Safety-Critical Systems

Prof. Dr. Jan Peleska
Universität Bremen — TZI
Dr. Ing. Cornelia Zahlten
Verified Systems International GmbH

Overview

1. The Notion of Dependability
2. Safety-Related Standards and V-Models
3. Modelling Safety-Critical Systems
4. Hazard Analysis and Risk Assessment
5. Design Criteria for Safety-Critical Systems
6. Validation, Verification and Test of Safety-Critical Systems

Overview

1. The Notion of Dependability
2. Safety-Related Standards and V-Models
3. Modelling Safety-Critical Systems
4. Hazard Analysis and Risk Assessment
5. Design Criteria for Safety-Critical Systems
6. Validation, Verification and Test of Safety-Critical Systems

Dependability – General Definition

Our objective: Development and verification of safety-critical reactive systems that continuously interact with their environment and control (parts of) the environment in order to perform the user-required services and enforce the required safety properties.

- **Dependability:** The trustworthiness of a computer system such that reliance can be justifiably placed on the service it delivers.
- **Service:** System behaviour as it is perceived by its users.
- **User:** Another system (human or physical) interacting with the target system.

Dependability Attributes

- **Availability:** readiness for usage
- **Reliability:** continuity of service
- **Safety:** avoidance of catastrophic consequences on the environment
- **Security:** prevention of unauthorised access and/or handling of information

  Security attributes:

  - confidentiality
  - integrity
  - availability

Dependability – Faults – Errors – Failures

- **Failure:** delivered service no longer complies with specification
- **Error:** part of the system state which is likely to lead to subsequent failure
- **Fault:** cause of an error

Dependability – Fault-Tolerance versus Fault-Avoidance

Classification of methods achieving dependability:

- **Fault-Tolerance:** provide a service complying with the specification in spite of faults
- **Fault-Avoidance (Fault-Prevention):** a-priori prevention of fault occurrence or introduction
- **Fault-Removal:** reduce the presence (number, seriousness) of faults
- **Fault-Forecasting:** estimate the present number, the future incidence, and the consequence of faults

Our Favourite Approach: Fault-Tolerance on HW level — Fault-Avoidance on specification and SW level !

Dependability — Safety versus Liveness Properties

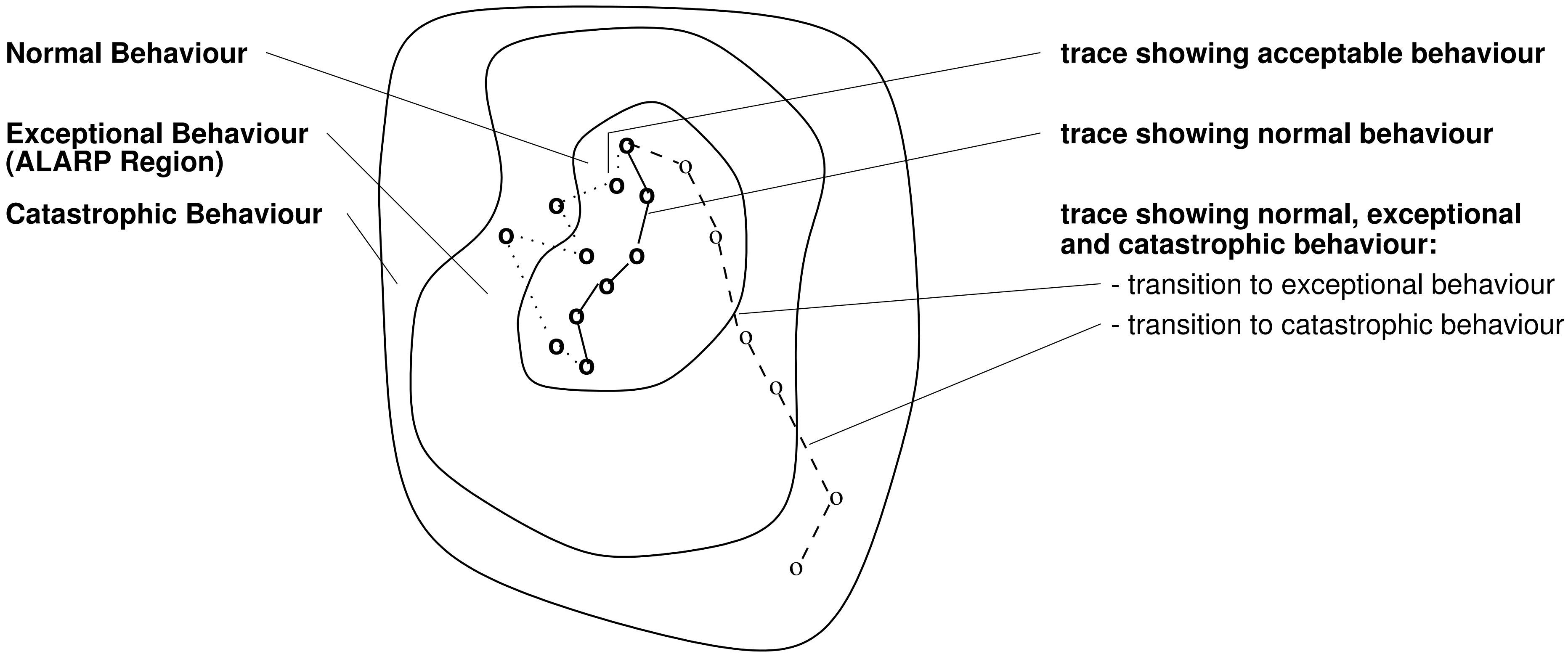
In computer science, specifications are classified as containing

- **Safety Properties S:** Any sequence of events or transitions etc. violating S contains a prefix all of whose infinite extensions violate S. (S always holds)
- **Liveness Properties L:** Any arbitrary finite sequence of events can be extended to an infinite sequence satisfying L. (L finally holds)

For safety-critical real-time systems, all dependability requirements should be specified as safety properties: Liveness properties can only guarantee that a service will FINALLY be delivered, which is not sufficient in the context of hard real-time systems.

Dependability — Classification of System Behaviour

System state space is partitioned into three areas. System execution is regarded as sequence of state transitions, possibly accompanied by visible actions (input/output).



Overview

1. The Notion of Dependability
2. Safety-Related Standards and V-Models
3. Modelling Safety-Critical Systems
4. Hazard Analysis and Risk Assessment
5. Design Criteria for Safety-Critical Systems
6. Validation, Verification and Test of Safety-Critical Systems

Safety-Related Terminology

- **Accident:** An undesired and unplanned event that results in a specified level of loss.
- **Severity of an Accident:** Specification of the level of loss caused by the accident, e. g., neglectable – minor – critical – catastrophic
- **Hazard:** Something that has the potential to do harm or can lead to an accident. Hazards “inherit” their severity attributes from the most harmful accidents they may cause.
- **(System) Safety Requirements:** A specification stating the acceptable relations

hazard severity ↔ probability of hazard occurrence

Safety-Related Standards – Common Understanding

Safety requirements should be derived from a **Risk Analysis**, consisting of **Hazard Analysis** and **Risk Assessment**.

- **Hazard Analysis:** list of possible hazards, their impact on the environment, their possible causes (e. g., sequences of faults leading to a hazard). Typically, it consists of the following items:
  - **Hazard List:** collection of the identified hazards
  - **Hazard-Severity Matrix:** relates hazards to severity
  - **Hazard-Probability Matrix:** relates hazards to the probability of their occurrence
  - **Hazard Model:** description of the possible causes – that is, sequences of events – leading to a hazard

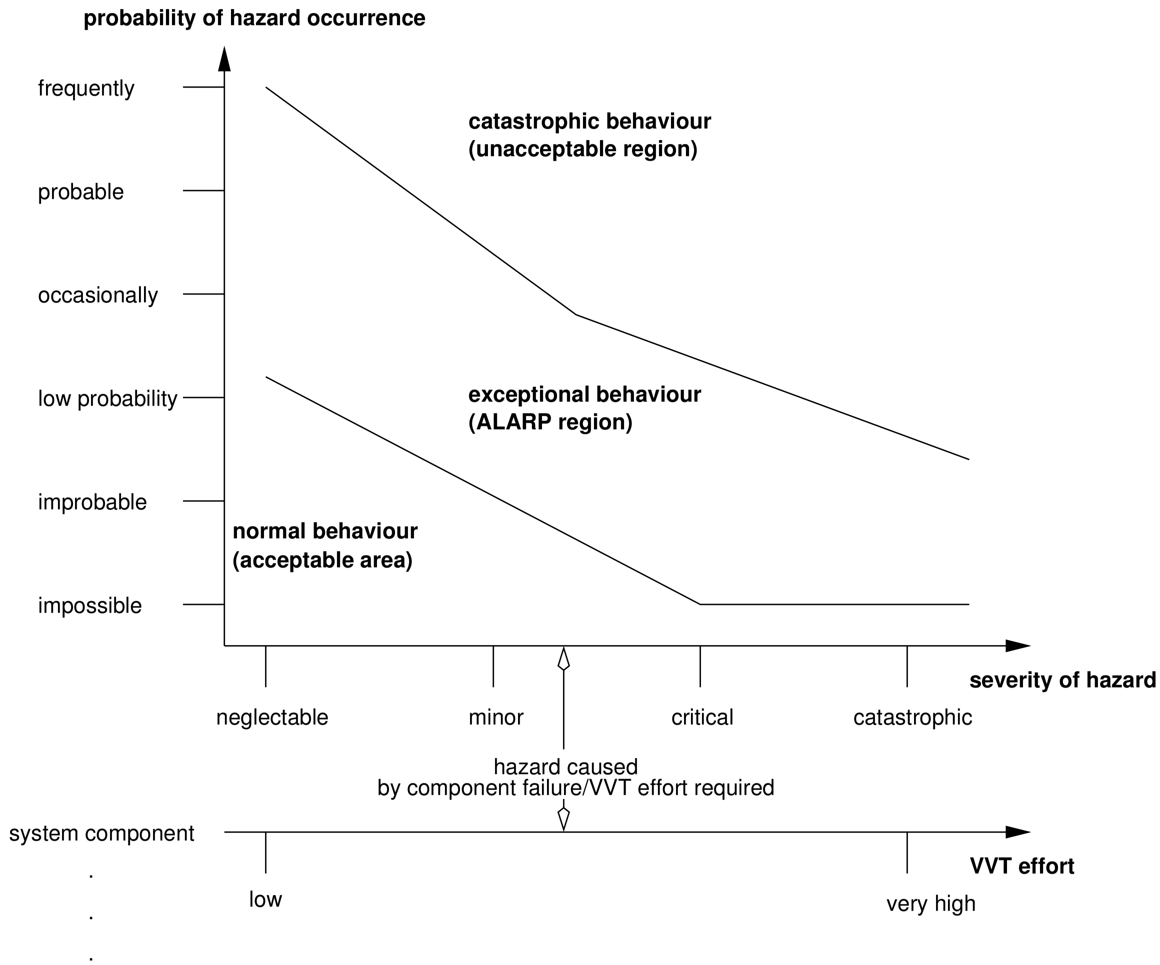
- **Risk Assessment:** quantitative or qualitative estimates for the probability that a hazard will occur

According to today's state of the art, a hazard model should at least be semi-formal, for example using fault-trees, event tree analysis of cause-consequence analysis.

