Advanced Reliable Embedded Systems

ARES

Andi Ming / Quelldateien

Table of contents

Safety, Risiko Management

> **!** Safety | Safety-Critical
>
> **Safety** is defined as preventing harm to humans/environment, while **safety-critical systems** ensure this property.

> **!** Risk | Safety-Integrity
>
> **Risk** is a measure of the likelihood, and the consequences, of a hazardous event.
> **Safety-integrity** is a measure of the likelihood of the safety system correctly performing its task.

Terms

- **Hazard**: A situation in which there is actual or potential danger for people or environment.
- **Accident**: Unintended event harming people or environment.
- **Incident**: Unintended event which does not harm, but has the potential to do so.
- **Risk**: Likelihood of hazard occurrence, and the likely consequences. Risk $=$ Severity $\times$ Probability
- **Fault**: Defect in system. Can be **random** or **systematic.**
- **Error**: Deviation from the required operation of the system.
- **System Failure**: Occures when system fails to perform its required function.
- **Casualities** (Kausalitäten): The presence of a fault *may* lead to an error, which *may* lead to a system failure, which *may* lead to an accident.

Requirements

Requirements give a system the properties of **integrity and dependability**.

This demands: (1) **Safety**, (2) **Reliability**, (3) **Availability**, (4) **Maintainability**.

> ⚠ **Conflicts**
>
> In general, the various requirements to a system are conflicting among themselves.

Process (Iterative!)

1. Identification of hazards associated with the system
2. Classification the hazards
3. Determination of methods to deal with hazards
4. Assignment of reliability and availability requirements
5. Determination of safety integrity level
6. Specification of development method appropriate to integrity level

Verification, Validation & Certification (V&V&C)

- **Verification**: Confirms system meets specifications
- **Validation**: Ensures fitness for intended purpose
- **Certification**: Obtains regulatory approval through evidence documentation
- Key distinction example: Medical device passing lab tests (verification) but failing clinical trials (validation)

Hazard & Risk Analysis

- **Hazard identification** methods:
  - *FMEA (Failure mode and effects analysis)*: Analyzes component failure effects on ultimate consequences.

Failure Mode Effects Analysis

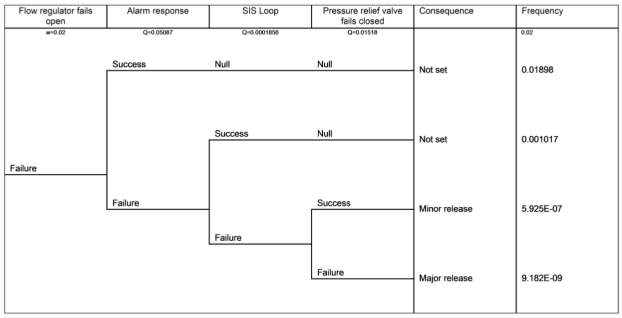
System Description: Landing Gear
Operation Mode: Flight - Level 2

Item Number	Item Description	Function	FM Id.	Failure Mode	Local Effect	Next Higher Effects	End Effects	Sev.	Detection Method	Compensating Provisions	Remarks
1.1.1	Main Pump	Provides pressure when requested by Pilot Command	1	Fails to operate	No effect during this phase	No effect during this phase	No effect	IV	Indication to pilot	None	
			2	Untimely operation	Untimely hydraulic pressure in Main Hydraulic Generation Assembly	Untimely hydraulic pressure from Main Hydraulic Generation Assembly to Actuator Assembly	Untimely extension of Landing Gear	I	Indication to pilot	None	
1.1.2	Check Valve (Main)	Prevents reverse flow	1	Stucked closed	Loss of fluid flow through the Main Hydraulic Generation Assembly check valve	No effect during this phase	No effect	IV	Indication to pilot	None	
			2	Stucked open	Permits fluid flow through the main assy check valve when not required	No effect during this phase	No effect	IV	Undetected	None	

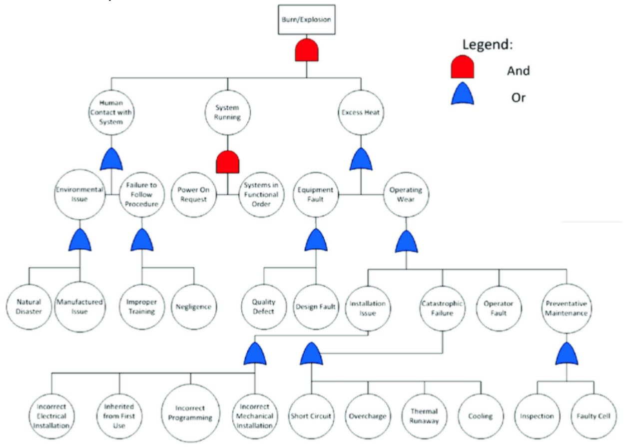
  - *HAZOP (Hazard and operability studies)*: Uses guidewords to detect operational deviations.

