Fault Tree Analysis (FTA)



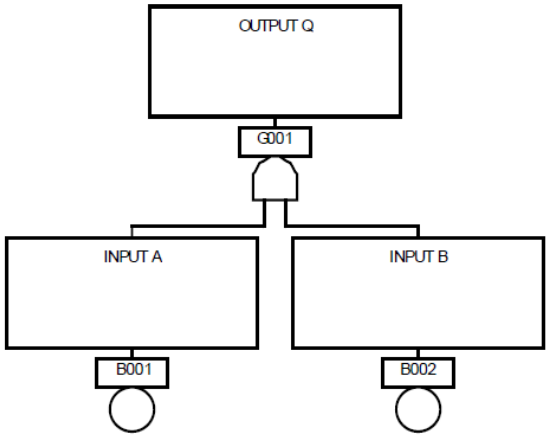


Figure 4-5. The AND-Gate

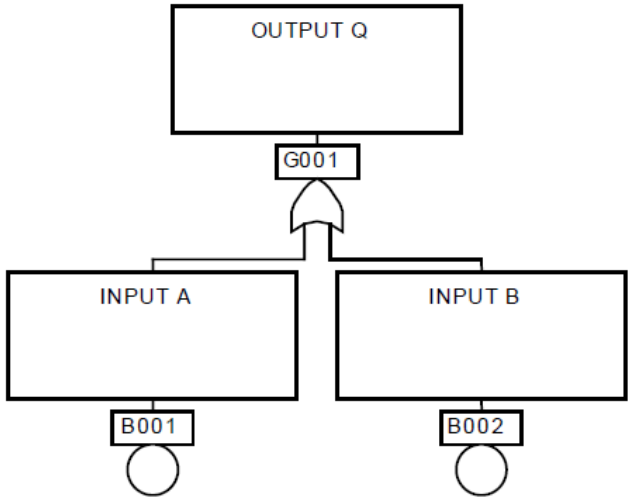


Figure 4-2. The OR-Gate

For full explanations refer to reference document [R4], section 4.1 Symbology—The Building Blocks of the Fault Tree

life.augmented

Principles of Fault Tree Analysis (FTA)

Quick recap of main FTA rules/specificities

Complete-the-Gate Rule: all inputs to a particular gate should be completely defined before further analysis of any one of them is undertaken.

No Gate-to-Gate Rule: gate inputs should be properly defined fault events, and gates should not be directly connected to other gates.

Minimum Cut Set: is the smallest combination of basic faults that can cause the top-level failure (top event). It represents a minimal set of component failures leading to system failure and one of the major benefits of underrunning a FTA analysis in your system..

Dependent Failure Analysis (DFA)

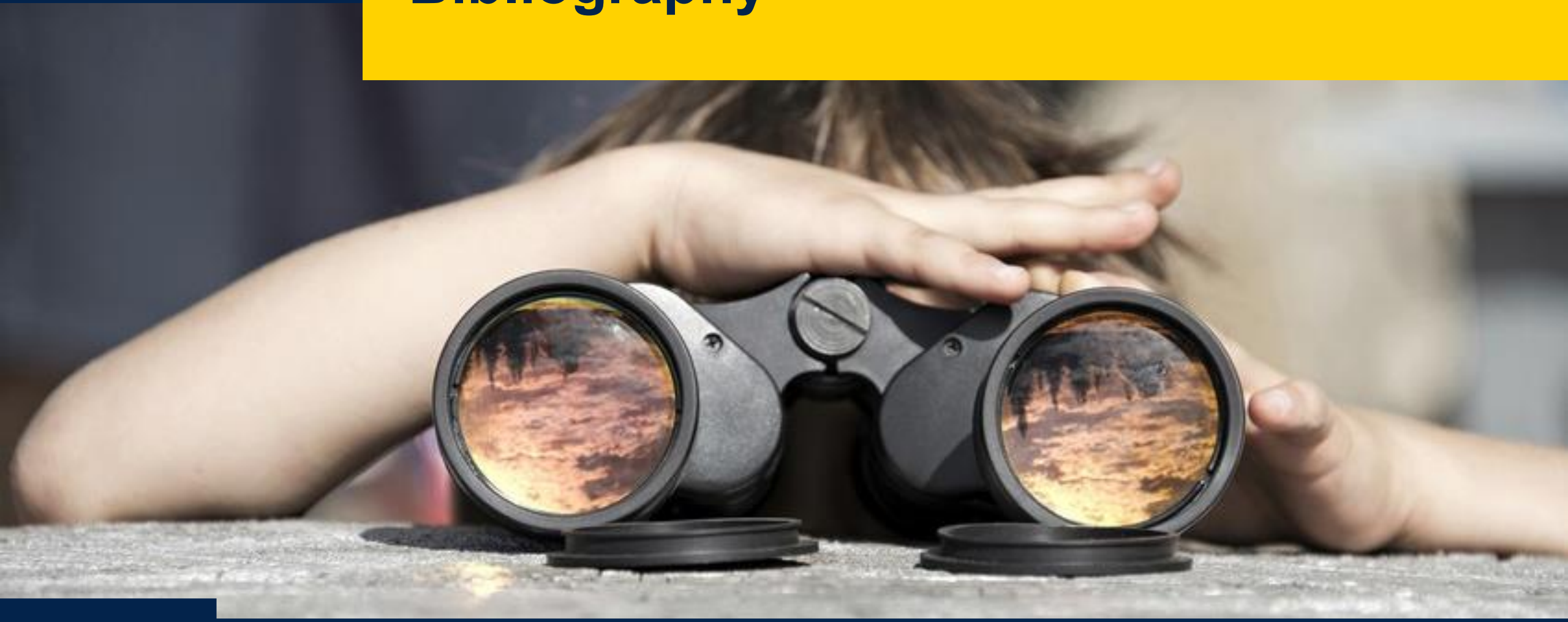
Purpose is to identify and analyze failures that are not independent but occur due to a common cause or dependency between components...

It is explored in detail mainly in ISO26262

It leverages on other safety analysis techniques (mainly FTA and sometimes FMEA), with focus on dependent failures finding.

Safety standards helps the research with specific guidance tables (typical topics to be analyzed in searching such DFA)

Bibliography



life.augmented

Reference documents

[R1]: Microelectronics Reliability:Physics-of-Failure Based Modeling and Lifetime Evaluation - Jet Propulsion LaboratoryCalifornia Institute of Technology Pasadena, California

[R2]: : Semiconductor Reliability Handbook – Renesas Electronics, Rev.2.50 Jan. 2017

[R3]: ExoMars 2016 - Schiaparelli Anomaly Inquiry (ESA) downloaded from https://exploration.esa.int/web/mars/-/59176-exomars-2016-schiaparelli-anomaly-inquiry

[R4]: Fault Tree Handbook with Aerospace Applications - NASA Office of Safety and Mission Assurance, V 1.1 , 2002

[R5]: open FTA software can be found on the wed, e.g. https://www.fault-tree-analysis.com/free-fault-tree-analysis-software, or check for OpenFTA download

Thank you

life.augmented