Safety-Related Standards – Common Understanding

- relate severity of hazard, probability of occurrence and system behaviour in **Risk Diagram** or **Hazard Risk Index**
- derive required effort for validation, verification and test (VVT) of system components from the risk diagram and the impact of component failure on hazard occurrence
- VVT activities for components with highest criticality should be performed by **Independent Parties**

Safety-Related Standards – Risk Diagram



Safety-Related Standards – Common Understanding

If – in spite of all safety-related precautions – a hazardous situation results in an accident, then detailed investigations are performed with the objective to prevent re-occurrence of similar accidents:

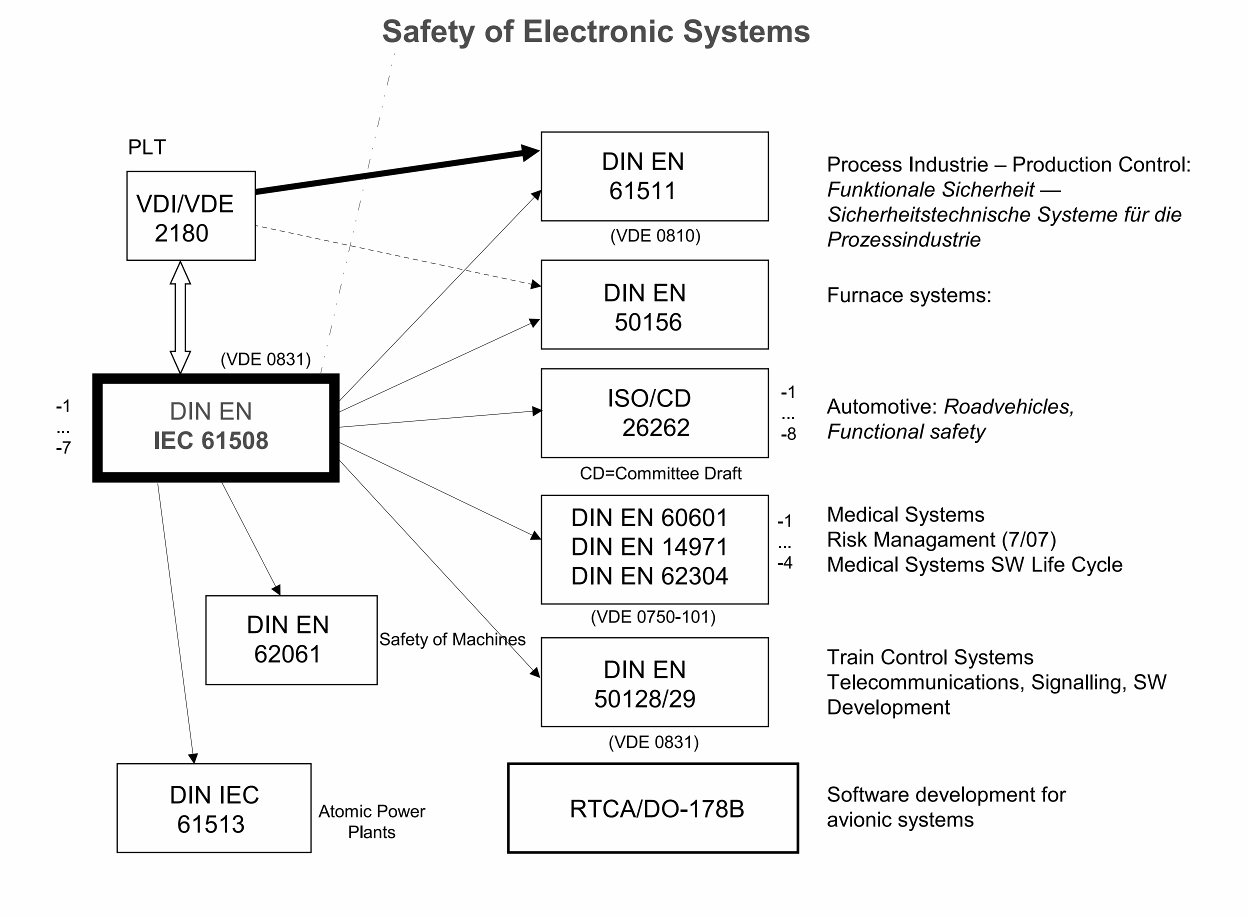
- **Root Cause Analysis** denotes the task to identify the crucial causes of an accident. The term "root cause" indicates that the crucial causes to look for are those at the **beginning** of the causal chains finally resulting in the accident.
- Of special interest is the subset of root causes whose occurrence can be **controlled** (that is, prevented) by technical or organisational measures.

Safety-Related Standards – Common Understanding

- Root cause analysis proceeds according to the following steps:

  1. Data collection
  2. Causal factor ("causal chain") charting
  3. Root cause identification
  4. Recommendation elaboration
  5. Recommendation implementation

Safety-Related Standards



Safety-Related Standards

Important standards for the development of safety-critical systems:

- IEC 61508: Sicherheit elektronischer Systeme
- IEC 61511: Funktionale Sicherheit – Sicherheitstechnische Systeme für die Prozessindustrie
- IEC 61513: Kernkraftwerke – Leittechnik für Systeme mit sicherheitstechnischer Bedeutung – Allgemeine Systemanforderungen
- EN 50129: Bahnanwendungen – Telekommunikationstechnik, Signaltechnik und Datenverarbeitungssysteme – Sicherheitsrelevante elektronische Systeme für Signaltechnik

Safety-Related Standards

Important standards for the development of safety-critical systems:

- EN 62061: Sicherheit von Maschinen – Funktionale Sicherheit sicherheitsbezogener elektrischer, elektronischer und programmierbarer elektronischer Steuerungssysteme
- ISO CD 26262: Road vehicles – Functional safety

Safety-Related Standards for Civil Aircraft Systems

The overall test process is driven by the following high-level standards

- **Air Transport Association (ATA) Chapters.** A systematic decomposition of a conceptual aircraft into aircraft systems. System descriptions induce the fundamental functions required in an aircraft

  Examples.

  - ATA-Chapter 21. Air Conditioning
  - ATA-Chapter 30. Ice and Rain Protection
  - ATA-Chapter 32. Landing Gear

  Note. The ATA description is “slightly outdated” from today’s point of view since it already suggests a function ↔ system association

Safety-Related Standards for Civil Aircraft Systems

Technical standards such as **Radio Technical Commission for Aeronautics (RTCA)** and **Aeronautical Radio Incorporated (ARINC)** standards specify the (minimal) requirements for equipment implementing aircraft functions

Examples.

- ARINC 653: Avionics Application Software Standard Interface
- ARINC 664: Aircraft Data Network (Avionics Full Duplex Switched Ethernet (AFDX))
- RTCA DO-200A: Standards for Processing Aeronautical Data

Safety-Related Standards for Civil Aircraft Systems

Technical standards such as **Radio Technical Commission for Aeronautics (RTCA)** and **Aeronautical Radio Incorporated (ARINC)** standards specify the (minimal) requirements for equipment implementing aircraft functions

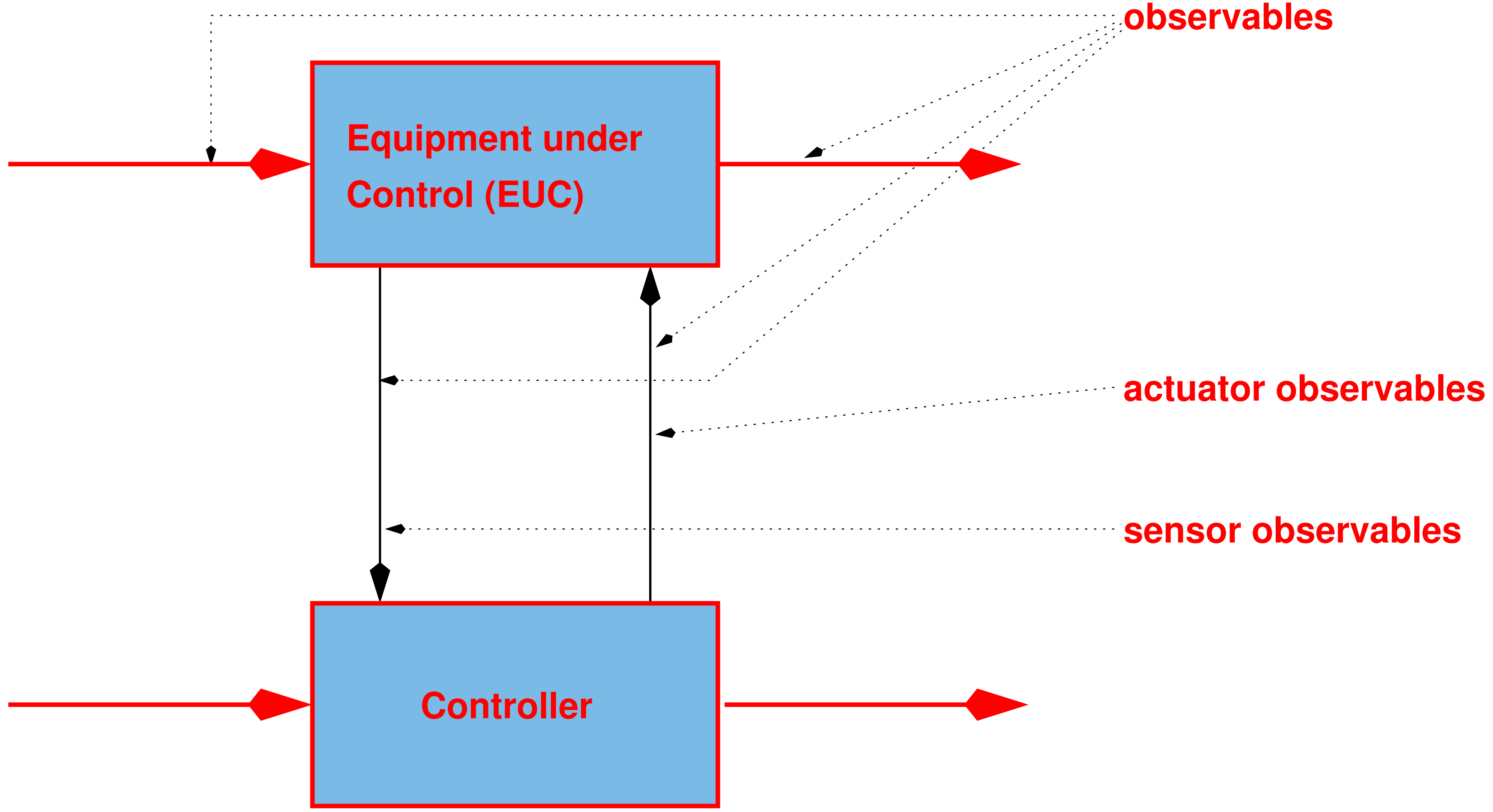
Examples.

- RTCA DO-185B: Minimum Operational Performance Standards for Traffic Alert and Collision Avoidance System II (TCAS II)
- RTCA DO-178B: Software Considerations in Airborne Systems and Equipment Certification

Overview

1. The Notion of Dependability
2. Safety-Related Standards and V-Models
3. Modelling Safety-Critical Systems
4. Hazard Analysis and Risk Assessment
5. Design Criteria for Safety-Critical Systems
6. Validation, Verification and Test of Safety-Critical Systems

Modelling Safety-Critical Systems



Modelling Safety-Critical Systems

- **Physical Model** specifies how the **Equipment Under Control (EUC)** behaves. This model should be independent of the presence/absence of a controller.
- **(System) Hazard Model** describes the possible causes that may lead to the identified system hazards.
- **Controller Model:** specifies requirements for a control system such that
  - EUC system hazards will not occur **(controller safety requirements)**
  - additional non safety-related EUC behaviour will be ensured **(user requirements)**

Observations:

- System safety requirements are often specified by separate authorities, not by the customer.
- Controller safety requirements have to be specified by the team responsible for the controller.
- User requirements specified by the customer may be in conflict with safety requirements.
- It has to be verified that the controller safety requirements will fulfil the system safety requirements.

The specification of controller safety requirements should always be separated from user requirements.