Guide Word	Deviation	Causes	Consequences	Action
NO	No cooling	Cooling water valve malfunction	Temperature increase in reactor	Install high temperature alarm (TAH)
REVERSE	Reverse cooling flow	Failure of water source resulting in backward flow	Less cooling, possible runaway reaction	Install check valve
MORE	More cooling flow	Control valve failure, operator fails to take action on alarm	Too much cooling, reactor cool	Instruct operators on procedures
AS WELL AS	Reactor product in coils	More pressure in reactor	Off-spec product	Check maintenance procedures and schedules
OTHER THAN	Another material besides cooling water	Water source contaminated	May be cooling ineffective and effect on the reaction	If less cooling, TAH will detect. If detected, isolate water source. Back up water source?

  - *ETA (Event tree analysis)*: Model effects from starting point forward to determine possible consequences.



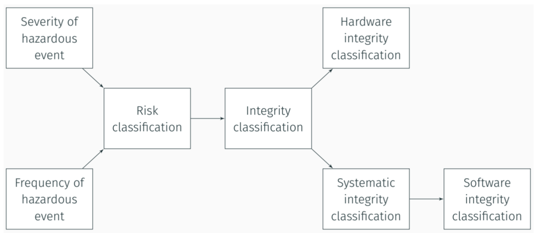
  - *FTA (Fault tree analysis)*: Identify hazards and determine their possible causes.



Risk Analysis

- **Risk classification** combines severity (catastrophic/negligible) and frequency (frequent/incredible). Risks are categorized as intolerable (I) to negligible (IV).

$$\text{Risk} = \text{Severity} \times \text{Probability}$$



Severity of Hazardous Event

Category	Definition
Catastrophic	Multiple deaths
Critical	Single death, and/or multiple severe injuries or severe occupational illnesses
Marginal	Single severe injury or occupational illness, and/or multiple minor injuries or minor occupational illnesses
Negligible	Single minor injury or minor occupational illness at most

Frequency of Hazardous Event

Category	Definition	Range (events per hour)
Frequent	Many times in system lifetime	$> 1 \times 10^{-3}$
Probable	Several times in system lifetime	1×10^{-3} to 1×10^{-4}
Occasional	Once in system lifetime	1×10^{-4} to 1×10^{-5}
Remote	Unlikely in system lifetime	1×10^{-5} to 1×10^{-6}
Improbable	Very unlikely to occur	1×10^{-6} to 1×10^{-7}
Incredible	Cannot believe that it could occur	$< 1 \times 10^{-7}$

Risk Classification

	Consequence			
Frequency	Catastrophic	Critical	Marginal	Negligible
Frequent	I	I	I	II
Probable	I	I	II	III
Occasional	I	II	III	III
Remote	II	III	III	IV
Improbable	III	III	IV	IV
Incredible	IV	IV	IV	IV

I Intolerable
II Undesirable, tolerable only if risk reduction is impractical
III Tolerable
IV Negligible

Integrity Classification

ALARP-Rule: Class II & III is only acceptable if it is **A**s **L**ow **A**s **R**easonably **P**racticable

Risk can be reduces by safety features. Achieved reduction depends upon integrity of these features.

Safety integrity is how likely a safety system is to perform its job correctly, under all conditions, and for the required time.

Safety Integrity Levels (SIL)

Safety Integrity Level	Continuous mode (prob. of dangerous failure per year)	Demand mode (prob. of failure to perform on demand)
4	$\geq 1 \times 10^{-5}$ to 1×10^{-4}	$\geq 1 \times 10^{-5}$ to 1×10^{-4}
3	$\geq 1 \times 10^{-4}$ to 1×10^{-3}	$\geq 1 \times 10^{-4}$ to 1×10^{-3}
2	$\geq 1 \times 10^{-3}$ to 1×10^{-2}	$\geq 1 \times 10^{-3}$ to 1×10^{-2}
1	$\geq 1 \times 10^{-2}$ to 1×10^{-1}	$\geq 1 \times 10^{-2}$ to 1×10^{-1}

Hardware Integrity

Hardware integrity is that part of the safety integrity relating to dangerous *random* hardware failures.