Modelling Safety-Critical Controllers – Specification Methods

Specification Methods suitable for physical model and controller model can be classified according to

Method	DM	FM	UBM	TBM	VVT	CG
SA/RT/IM	+	+	·	-	·	·
STATECHARTS	·	+	+	·	·	·
SDL	·	+	+	·	·	+
CSP	·	·	+	+	+	+
CCS	·	·	+	+	+	+
LOTOS	·	·	+	-	+	+
Z	+	+	·	-	·	·
VDM	+	+	·	-	·	+
UML2.0	+	+	+	·	+	+
HybridUML	+	+	+	+	+	+

DM = Data Model, FM = Functional Model , UBM = Untimed Behavioural Model, TBM = Timed Behavioural Model, VVT = support for Validation Verification and Test, CG = support for code generation from specifications , + = good support , · = weak support , - = no support

Modelling Safety-Critical Controllers – Specification Methods

Example 1: Physical Model and Controller Model specification with CSP – door locking mechanism for laboratory: When laser system is active, door shall be locked and laboratory shall be empty.

Physical model (EUC):

```
EUC = LASER ||| (DOOR [| open, close |] PERSON)
LASER = switchOn -> laserActive ->
        switchOff -> laserPassive -> LASER
DOOR = open -> close -> DOOR
PERSON = open -> enter -> close ->
         stayInLaboratory -> open -> leave ->
         close -> PERSON
```

Modelling Safety-Critical Controllers – Specification Methods

Example 1 (continued): Hazardous traces are the ones containing event sequences

```
<...,open,switchOn, ... >
<...,open,enter,switchOn, ... >
<...,open,enter,close,switchOn, ... >
<...,open,enter,close,stayInLaboratory,switchOn, ... >
<...,open,enter,close,stayInLaboratory,open,switchOn, ... >
<...,open,enter,close,stayInLaboratory,open,leave,switchOn,
<...,switchOn,open,...>
<...,switchOn,laserActive,open,...>
<...,switchOn,laserActive,switchOff,open,...>
```

Example 1 (continued): Hazardous traces are formally specified using **trace specifications**

$$
\begin{aligned}
&\mathrm{HAZARD}(tr) \equiv \exists u_1, u_2, u_3 : \\
&\quad tr = u_1 \frown \langle open \rangle \frown u_2 \frown \langle open \rangle \frown u_3 \frown \langle close \rangle \wedge \\
&\quad \#(u_1 \upharpoonright \{open\}) \mod 2 = 0 \wedge u_2 \upharpoonright \{open\} = \langle\ \rangle \wedge \\
&\quad ((u_1 \upharpoonright \{switchOn, laserActive, switchOff, laserPassive\} \neq \langle\ \rangle \wedge \\
&\qquad last(u_1 \upharpoonright \{switchOn, laserActive, switchOff, laserPassive\}) \neq laserPassive) \\
&\quad \vee (u_2 \upharpoonright \{switchOn, laserActive, switchOff\} \neq \langle\ \rangle) \\
&\quad \vee (u_3 \upharpoonright \{switchOn, laserActive, switchOff\} \neq \langle\ \rangle))
\end{aligned}
$$

Example 1 (continued): The development objective for the controller is to ensure for system

```
SYSTEM = EUC [| I |] CONTROLLER
```

with some suitable interface I the safety specification

$$SYSTEM\ \mathbf{sat}\ \neg \mathrm{HAZARD}(tr)$$

Example 1 (continued): A suitable interface is

$$I = \{open, close, switchOn, laserPassive\}$$

and a suitable controller can be specified as

```
CONTROLLER = open -> C1 [] switchOn -> C2
C1 = close -> open -> close -> CONTROLLER
C2 = laserPassive -> CONTROLLER
```

Example 1 (continued): The safety verification is typically performed by using a **watchdog process** which monitors all events from the system alphabet A and induces a deadlock as soon as a safety violation occurs:

```
A = {switchOn,laserActive,switchOff,laserPassive,
     open,close,enter,stayInLaboratory,leave}
WATCHDOG = W(<>)
W(tr) = if ( \text{HAZARD}(tr) ) then STOP
        else ([] e:A @ e -> W(tr^<e>))
```

If `VERIFY = SYSTEM [| A |] WATCHDOG` is free of deadlocks, this proves the safety of `SYSTEM`

Example 1 – state-based description with specification in temporal logic:

State-Transition System $STS = (S, s_0, V, T)$:

- V : Set of variable symbols
- S : Set of states, each state $s \in S$ a valuation $s : V \to D$ of variables (D the associated variable domain)
- $T \subseteq S \times S$: The transition relation

Run (execution) of STS: State sequence $\langle s_0, s_1, s_2, \ldots \rangle$ with

$$\forall i \geq 0 : (s_i, s_{i+1}) \in T$$

Example 1 – modelled as STS:

- Variable symbols $V = \{door, laser, dcnt\}$
- Auxiliary variable *dcnt* ("door-open/closed counter") counts how often the door has been opened or closed
- Domains $D = D(door) \cup D(laser) \cup D(dcnt)$,
  $D(door) = \{open, closed\}$,
  $D(laser) = \{passive, on, active, off\}$, $D(dcnt) = \mathbb{N}_0$
- Valuation can be written like
  $s = \{door \mapsto open, laser \mapsto on, dcnt \mapsto 5\}$

Example 1 – modelled as STS – transition relation:

$T \subseteq S \times S$: without control, T allows any (possibly unsafe) transitions (s, s') satisfying

1. $s_0(door) = closed \wedge s_0(dcnt) = 0 \wedge s_0(laser) = passive$
2. $s(laser) = passive \Rightarrow s'(laser) \in \{passive, on\}$
3. $s(laser) = on \Rightarrow s'(laser) \in \{on, active\}$
4. $s(laser) = active \Rightarrow s'(laser) \in \{active, off\}$
5. $s(laser) = off \Rightarrow s'(laser) \in \{off, passive\}$

Example 1 – modelled as STS – transition relation:

$$
\begin{aligned}
&(s(door) = d \wedge s(dcnt) = n) \Rightarrow \\
&\quad ((s'(door) = d \wedge s'(dcnt) = n) \vee (s'(door) \neq d \wedge s'(dcnt) = n + 1))
\end{aligned}
$$

Modelling Safety-Critical Controllers – Specification Methods

Example 1 – specification of safe runs using temporal logic:

Recall operators of **Linear Temporal Logic (LTL)**, defined on runs $r = \langle s_0, s_1, s_2, \ldots \rangle$ of STS:

- **State formulas p over V:** logic formulas with free variables in V, involving $\exists, \forall, \neg, \wedge, \vee, \Rightarrow, \iff$

  Interpretation: p holds in state $s \in S$ if its valuation $s(p)$ is true.

  Example: $door = open \Rightarrow laser \neq active$ is interpreted in state s as $s(door) = open \Rightarrow s(laser) \neq active$

Example 1 – specification of safe runs using temporal logic – Temporal operators:

- **Globally** p**:** $\Box\, p$ (or $G\ p$) holds for run r iff $\forall i \geq 0 : s_i(p)$
- **Next** p**:** $\bigcirc\, p$ (or $X\ p$) holds in state s_i of run r iff $s_{i+1}(p)$ is true
- **Eventually (finally)** p**:** $\Diamond\, p$ (or $F\ p$) holds for run r iff $\exists i \geq 0 : s_i(p)$
- p **Until** q**:** pUq holds for run r iff

$$(\exists i \geq 0 : s_i(q)) \wedge (\forall j < i : s_j(p))$$

Example 1 – physical model specification using temporal logic:

The restrictions about the physical model can be specified as follows:

1. $s_0(door) = closed \wedge s_0(dcnt) = 0 \wedge s_0(laser) = passive$
2. $\Box(laser = passive \Rightarrow \bigcirc(laser \in \{passive, on\}))$
3. $\Box(laser = on \Rightarrow \bigcirc(laser \in \{on, active\}))$
4. $\Box(laser = active \Rightarrow \bigcirc(laser \in \{active, off\}))$
5. $\Box(laser = off \Rightarrow \bigcirc(laser \in \{off, passive\}))$

Modelling Safety-Critical Controllers – Specification Methods

Example 1 – physical model specification using temporal logic:

$$\begin{aligned}&\forall d \in \{open, closed\}, n \in \mathbb{N}_0 : \\ &\quad \Box(door = d \wedge dcnt = n \Rightarrow \\ &\qquad \bigcirc((door = d \wedge dcnt = n) \vee (door \neq d \wedge dcnt = n + 1)))\end{aligned}$$

Observe: The laboratory is empty if and only if $dcnt \mod 4 = 0$

Example 1 – hazard specification using temporal logic:

Hazards can be characterised by formula

$$\mathrm{HAZARD} \equiv \Diamond((door = open \vee dcnt \mod 4 \neq 0) \wedge laser \neq passive)$$

In natural language, the hazard formula expresses

- Whenever the laser is not in passive state, it is hazardous if the door is open.
- Whenever the laser is not in passive state, it is hazardous if somebody is inside the laboratory (though the door may be closed).

Example 1 – safety specification using temporal logic:

The associated safety condition is the negation of the hazard characterisation:

$$\neg \mathrm{HAZARD} \equiv \Box((door = open \vee dcnt \mod 4 \neq 0) \Rightarrow laser = passive)$$

Example 1 – state-based description – Formal model for implementations:

Time-Discrete Input-Output Hybrid Systems

TDIOHS $\mathcal{H} = (Loc, Init, V, I, O, Trans)$**:**

- Loc : Locations
- V : variable symbols, $I, O \subset V$, $I \cap O = \emptyset$
- $Guard$: quantifier-free predicates over V
- $Init : Loc \longrightarrow Guard$: initialisation constraints
- $Assign$ the set of all pairs $(\vec{x} := \vec{t})$ with

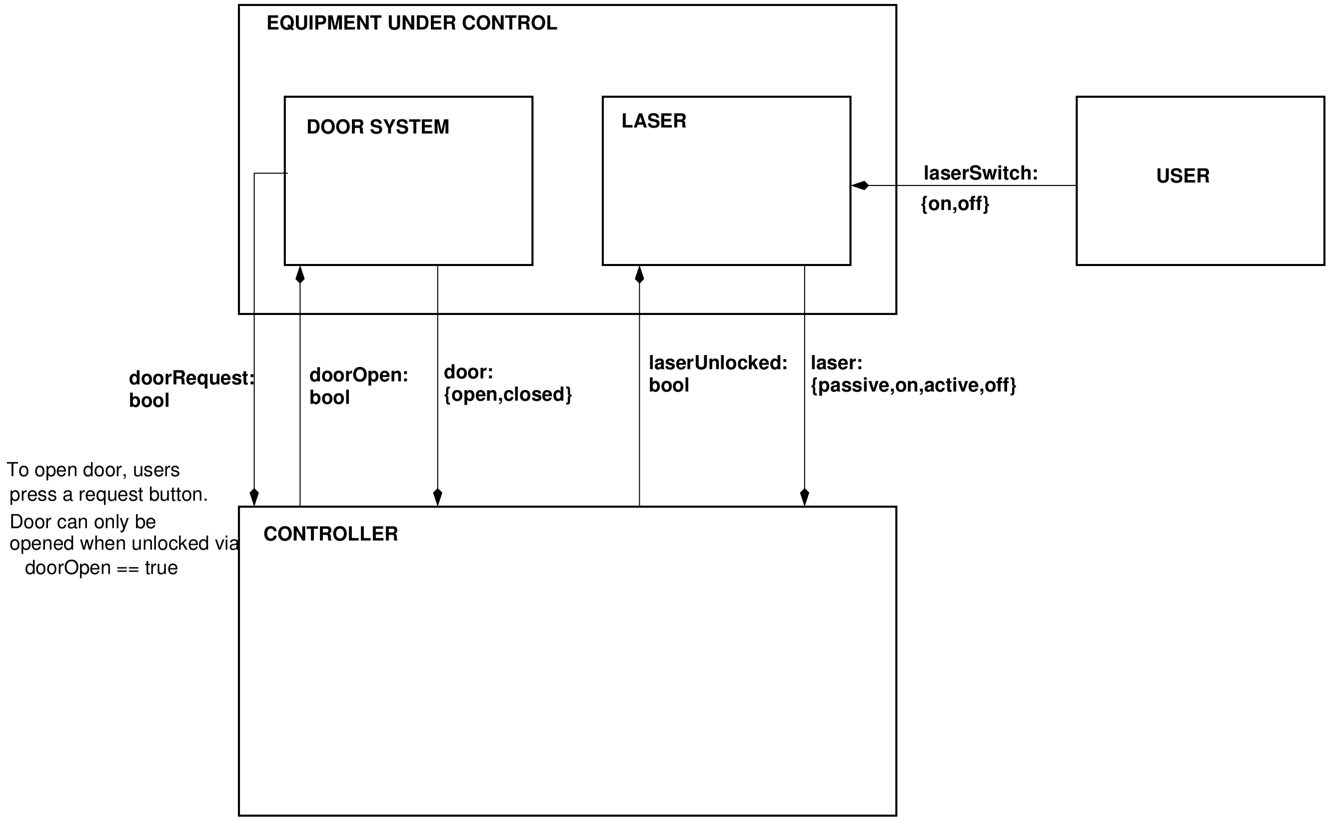
  - $\vec{x} = (x_1, x_2, \ldots)$ vector of all variables in $V - I$
  - $\vec{t} = (t_1, t_2, \ldots) \in T^{|V-I|}$
  - T terms over V