Systematic Integrity

Systematic integrity is that part of the safety integrity relating to dangerous *systematic* failures.

Software Integrity

Software integrity is that part of the safety integrity relating to dangerous *software* failures.

Achieving Safety Integrity

The process involves iterative design stages and layered fault mitigation strategies to meet safety-critical system requirements.

Core Design Process

1. **Abstraction**: Identify essential system properties
2. **Decomposition**: Break systems into analyzable components
3. **Elaboration**: Add implementation details
4. **Decision**: Select optimal design alternatives

Fault Mitigation Strategies

Four complementary approaches:

1. **Avoidance**: Prevent faults during design phase
2. **Removal**: Eliminate faults through testing/reviews
3. **Detection**: Identify faults during operation
4. **Tolerance**: Maintain functionality despite faults

Fault Characteristics

Category	Types	Examples
Nature	Random (HW) vs Systematic	α-particle errors
Duration	Permanent/Transient/Intermittent	Broken chip vs c
Extent	Localized vs Global	Single sensor vs

Hardware Fault Tolerance:

- **Static** (TMR/NMR): Mask faults via majority voting (3-5 modules)
- **Dynamic**: Detect & switch to backups
- **Hybrid**: Combine masking + reconfiguration

Software Fault Tolerance:

- **N-version Programming**: Parallel diverse implementations (Airbus/Shuttle)
- **Recovery Blocks**: Fallback modules with acceptance tests
- **Information**: Additional data (parity / checksum)
- **Temporal**: Repeat calculations

Key Challenges

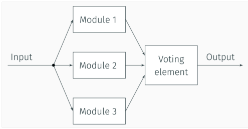
- Common-mode failures require **diversity** in:
  - Implementation methods
  - Programming languages
  - Hardware platforms
- **Systematic faults (spec/design errors)** are harder to mitigate than random HW faults
- No single technique provides complete protection

Critical Insight: Achieving safety integrity requires combining multiple fault mitigation strategies through iterative design refinement, as perfect fault elimination is impossible in complex systems.

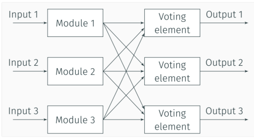
Detection: Functional checking, Consistency checking, Signal comparison, Checking pairs, Information redundancy, Instruction monitoring, Loopback testing, Watchdog timers, Bus monitoring, Power supply monitoring

Fault Tolerance

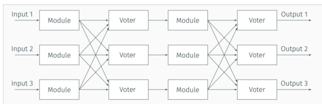
- **Redundancy strategies**:
  - *TMR (Triple Modular Redundancy):* Voting systems mask faults via majority logic (3 modules)



+ simple
+ prevents from failure of a single component, i.e. single-point failure
− leaves input and voter as sources for single-point failures
− does not prevent from systematic failures
− voter is dependable

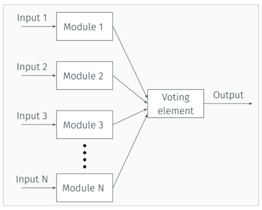


+ all outputs are correct in case a single module fails
− more components required
− no protection against simultaneous failure of two or more modules
− does not prevent from systematic failures



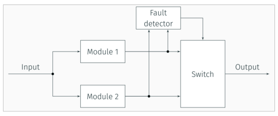
+ allows a single module to fail at each level
+ allows a single voter to fail at each level
− no protection against simultaneous failure of two or more units at same level
− does not prevent from systematic failure

  - *NMR*: Voting systems mask faults via majority logic (3-5 modules). Allows for $\frac{N-1}{2}$ modules to fail



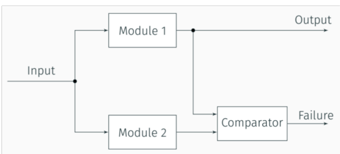
+ allows (N−1) /2 modules to fail
− higher complexity of voter
− higher cost, size, power consumption

  - *Dynamic redundancy*: Switches to backup modules after fault detection.



+ allows one modules to fail
+ gives indication of fault
− fault detector required (single-point failure!)
± either hot standby or cold standby possible
± can be extended to N modules

  - *Self checking pair*: The outputs are compared and give indication of failure



+ simple, reliable
+ gives indication of fault
− no redundancy

- **Diversity**: Combines different implementations/languages to avoid common-mode failures.
- **Software fault tolerance**: Uses *N-version programming* (parallel implementations) or *recovery blocks* (fallback modules with acceptance tests).