- $Trans \subseteq Loc \times Guard \times Assign \times Loc$: Transitions

Example 1 – I/O-safe TDIOHS: Programs adhere to the following **programming model:**

- Processing is performed in a sequential task operating in a main loop with the following processing phases:
  - **Input phase:** Inputs are read and copied to (global, static, heap or stack) variables – these variables are called **processing variables**
  - **Processing phase:** The control decisions are computed, operating on processing variables only
  - **Output phase:** The processing variables containing pre-computed output values are copied to their corresponding outputs

Example 1 – EUC and Controller:



Example 1 – Controller Code:

```
// Processing variables (initialised by false/passive/closed)
dop:bool; dx:{open,closed}; lu:bool; l:{passive,on,active,off};
dcnt:int; doorOld:bool; dr:bool;
while ( true ) {
 // Input phase ----------------------------------------------
 dx = door; l = laser; dr = doorRequest;
 // Processing phase -----------------------------------------
 if ( dx != doorOld ) { doorOld = dx; dcnt++ }
 dop = dx or (dcnt%4 != 0) or (dr and l == passive);
 lu = not dop;
 // Output phase ---------------------------------------------
 doorOpen = dop; laserUnlocked = lu;
 // Safety monitor -------------------------------------------
 if ( (dcnt%4 != 0 or doorOpen or door == open) and laser != passive )
    EMERGENCY_SHUTDOWN(); // Switch off power supply to laser
}
```

Example 1 – Controller Code Safety Proof:

First prove auxiliary property that *dcnt* is updated as required for the physical model, i. e., according to formula

$$\forall d \in \{open, closed\}, n \in \mathbb{N}_0 : \\ \Box(door = d \wedge dcnt = n \Rightarrow \\ \bigcirc((door = d \wedge dcnt = n) \vee (door \neq d \wedge dcnt = n + 1))) \quad (*)$$

This proof follows from the program property which holds at line 8

$$\forall d \in \{open, closed\}, n \in \mathbb{N}_0 : \\ \Box(door = d \wedge dcnt = n \wedge \bigcirc(door \neq d \Leftrightarrow doorOld \neq dx))$$

so the increment `dcnt++` in line 8 establishes (*).

Example 1 – Controller Code Safety Proof:

Verification objective: Show that ¬HAZARD holds at the end of each output phase.

Proof relies on **physical property of electro-mechanical door/laser safety system**:

$$\Box\,(\mathsf{laser} \neq \mathit{passive} \Rightarrow \mathsf{laserUnlocked}) \tag{1}$$

We will prove that **program ensures**

$$\Box(\mathsf{doorOpen} \Rightarrow \neg\mathsf{laserUnlocked}) \tag{2}$$
$$\Box(\mathsf{dcnt}\%4 \neq 0 \Rightarrow \neg\mathsf{laserUnlocked}) \tag{3}$$
$$\Box(\mathsf{door} = \mathit{open} \Rightarrow \mathsf{doorOpen}) \tag{4}$$

Properties (1),(2),(3),(4) obviously imply ¬HAZARD

Example 1 – Controller Code Safety Proof:

For I/O-safe TDIOHS the proof of a property $\Box\phi$ is equivalent to showing that ϕ is an invariant of the program's main while-loop. We therefore have to prove that the following properties are invariants:

$$\mathsf{doorOpen} \Rightarrow \neg\mathsf{laserUnlocked} \qquad (2')$$

$$\mathsf{dcnt}\%4 \neq 0 \Rightarrow \neg\mathsf{laserUnlocked} \qquad (3')$$

$$\mathsf{door} = open \Rightarrow \mathsf{doorOpen} \qquad (4')$$

Example 1 – Controller Code Safety Proof:

Proof is based on operational semantics of while languages.

Properties (2’), (3’), (4’) hold when initially entering the while loop at line 4, due to variable initialisations.

Now suppose that (2’), (3’), (4’) hold in line 5 (s_ℓ denotes the variable state before execution of line ℓ):

$$s_5 \models \mathsf{doorOpen} \Rightarrow \neg\mathsf{laserUnlocked}$$
$$s_5 \models \mathsf{dcnt}\%4 \neq 0 \Rightarrow \neg\mathsf{laserUnlocked}$$
$$s_5 \models \mathsf{door} = open \Rightarrow \mathsf{doorOpen}$$

It has to be shown that then these properties also hold at line 13 in program state s_{13}.

Example 1 – Controller Code Safety Proof:

Property (2') follows from the assignment and sequence rules of the operational semantics, applied to lines 10 — 12:

$$s_{12} = s_{10} \oplus \{\mathsf{lu} \mapsto \neg s_{10}(\mathsf{dop})\}$$
$$s_{13} = s_{12} \oplus \{\mathsf{doorOpen} \mapsto s_{12}(\mathsf{dop}), \mathsf{laserUnlocked} \mapsto s_{12}(\mathsf{lu})\}$$

which implies $s_{13}(\mathsf{laserUnlocked}) = \neg s_{10}(\mathsf{dop})$ and, since $s_{12}(\mathsf{dop}) = s_{10}(\mathsf{dop})$,

$$s_{13} \models \mathsf{doorOpen} \Rightarrow \neg \mathsf{laserUnlocked}$$

Example 1 – Controller Code Safety Proof:

Property (3') follows from the assignment and sequence rules of the operational semantics, applied to lines 9 — 12:

$$\begin{aligned} s_{10} &= s_9 \oplus \{\mathsf{dop} \mapsto (s_9(\mathsf{dx}) = open) \vee s_9(\mathsf{dcnt})\%4 \neq 0 \vee \\ &\qquad (s_9(\mathsf{dr}) \wedge s_9(\mathsf{l}) = passive)\} \\ s_{12} &= s_{10} \oplus \{\mathsf{lu} \mapsto \neg s_{10}(\mathsf{dop})\} \\ s_{13} &= s_{12} \oplus \{\mathsf{doorOpen} \mapsto s_{12}(\mathsf{dop}), \mathsf{laserUnlocked} \mapsto s_{12}(\mathsf{lu})\} \end{aligned}$$

which implies

$$s_{13} \models \mathsf{dcnt}\%4 \neq 0 \Rightarrow \mathsf{doorOpen} \quad \text{and therefore}$$
$$s_{13} \models \mathsf{dcnt}\%4 \neq 0 \Rightarrow \neg\mathsf{laserUnlocked}$$

Example 1 – Controller Code Safety Proof:

Property (4') follows from the assignment and sequence rules of the operational semantics, applied to lines 6 — 12:

$$\begin{aligned} s_{10} &= s_9 \oplus \{\mathsf{dop} \mapsto (s_6(\mathsf{door}) = \mathit{open}) \vee s_9(\mathsf{dcnt}) \% 4 \neq 0 \vee \\ &\qquad (s_9(\mathsf{dr}) \wedge s_9(\mathsf{l}) = \mathit{passive})\} \\ s_{12} &= s_{10} \oplus \{\mathsf{lu} \mapsto \neg s_{10}(\mathsf{dop})\} \\ s_{13} &= s_{12} \oplus \{\mathsf{doorOpen} \mapsto s_{12}(\mathsf{dop}), \mathsf{laserUnlocked} \mapsto s_{10}(\mathsf{lu})\} \end{aligned}$$

which implies

$$s_{13} \models \mathsf{door} = \mathit{open} \Rightarrow \mathsf{dop} \quad \text{and therefore}$$
$$s_{13} \models \mathsf{door} = \mathit{open} \Rightarrow \mathsf{doorOpen}$$

Overview

1. The Notion of Dependability
2. Safety-Related Standards and V-Models
3. Modelling Safety-Critical Systems
4. Hazard Analysis and Risk Assessment
5. Design Criteria for Safety-Critical Systems
6. Validation, Verification and Test of Safety-Critical Systems

Risk Analysis

Risk Analysis:
consists of two steps: **Hazard Analysis** and **Risk Assessment**

Hazard Analysis: Which conditions cause a hazard?

Risk Assessment: What are the probabilities that a hazard occurs, provided the hazard analysis is consistent and complete?

Modelling Techniques for Hazard Analysis

Overview of techniques:

- **FMEA:** Failure modes and effects analysis
- **FMECA:** Failure modes, effects and criticality analysis
- **HAZOP:** Hazard and operability studies
- **ETA:** Event tree analysis
- **FTA:** Fault tree analysis
- **FSM:** Finite state machines with reachability analysis

Hazard Analysis – FMECA

Objective: Investigate the possible ways (=**modes**) a component may fail and check whether this can lead to a hazard.

Advantages: Systematic analysis of each possible component failure and its hypothetic relationship to a hazard

Disadvantages:

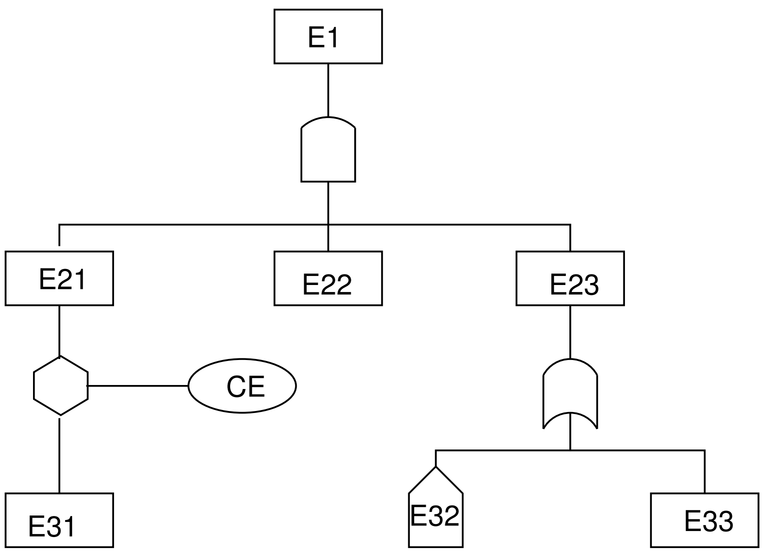
- Investigation of many components which have no effect on hazard occurrence
- No analysis of **simultaneous** faults in several components

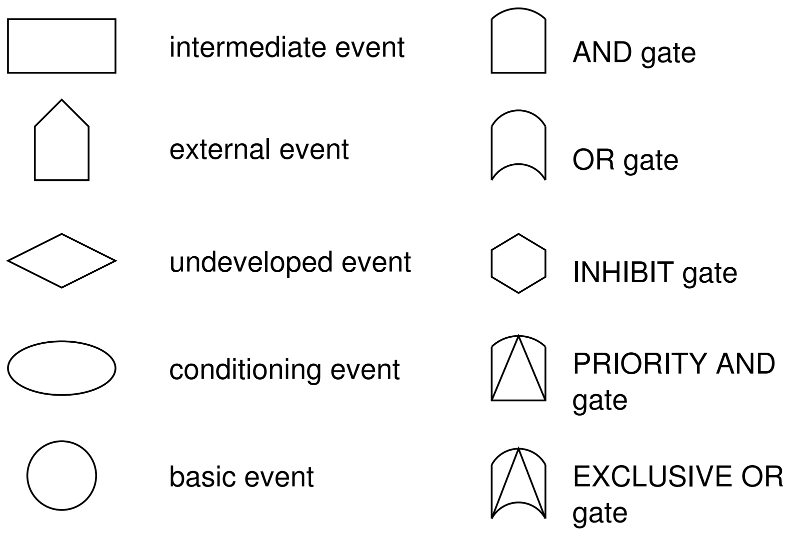
Recommendation: Use to complement FTA

Hazard Analysis – FMECA

Item	Failure Mode	Cause of failure	Possible Effects	Prob.	Criticality	Possible Actions to Reduce Failure Rate or Effects
Aircraft Smoke Detector	Smoke threshold too high	(1) Defect humidity sensor (2) Arithmetic error in threshold calculation software	Smoke in compartment remains undetected	$10^{-6}/1000$ flight hours	B	(1) Use redundant smoke detectors (2) Perform detector tests (3) Let detector issue *pre-threshold warning*: This indicates that the detector becomes "blind" so that higher smoke intensities are required to lead to a smoke alarm
Aircraft Smoke Detector	Smoke threshold too low	– see above –	Illegal smoke alarms	$10^{-6}/1000$ flight hours	C (Aircraft cannot start or continue flight)	– see above –

Hazard Analysis – Fault Trees





Deriving Safety Requirements from Hazard Analysis

If hazard analysis has been performed using fault trees, every set of conditions ensuring that the root of the fault tree will never be reached represent a sufficient set of safety requirements.

- root-hazard E0 gives rise to initial requirement **not(E0)** to be refined by the requirements derived from lower-level nodes in the tree
- OR-gates E1,...,En give rise to requirement **not(E1) AND ... AND not(En)**
- AND-gates E1,...,En give rise to requirement **not(E1) OR ... OR not(En)**
- refinement of requirements stops when the leaves of the fault tree have been reached

Deriving Safety Requirements from Hazard Analysis

Example for fault-tree shown above.

Step 1: term replacement according to fault-tree

$$E1$$
$$\Longleftrightarrow$$
$$E21 \textbf{ and } E22 \textbf{ and } E23$$
$$\Longleftrightarrow$$
$$(E31 \textbf{ and } \textbf{not}(CE)) \textbf{ and } E22 \textbf{ and } (E32 \textbf{ or } E33)$$

Step 2: transform into disjunctive normal form

$$\Longleftrightarrow$$
$$(E31 \textbf{ and } \textbf{not}(CE) \textbf{ and } E22 \textbf{ and } E32)$$
$$\textbf{or}$$
$$(E31 \textbf{ and } \textbf{not}(CE) \textbf{ and } E22 \textbf{ and } E33)$$

Example for fault-tree shown above.

Step 3: determine resulting safety requirements by negation of disjunctive normal form

Safety Requirement 1:
not($E31$ **and** **not**(CE) **and** $E22$ **and** $E32$)

Safety Requirement 2:
not($E31$ **and** **not**(CE) **and** $E22$ **and** $E33$)

Deriving Safety Requirements from Hazard Analysis

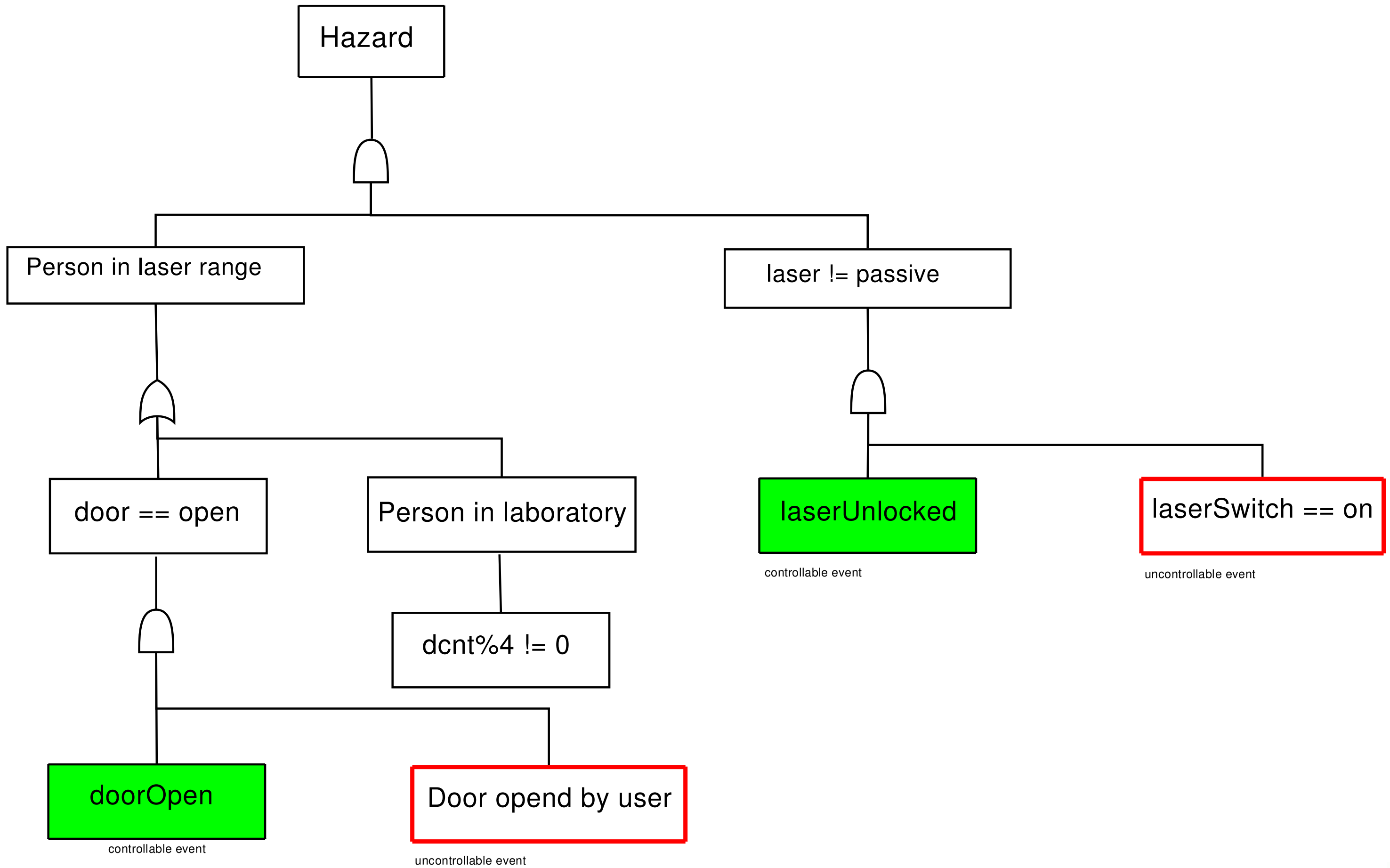
Definition of Cut Sets: Each (minimal) collection of elementary conditions jointly leading to a hazard is called a cut set.

Definition of Single-Point Failure: Cut set containing only one element

Cut sets derived from fault-tree shown above: given by disjunctive normal form as

$$\{E31,\ \mathbf{not}(CE), E22, E32\}$$
$$\{E31,\ \mathbf{not}(CE), E22, E33\}$$

Example 1 (continued): FTA for laboratory with laser



Deriving Safety Requirements from Hazard Analysis

Example 1 (continued): Cut sets for laboratory with laser derived from FTA above:

$$C_1 = \{\mathsf{doorOpen}, \mathsf{laserUnlocked}, \mathsf{Door\ opened\ by\ user},\ \mathsf{laserSwitch} == \mathsf{on}\}$$
$$C_2 = \{\mathsf{dcnt\%}4 \neq 0, \mathsf{laserUnlocked}, \mathsf{laserSwitch} == \mathsf{on}\}$$

All uncontrollable events are assumed to always happen in order to contribute to a hazard situation. Therefore the controller has to ensure that event combinations

$$C_1' = \{\mathsf{doorOpen}, \mathsf{laserUnlocked}\}$$
$$C_2' = \{\mathsf{dcnt\%}4 \neq 0, \mathsf{laserUnlocked}\}$$

can never occur:

$$\text{Safety Requirement 1} \equiv \Box(\neg(\mathsf{doorOpen} \land \mathsf{laserUnlocked}))$$
$$\text{Safety Requirement 2} \equiv \Box(\neg(\mathsf{dcnt\%}4 \neq 0 \land \mathsf{laserUnlocked}))$$

Deriving Safety Requirements from Hazard Analysis

Example 1 (continued): In order to prove that the control program for the laboratory introduced above really fulfills safety requirements 1 and 2 above we have to show that

$$\mathsf{INV} \equiv \neg(\mathsf{doorOpen} \wedge \mathsf{laserUnlocked}) \wedge \neg(\mathsf{dcnt}\%4 \neq 0 \wedge \mathsf{laserUnlocked})$$

is an invariant of the program's while loop. Observe that the fault tree analysis above relies on the additional fact that

$$\mathsf{door} = open \Rightarrow \mathsf{doorOpen}$$

Controller Hazard Analysis

Safety mechanisms have to be ensured even in presence of internal system faults.

A second internal hazard analysis is necessary to show that the system design contains the proper mechanisms to tolerate faults without violating the essential safety requirements.

Internal hazard analysis should take into account:

- faults/errors/failures of hardware components
- erroneous behaviour of SW components
- corrupted data

Risk Assessment - General Definitions

Unreliability:

Define the **Unreliability Function $Q(t)$**, that is, the probability for a component to fail in interval $[0, t]$, as

$$Q(t) = \int_0^t f(u)du$$

where $f(t)$ is a **Probability Density Function (PDF)**, the **failure density**. Mathematically speaking, $Q(t)$ is a **Cumulative Density Function (CDF)**. Obviously

$$Q(\infty) = \int_0^\infty f(u)du = 1$$

since $f(t)$ is a PDF.

Risk Assessment - General Definitions

Define the **Reliability Function** $R(t)$ as the probability for a component **not** to fail in interval $[0, t]$, that is,

$$R(t) = 1 - Q(t)$$

If $Q(t)$ is defined as above with density function $f(t)$, then

$$\begin{aligned} R(t) &= 1 - \int_0^t f(u)du \\ &= \int_0^\infty f(u)du - \int_0^t f(u)du \\ &= \int_t^\infty f(u)du \end{aligned}$$

Risk Assessment - General Definitions

$Q(t)$ may be determined approximately by testing a large number N of identical components and setting

$$Q(t) = \frac{n_f(t)}{N},$$

where $n_f(t) \leq N$ is the number of components which failed in interval $[0, t]$.

Conversely, $R(t)$ may be determined approximately by testing a large number N of identical components and setting

$$R(t) = \frac{n(t)}{N},$$

where $n(t) \leq N$ is the number of components still functioning correctly at after a time interval of length t has passed.