Reliability

> **Reliability**
>
> Reliability R is the probability of a component or system functioning correctly over time $R(t)$. Describing a statistical behaviour of a component or system.
> Given: Period of time, set of operating conditions.
>
> $$R(t) = \frac{n(t)}{N}$$
>
> with $n(t)$ number of working elements, and N number of original elements.

Unreliability

Probability $Q(t)$ that a system will **not** function over a given period of time. $Q(t) + R(t) = 1$

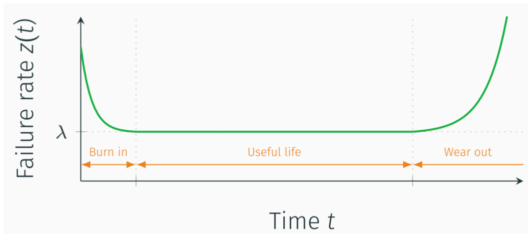
$$Q(t) = \frac{n_f(t)}{N} = 1 - R(t)$$

with $n_f(t)$ number of failed components at time t.

Failure Rate

The rate $z(t)$ at which a device fails

$$z(t) = \frac{1}{n(t)} \cdot \underbrace{\frac{dn_f(t)}{dt}}_{Failures}$$



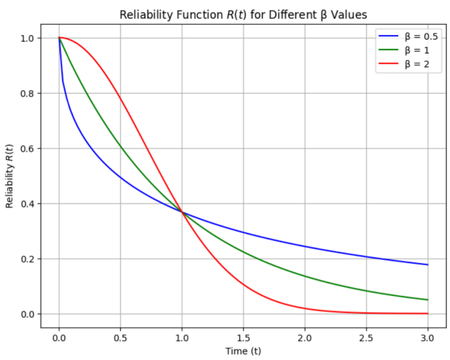
For a constant **failure rate** $z(t) = \lambda$ the probability of a system working correctly decreases exponentially

$$R(t) = e^{-\lambda t}$$

Time-Variant Failure Rates

Software failures which are systematic and correctable the failure rate decreases with time. **Weibull** distribution

$$R(t) = e^{-\left(\frac{t}{\eta}\right)^{\beta}}$$



Mean Times

Mean Time to Failure

Expected time before first failure

$$MTTF = \int_0^\infty R(t)dt = \frac{1}{\lambda}$$

Reliability

With $\lambda = 0.001$ failure/h $MTTF = 1000h$.
But at $t = 1000h$ the reliability is only $R(t) \approx 0.37$ (chance for running at 1000h mark is 37%)

Mean Time to Repair

Time to repair given by repairability μ

$$MTTR = \frac{1}{\mu}$$

Mean Time Between Failures

$$MTBF = MTTF + MTTR$$

Failure in Time

Number of failures expected in $1 \times 10^9 h$ of cumulative operation hours

$$FIT = 1 \times 10^9 \cdot \frac{1}{MTBF}$$

Reliability Modelling

Series Systems

Failure of **any component fails**

$$R(t) = R_1(t) \cdot R_2(t) \cdots R_N(t) = \prod_{i=1}^{N} R_i(t)$$

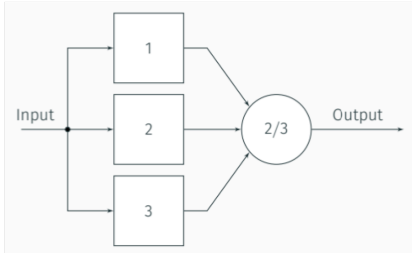
$$\lambda = \lambda_1 + \lambda_2 + \cdots + \lambda_N = \sum_{i=1}^{N} \lambda_i$$

Parallel Systems

System operational as long as **one component is functioning**

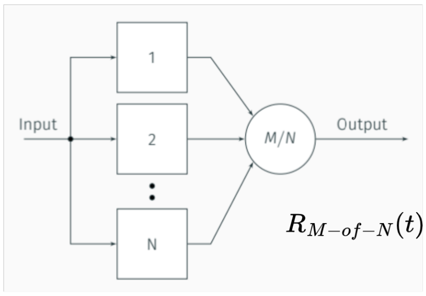
$$R(t) = 1 - Q(t) = 1 - \prod_{i=1}^{N}(1 - R_i(t))$$