Failure Density:

Indicates tendency (“constant”, “strongly/weakly increasing/decreasing”) of unreliability at time point t:

$$f(t) = \frac{dQ(u)}{du}(t)$$

for differentiable unreliability function Q. This implies

$$\begin{aligned}\frac{dR(u)}{du}(t) &= \frac{1 - Q(u)}{du}(t) \\ &= -\frac{dQ(u)}{du}(t) \\ &= -f(t)\end{aligned}$$

Risk Assessment - General Definitions

Failure Rate:

Probability for system to fail in interval $[t, t + \Delta t]$, provided that no failure occurred before t:

$$\text{Failure Rate} = \frac{F(t + \Delta t) - F(t)}{\Delta t R(t)}$$

Hazard Rate:

Limit of failure rate for $\Delta t \rightarrow 0$:

$$z(t) = \lim_{\Delta t \rightarrow 0} \frac{F(t + \Delta t) - F(t)}{\Delta t R(t)} = \frac{f(t)}{R(t)}$$

Risk Assessment - General Definitions

Mean Time To Failure:

Expected value of the time the system will operate before the occurrence of the first failure.

$$MTTF = \int_0^\infty R(u)du$$

Mean Time To Repair: Average repair time.

Mean Time Between Failures:

$$MTBF = MTTR + MTTF$$

provided that the systems is "as good as new" after each repair (i.e. MTTF stays the same).

Risk Assessment - General Definitions

Availability:

Probability for the system to function correctly at a given time.

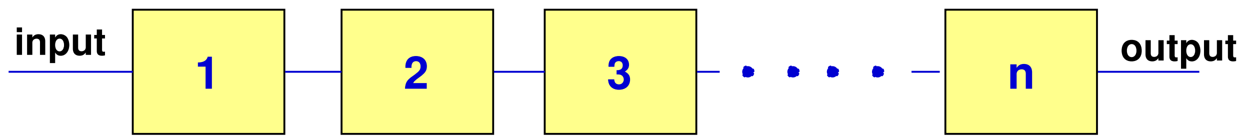
$$\text{Availability} = \frac{MTTF}{MTTF + MTTR} = \frac{MTTF}{MTBF}$$

Unavailability:

$$\text{Unavailability} = 1 - \text{Availability}$$

Reliability Modelling - Combinational Models

A series combination of components



If $R_i(t)$ is the reliability of component i, and the components are independent, the system reliability computes to

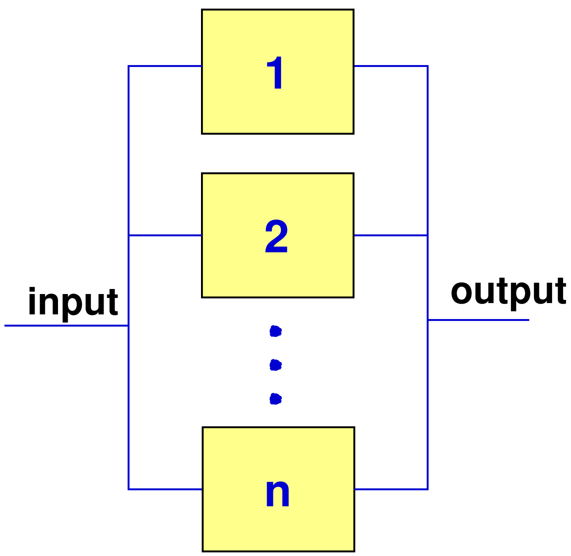
$$R(t) = \prod_{i=1}^{n} R_i(t)$$

Example: A series combination of 100 independent components with $R_i(t) = 0.999$ for each i, yields a low system reliability of

$$R(t) = 0.999^{100} = 0.905.$$

Reliability Modelling - Combinational Models

A parallel combination of components



If $R_i(t)$ is the reliability of component i, and the components are independent, the system reliability computes to

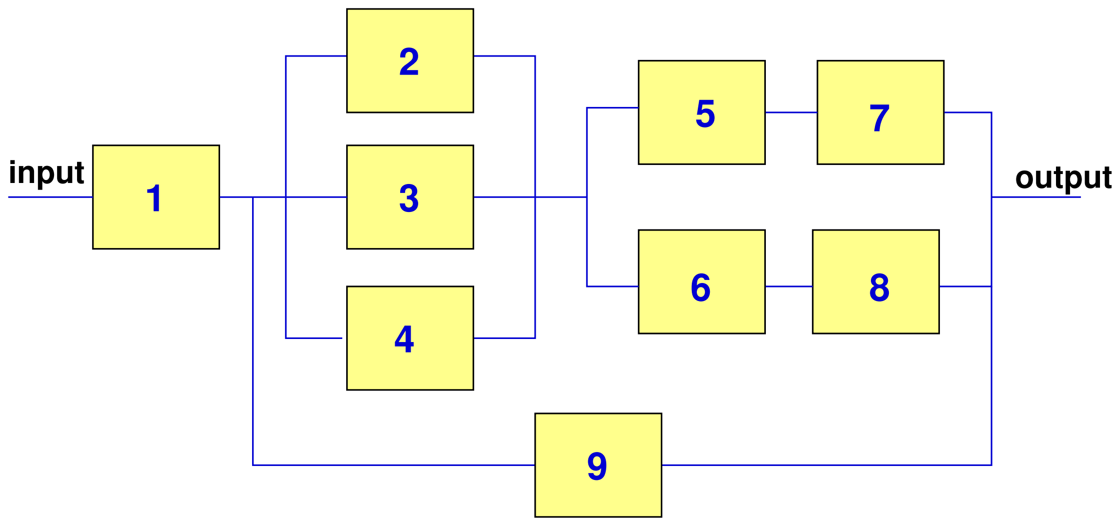
$$R(t) = 1 - \prod_{i=1}^{n}(1 - R_i(t))$$

Example: A parallel combination of 3 independent components with $R_i(t) = 0.999$ for each i, yields a system reliability of

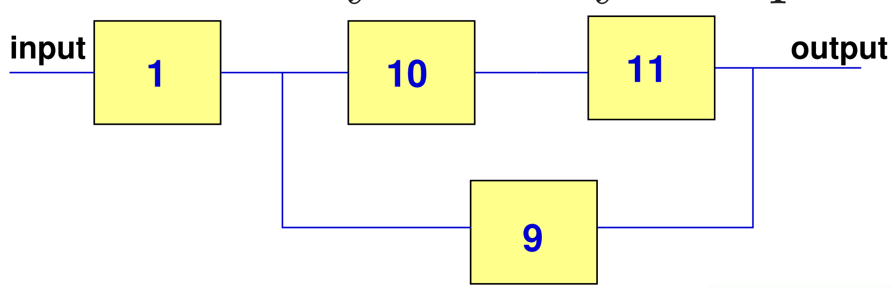
$$R(t) = 1 - (1 - 0.999)^3 = 0.999\,999\,999$$

Reliability Modelling - Combinational Models

Series-parallel combinations



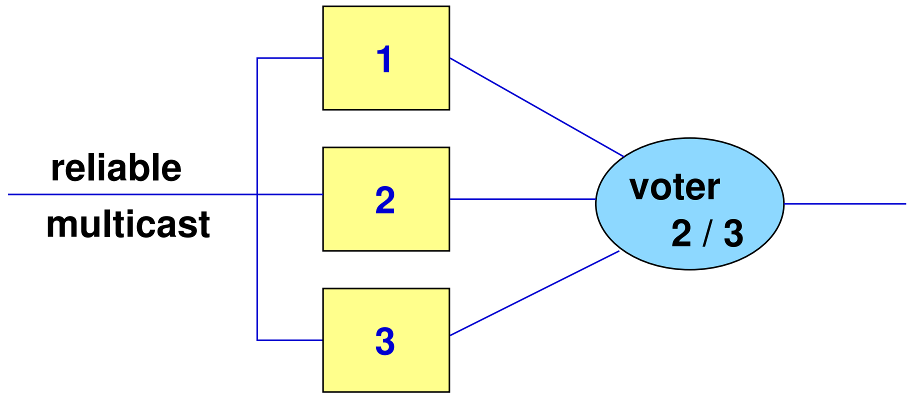
From the above formulas, the system reliability is easily computed in two steps, using the following abstraction:



N-modular Redundancy

Use N redundant components. The system will function correctly if at least $m < N$ modules are operational.

Example:
$N = 3, m = 2$
independent,
voter correct.



$$R(t) = R_1(t)R_2(t)R_3(t) + (1 - R_1(t))R_2(t)R_3(t) + R_1(t)(1 - R_2(t))R_3(t) + R_1(t)R_2(t)(1 - R_3(t))$$

For $R_1(t) = R_2(t) = R_3(t) = 0.95$ holds $R(t) = 0,993$,
but $R_1(t) = R_2(t) = R_3(t) = 0.40$ yields $R(t) = 0,352$.

Failure Distributions

Exponential probability density function

$$f(u) = \lambda e^{-\lambda u}$$

Since $\frac{d(-e^{-\lambda u})}{du}(t) = f(t)$, this distribution leads to a failure probability (the so-called **exponential failure law**)

$$\begin{aligned} Q(t) &= \int_0^t f(u)du \\ &= -e^{-\lambda t} - (-e^{-\lambda 0}) \\ &= 1 - e^{-\lambda t} \end{aligned}$$

and a reliabilty

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

Failure Distributions

Exponential probability density function and hazard rate

The exponential PDF leads to a hazard rate of

$$
\begin{aligned}
z(t) &= \frac{f(t)}{R(t)} \\
&= \frac{\lambda e^{-\lambda t}}{e^{-\lambda t}} \\
&= \lambda
\end{aligned}
$$

Interpretation: The exponential PDF is appropriate for system components during their most reliable phase of life (the **useful life stage**), where the hazard rate does not depend on time, but is constant.

Failure Distributions

Exponential probability density function and MTTF

The exponential PDF leads to an MTTF

$$
\begin{aligned}
MTTF &= \int_0^\infty R(u)du \\
&= \int_0^\infty e^{-\lambda u} du \\
&= \lim_{t \to \infty} (-\frac{1}{\lambda} e^{-\lambda t}) - (-\frac{1}{\lambda} e^{-\lambda 0}) \\
&= \frac{1}{\lambda}
\end{aligned}
$$

Failure Distributions

Experimental determination of exponential PDF

Since $R(t) = e^{-\lambda t}$ we can determine an approximate value of λ, if tests of the form

$$R(t_i) = \frac{n(t_i)}{N}, i = 1, \ldots, m$$

(N a large number of identical components, $n(t_i)$ the number of components still operable at time t_i) can be performed: Solve optimisation problem

$$\textbf{minimise} \;\; \Phi(\lambda) =_{\text{def}} \sum_{i=1}^{m} (R(t_i) - e^{-\lambda t_i})^2$$

Experimental determination of exponential PDF

A local minimum of $\Phi(\lambda)$ can be found by solving equation $d\Phi/d\lambda(u) = 0$, that is,

$$\sum_{i=1}^{m} t_i e^{-\lambda t_i}(R(t_i) - e^{-\lambda t_i}) = 0$$

The root $\lambda_0 \in \mathbb{R}$ with $d\Phi/d\lambda(\lambda_0) = 0$ can be approximated, for example, by using Newton's method.

Φ is a **maximum likelihood estimator** if the errors occurring in the measurement of t_i have normal distribution.

Failure Distributions

Experimental determination of exponential PDF

If, however, the t_i-measurements are rather uniformely distributed and/or contain freak values, other estimators are preferable, e. g.

$$\Psi(\lambda) =_{\text{def}} \sum_{i=1}^{m} |R(t_i) - e^{-\lambda t_i}|$$

Failure Distributions

The Weibull PDF is used for situations where time-dependent hazard rates have to be considered. The general form of the Weibull distribution has three free parameters:

$$f(u) = \frac{\beta}{\eta}\left(\frac{u-\gamma}{\eta}\right)^{\beta-1} e^{-\left(\frac{u-\gamma}{\eta}\right)^{\beta}}$$

which is defined for

$$u - \gamma \geq 0, \beta > 0, \eta > 0, \gamma \in \mathbb{R}$$

Failure Distributions

The failure probability $F(t)$ for the general Weibull PDF is

$$
\begin{aligned}
F(t) &= \int_0^t \frac{\beta}{\eta} \left(\frac{u-\gamma}{\eta} \right)^{\beta-1} e^{-\left(\frac{u-\gamma}{\eta} \right)^{\beta}} du \\
&= 1 - e^{-\left(\frac{t-\gamma}{\eta} \right)^{\beta}}
\end{aligned}
$$

Failure Distributions

The parameters are called

- **scale parameter** η
- **shape parameter (slope)** β
- **location parameter** γ: $\gamma \neq 0$ is used if the experiment does not start at $u = 0$, but at any time $\gamma \in \mathbb{R}$

Failure Distributions

- For experiments starting at time $u = 0$ we can use the **two-parameter Weibull PDF** by setting $\gamma = 0$:

$$f(u) = \frac{\beta}{\eta}\left(\frac{u}{\eta}\right)^{\beta-1} e^{-\left(\frac{u}{\eta}\right)^{\beta}}$$

- For experiments starting at time $u = 0$ with known parameter $\beta = C$ we get the **one-parameter Weibull** PDF

$$f(u) = \frac{C}{\eta}\left(\frac{u}{\eta}\right)^{C-1} e^{-\left(\frac{u}{\eta}\right)^{C}}$$

- For experiments starting at time $u = 0$ with known $\beta = 1$ we get the exponential PDF with $\lambda = \frac{1}{\eta}$

Failure Distributions

The MTTF for the three-parameter Weibull PDF is

$$\begin{aligned} MTTF &= \int_0^\infty \frac{\beta}{\eta}\left(\frac{u-\gamma}{\eta}\right)^{\beta-1} e^{-\left(\frac{u-\gamma}{\eta}\right)^{\beta}} du \\ &= \gamma + \eta \cdot \Gamma\left(\frac{1}{\beta}+1\right) \end{aligned}$$

with the **gamma function** $\Gamma(n)$ defined as

$$\Gamma(n) = \int_0^\infty e^{-u} u^{n-1} du$$

Observe that for $\gamma = 0, \eta = \frac{1}{\lambda}, \beta = 1$ this coincides with the MTTF of the exponential PDF, since $\Gamma(1) = 1$ and $\Gamma(x+1) = x \cdot \Gamma(x)$, so $\Gamma(2) = 1$.

Failure Distributions

The **Weibull hazard rate** is given by

$$z(t) = \frac{\beta}{\eta}\left(\frac{t-\gamma}{\eta}\right)^{\beta-1}$$

Overview

1. The Notion of Dependability
2. Safety-Related Standards and V-Models
3. Modelling Safety-Critical Systems
4. Hazard Analysis and Risk Assessment
5. Design Criteria for Safety-Critical Systems
6. Validation, Verification and Test of Safety-Critical Systems

Design Criteria for Safety-Critical Systems

- **Partitioning:** Faults and errors caused by non-critical processes should not be able to affect the behaviour of critical processes. This is achieved by
  - memory protection for reachable address ranges
  - robust interfaces to critical processes (abstract data types using partitioned shared memory, validity checks for data)
  - limited resources (CPU, IO-channels, memory) for non-critical processes

Design Criteria for Safety-Critical Systems

- **State-Based Behaviour:** Critical system behaviour should not depend on events (i. e. short signals or pulses which may be lost) but on state:
  - input messages are stored in state space
  - repetition of the same input should not lead to state changes (e. g., send absolute values instead of delta-values referring to previous state/message)
  - after system crash, last state should be recoverable from environment and/or stable storage

After system initialisation, there should be no more dynamic memory allocation at run-time.

Design Criteria for Safety-Critical Systems

- **Hard Real-Time Behaviour:** Real-time behaviour should be based on *discrete time semantics*:
  - late messages are equivalent to lost messages
  - fixed-cycle/fixed transmission length frame protocols for communication
  - applications should operate in fixed time frames
  - fixed house keeping phase reserved for I/O processing and safety control mechanisms

Design Criteria for Safety-Critical Systems

- **Use of Operating System:** Preferably, the operating system should only be used to service HW interrupts for I/O processing.
  - applications should operate according to the state machine programming paradigm, with round robin scheduling or cooperative multi tasking
  - use shared memory (protected by abstract data types and partitioning) for process communication instead of signals, message queues, pipes etc
  - use tightly coupled multi-processor systems with simple process-to-CPU allocation instead of complex scheduling policies

- on multi-processor systems use active wait and polling mechanisms instead of semaphores, signals and alarms

Design Criteria for Safety-Critical Systems

- **Fault-Tolerance:** For safety-critical systems, at least safety requirements must hold even in presence of faults caused by environment or by the system itself.
  - specify **stable safe states**
  - design robust mechanisms (HW/SW) which guarantee transition into stable safe state in case of errors that might otherwise lead to catastrophic behaviour
  - use **active replication** techniques to ensure safety requirements, if stable safe state cannot be found (for example in the control of aircraft engines)

- exploit data, code and HW redundancy to identify faulty components
- use fail-stop components or otherwise use Byzantine agreement protocols to reach consent between correctly operating components to isolate faulty ones
- use watchdog mechanisms which guarantee that **errors are revealed “as soon as possible”**

Observe that even fully verified SW may lead to erroneous system behaviour due to transient HW errors. As a consequence, SW diversity (code and data) is mandatory if HW redundancy is not available.

Design Criteria for Safety-Critical Systems

Instead of re-inventing every new safety-critical controller from scratch, use Collaborations, Design Patterns and Frameworks with proven generic correctness properties.

Collaboration: Description of a collection of cooperating objects by specification of their

- **static properties:**
  - architecture of the cooperating objects
  - roles of the objects
  - relationships between objects
- **dynamic properties regarding the message flow** (interactions) between objects:
  - sequencing
  - synchronisation
  - timing

Design Pattern:

- generic "small" collaboration (a "mechanism")
- usable in different application contexts
- generic parameters are Name, Data Type, Number of Objects, Methods etc.

Examples for Design Patterns:

- **Reader-Writer-Ringbuffer Pattern:** generic model for the asynchronous FIFO communication between reader and writer tasks without semaphore utilisation

- **Model-View-Controller Pattern:** generic model for the interaction between
  - application objects (“models”)
  - visualisation objects (“views”)
  - interaction control objects (“controller”)
- **Index-Stack/Data-Array Pattern:** dynamic data management without utilisation of operating system data allocation facilities
- **Multiplexer/Concentrator Pattern:** generic process communication model guaranteeing deadlock-free communication networks under boundary conditions which are trivial to check

Design Criteria for Safety-Critical Systems

Framework:

- generic “large” collaboration
- taylored for specific fields application

Examples for Frameworks:

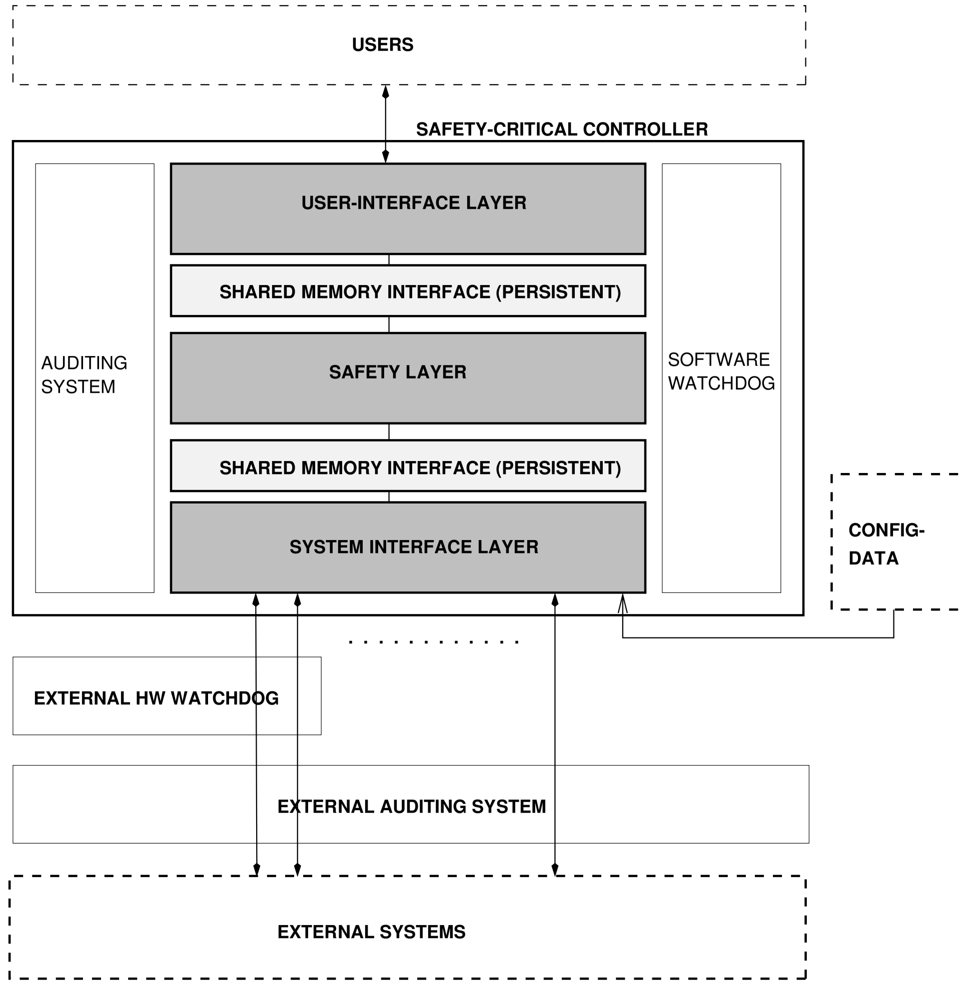
- **Funds Transfer Framework:** generic model for transaction management of bank accounts
- **Safety Architecture Framework:** generic architecture for safety-critical fail-stop controllers without controller redundancy (explained in more detail below)
- **Master-Standby Framework:** generic architecture for fault-tolerant 2-redundant master/slave system

- **N-Modular Redundancy/Voter Framework ($n \geq 3$):** n-redundant fault-tolerant systems with active replication – suitable for hard real-time applications

- **N-Modular Redundancy/Byzantine Agreement Framework ($n \geq 4$):** n-redundant fault-tolerant systems with active replication – suitable for hard real-time applications – protects against Byzantine component failures – guarantees agreement between non-faulty components

- **Time-Frame Communication/Scheduling Framework:** generic model for scheduling and communication in a network of controllers with hard real-time requirements

Example: Safety Architecture Framework



Example: Safety Architecture Framework

SAFETY LAYER

- takes safety-critical control decisions
- implements (parallel network of) real-time state machines – this can be generated automatically from formal specifications !
- inputs for state machines: abstracted events and states
- outputs of state machines: abstracted control commands and state changes

Example: Safety Architecture Framework

SYSTEM INTERFACE LAYER

- contains hardware-specific drivers for each external interface
- implements interface-specific protocols
- transforms concrete inputs from external systems into abstract events
- transforms abstract control commands from SAFETY LAYER into concrete interface data for associated external systems
- updates state information

Example: Safety Architecture Framework

USER INTERFACE LAYER

- interprets state information stored in shared memory and displays user data
- interprets user inputs, generates associated abstract events for SAFETY LAYER and updates state information stored in shared memory

Example: Safety Architecture Framework

SHARED MEMORY INTERFACES

- provides persistent storage of event lists and state information $\Longrightarrow$ possibility to restart a software layer in the proper system state as long as the operating system works correctly
- partitions state space by using several separate shared memory segments

CONFIG-DATA

- contains project-specific configuration data $\Longrightarrow$ state machines of the SAFETY LAYERS can be developed as generic specifications and instantiated with specific configuration data

Example: Safety Architecture Framework

SOFTWARE WATCHDOG

- checks life flags of each software layer
- checks integrity constraints of state information and code, so that internal errors are revealed as soon as possible

EXTERNAL HARDWARE WATCHDOG

- checks basic safety constraints on hardware interface level
- uses hardware (and possibly simple software) which is independent of possibly corrupt controller behaviour

Example: Safety Architecture Framework

AUDITING SYSTEM

- records safety-related user interactions, events, actions and internal state transitions
- useful mainly for debugging faulty controller behaviour

EXTERNAL AUDITING SYSTEM

- records safety-related I/O on hardware interface level
- uses hardware (and possibly simple software) which is independent of possibly corrupt controller behaviour – e. g. network snooping mechanism
- mandatory for legally valid proof of correct controller behaviour

Overview

1. The Notion of Dependability
2. Safety-Related Standards and V-Models
3. Modelling Safety-Critical Systems
4. Hazard Analysis and Risk Assessment
5. Design Criteria for Safety-Critical Systems
6. Validation, Verification and Test of Safety-Critical Systems

Validation, Verification and Test (VVT) of Safety Properties

- **Validation:** determine that the requirements are the right requirements and that they are complete
- **Verification:** evaluate development products to ensure their consistency with respect to applicable reference documents
- **Test:** execute implemented system components by providing specific data at their (input) interfaces, while monitoring the component behaviour.