Redundancy

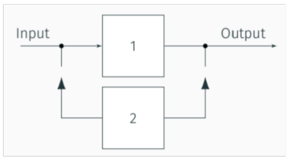


$$R_1 = R_2 = R_3 = R_m$$

$$R(t) = 3R_m^2(t) - 2R_m^3(t)$$



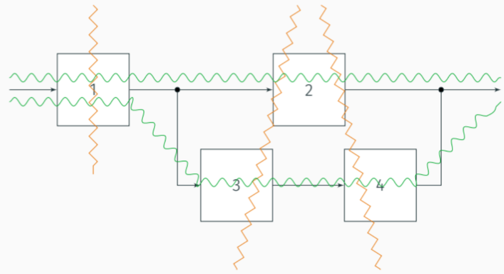
$$R_{M-of-N}(t) = \sum_{i=0}^{N-M} \frac{N!}{(N-i)!i!} \cdot R_m^{N-}$$



If module 1 fails, module 2 is activated

Fault Coverage C_m

$$R(t) = R_m(t) + (1 - R_m(t))C_m R_m(t)$$



cut : sets of simultaneous failures leading to a system failure
tie : sets of working modules guaranteeing a working system

Boundaries

$$1 - \sum_{j=1}^{N_C} \prod_{i=1}^{n_j}(1 - R_i(t)) \le R(t) \le \sum_{j=1}^{N_T} \prod_{i=1}^{n_j} R_i(t)$$

Reliability Prediction

There is extensive (usually MIL std.) literature but often with lots of unknown variables.

Reliability Assessment

How to proof that a system fails less then once in 1×10^9 hour (i.e. $\approx$ 100 000 year) of operation?
Trust the development techniques.

Software Safety

Common faults:

Coding faults, logical errors within calculations, numeric under- and overflows, stack under- and overflows, range under- and overflows (arrays!), uninitialised variables, unintended side effects, truncation by casts, rounding effects, memory leaks, ...

Capablity Maturity Model

CMM Level	Focus	Defects / 1000 LOC
1	None	7.5
2	Project Mngt.	6.2
3	Software Eng.	4.7
4	Quality Processs	2.3
5	Cont. Improvement	1.1

LOC: Lines of Code

Formal Methods

Apply mathematically rigorous techniques for the specification development and verification of the software and hardware systems.

Examples

- B-Method – abstract machine notation, became Event-B, Rodin as tool
- Esterel – synchronous programming language, generates C code
- Z notation – specification language
- SPIN – model checker basing on Promela language
- SPARK – refinement of Ada also possible to have a program in Ada and submodules in spark (more save)
- Frama-C – basing on ACSL specification language,

Frama-C

- Frama-C is an open source framework
- core to read C files and build abstract syntax trees
- set of plug-ins to do static analysis and to annotate syntax trees
- plug-ins can collaborate, i.e. use another plug-in
- plug-ins programmed in OCaml language
- major plug-ins: EVA & WP
- ACSL (ANSI/ISO C Specification Lanugage) for annotations by C comments /* @ ... */

Evolved Value Analysis (EVA)

Computes variation domains for variables

```c
int abs(int x) {
   if (x < 0)
      return -x;
   else
      return x;
}
```

```
pipistrellus@N0007494:~$ frama-c -eva example_1.c -main abs
[kernel] Parsing example_1.c (with preprocessing)
[eva] Analyzing a complete application starting at abs
[eva:initial-state] Values of globals at initialization

[eva:alarm] example_1.c:3: Warning: signed overflow. assert -x ≤ 2147483647;
[eva] ====== VALUES COMPUTED ======
[eva:final-states] Values at end of function abs:
  __retres ∈ [0..2147483647]
```

Weakest Precondition (WP)

Proofing certain properties

```c
/*@ ensures \result == (a+b)/2;
  @ assigns \nothing;
  @ */
int mean(int a, int b) {
   return (a+b)/2;
}
```

```
pipistrellus@N0007494:~$ frama-c -wp example_4.c
[kernel] Parsing example_4.c (with preprocessing)
[wp] Warning: Missing RTE guards
[wp] 2 goals scheduled
[wp] Proved goals:    4 / 4
  Terminating:     1
  Unreachable:     1
  Qed:             2
```

Methoden, Prozesse, Zuverlässigkeit

Formale Methoden - Frama

“Rauswurf”

Block Cypher

ECC and Encryption Modes

Hash Functions

Key Distribution

Reliability

Vulnerability

Boot

Memory

Execution

Fault Handling