Observe that the test of liveness properties would be impossible, since you never know when to stop waiting for the right result!

Hazard-Analysis-Driven Selection of VVT-Methods

The fundamental VVT problem . . .

- time and budget restrictions do not allow for exhaustive verification, validation and test coverage with respect to **every** system requirement

and its practical solution:

- use **conventional acceptance testing** to make sure that requirements were really implemented
- use **maximum coverage inspection/model checking/testing** for those requirements whose violation would lead to identified hazards
- invest **effort proportional to hazard severity**

Hazard-Analysis-Driven Selection of VVT-Methods

Recall:

System Hazard Analysis

- Root hazards refer to the physical model (EUC)
- Hazard analysis implies safety requirements for the safety controller

Controller Hazard Analysis

- Root hazard is "Controller violates its safety requirements"
- Software modules as leaves of the hazard analysis (e.g. fault tree analysis)
- Tree structure is induced by the controller design structure

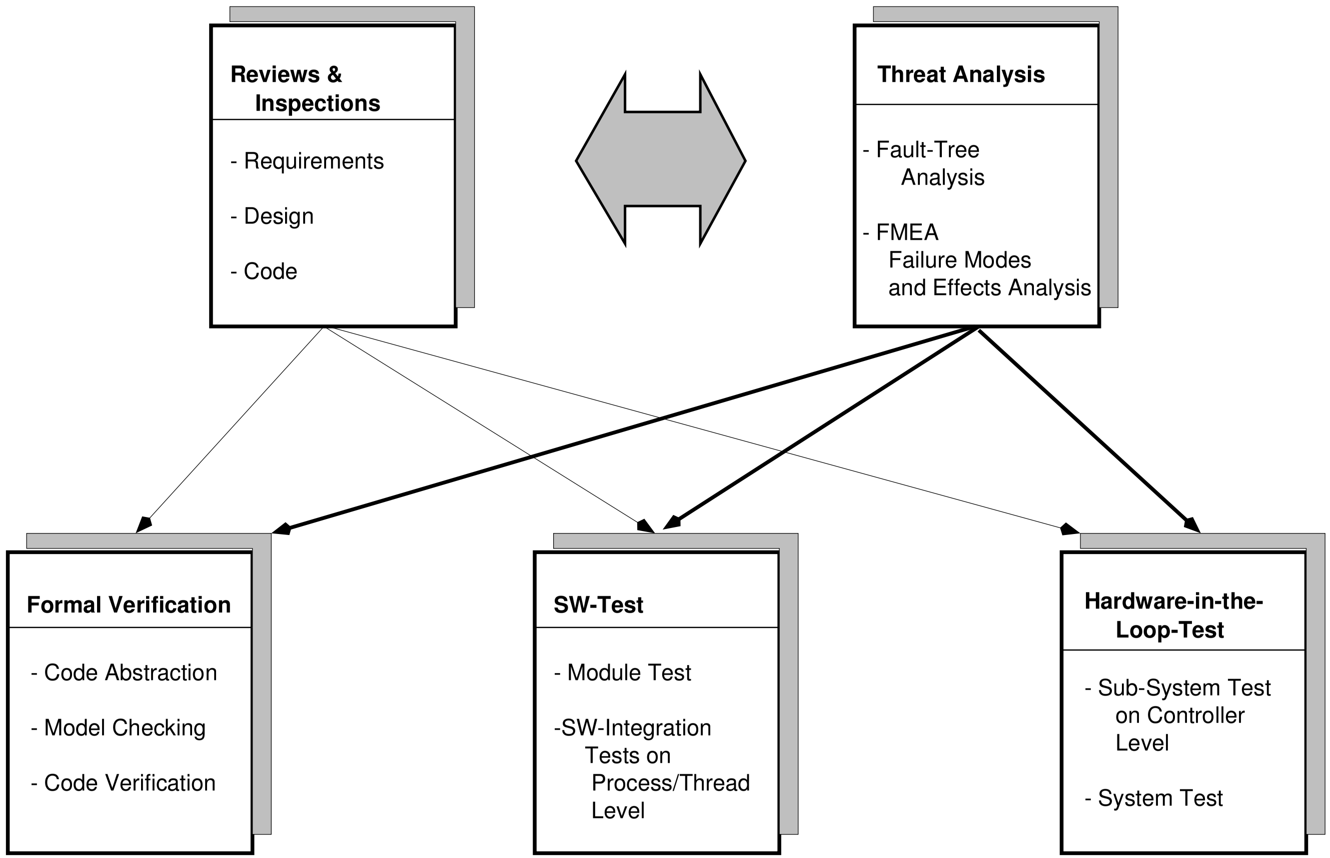
Hazard-Analysis-Driven Selection of VVT-Methods

VVT-Approach: Select VVT methods (inspections, reviews, tests and formal verification) on the basis of the controller hazard analysis !

Today's most promising VVT methods:

- interactive, structured **inspections** of requirements, design and code
- formal verification of requirements, design and code by **model checking**
- automated **hardware-in-the-loop test**

Hazard-Analysis-Driven Selection of VVT-Methods



Hazard-Analysis-Driven Selection of VVT-Methods

Example: Derive test configuration and test case from fault-tree discussed in Section 4.

Assume that events CE and $E32$ are generated by the environment and therefore cannot be prevented by the controller.

Revised requirements:

Safety Requirement 1':
$(\textbf{not}(CE) \textbf{ and } E32) \Rightarrow \textbf{not}(E31 \textbf{ and } E22)$

Safety Requirement 2':
$(\textbf{not}(CE) \textbf{ and } \textbf{not}(E32)) \Rightarrow \textbf{not}(E31 \textbf{ and } E22 \textbf{ and } E33)$

Hazard-Analysis-Driven Selection of VVT-Methods

Example – continued.

Revised requirements lead to a test configuration where

- CE and $E32$ can be **controlled** by the test system simulating the operational environment of the system under test
- requirements 1' and 2' can be **checked** by the test system
- test coverage to be achieved can be set **proportional to severity of hazard E1**

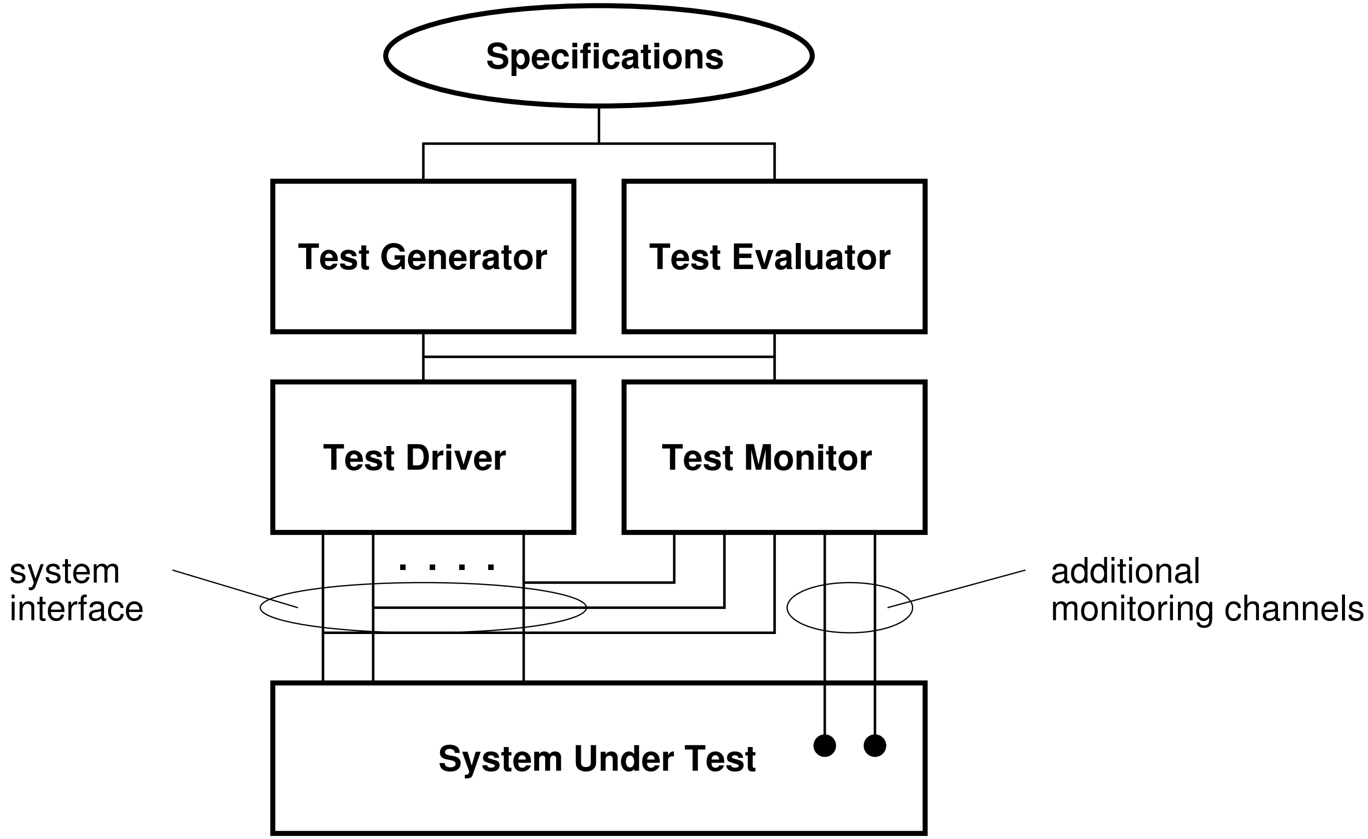
Remark: When to use . . .

- **inspections:** logical checks of sequential algorithms localised in isolated functions
- **model checking:** logical check of synchronisation mechanisms, distributed algorithms and communication protocols involving several parallel processes
- **hardware-in-the-loop test:** check of the proper integration of logically correct software on the target system hardware

Model Checking

- specification $SPEC$ is expressed in one of various formalisms allowing to describe systems of parallel communicating (timed) state machines
- correctness condition to be checked is expressed by logical formula F
- model checker performs exhaustive state analysis to investigate whether F holds in every state of $SPEC$

Components of Test Automation System for Reactive RT-Systems



Conclusion

The following measures will considerably improve the cost effectiveness and the trustworthiness of the development process for safety-critical control systems:

Development process related measures:

- derivation of safety requirements from formal hazard analysis
- elaboration and systematic use of safety-related design patterns and frameworks
- "design for testability"

Conclusion

Quality assurance related measures:

- design of VVT measures driven by hazard analysis
- combined/complementary use of
  - interactive, structured inspections
  - formal verification by model checking
  - automated testing
- use of application-dependent quality measures

Management process related measures:

- use of safety-centered V-models
- total quality management campaigns to increase safety awareness