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

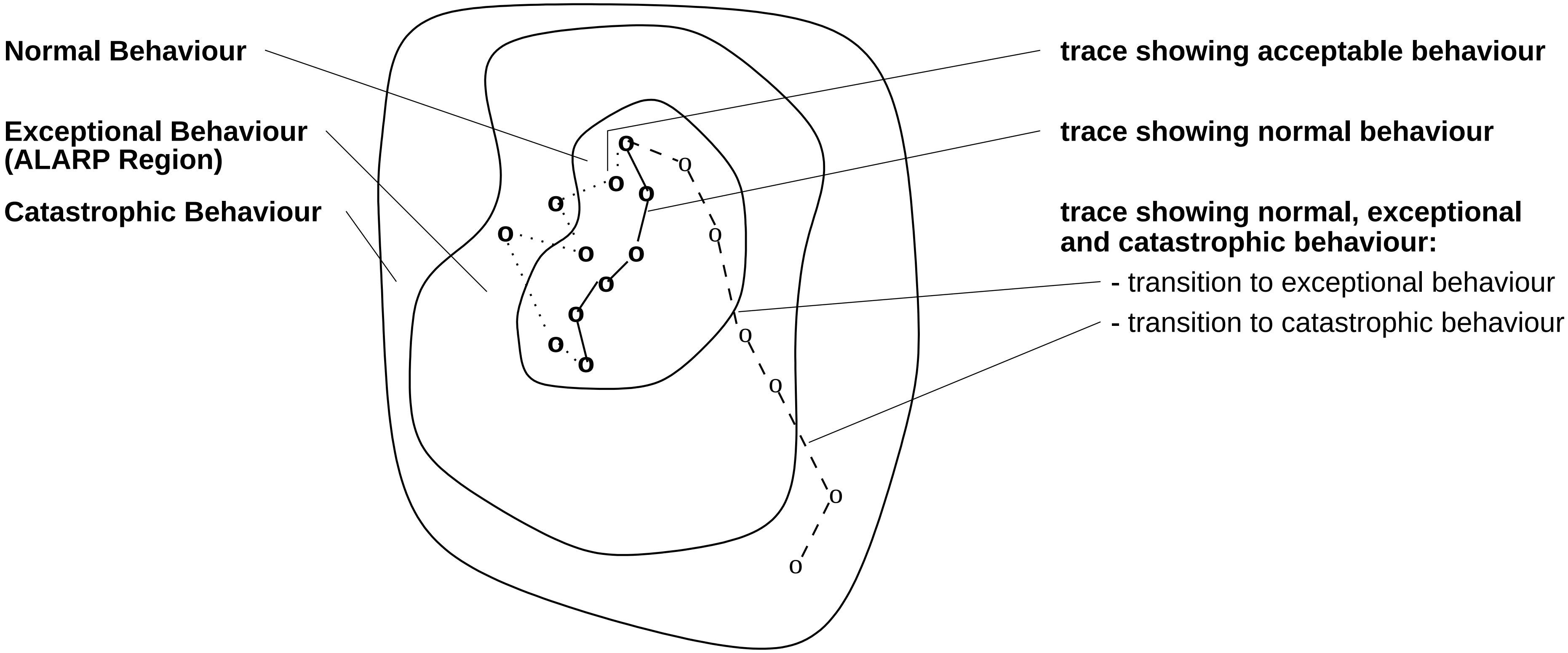
In computer science, specifications are classified as containing

- **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.

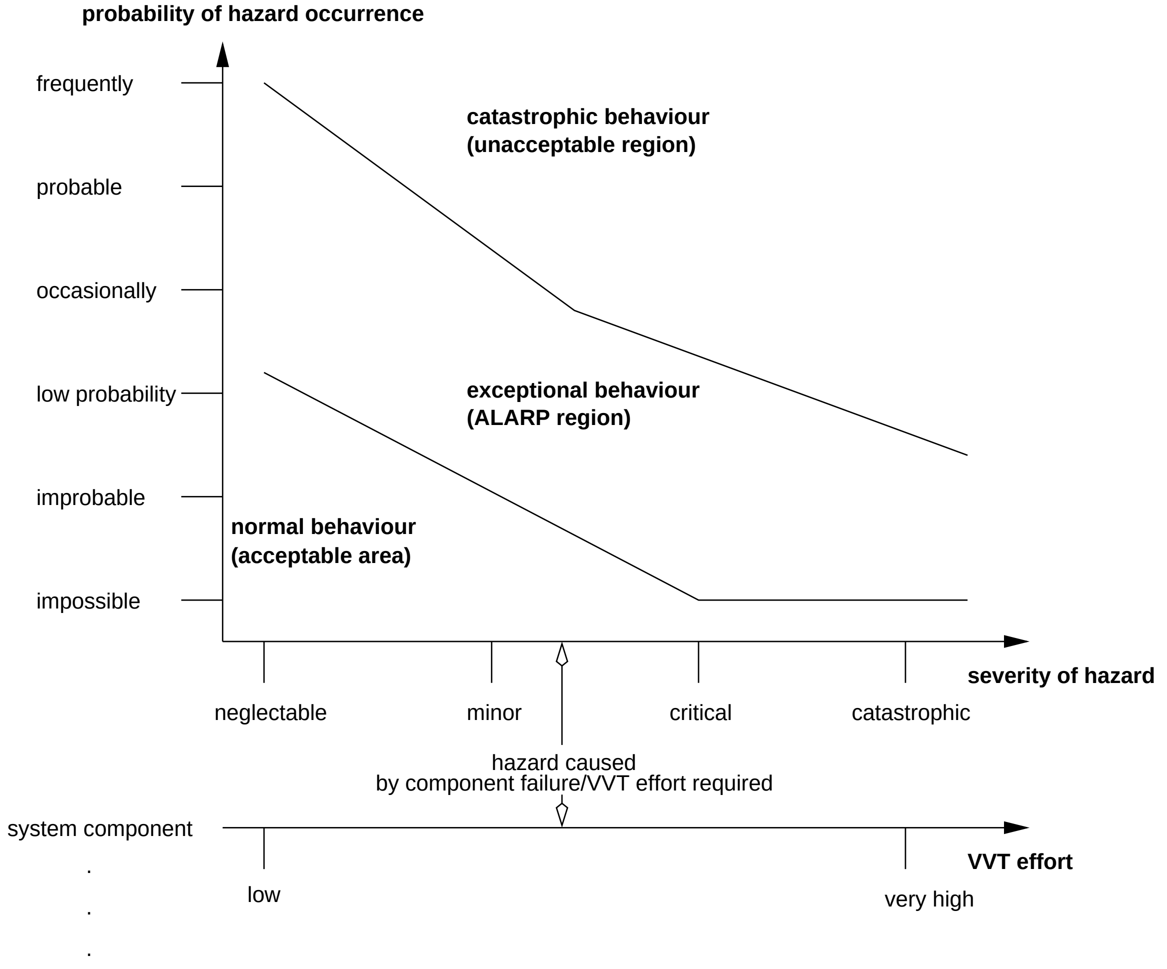
Safety-Related Standards – Common Understanding

- 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**

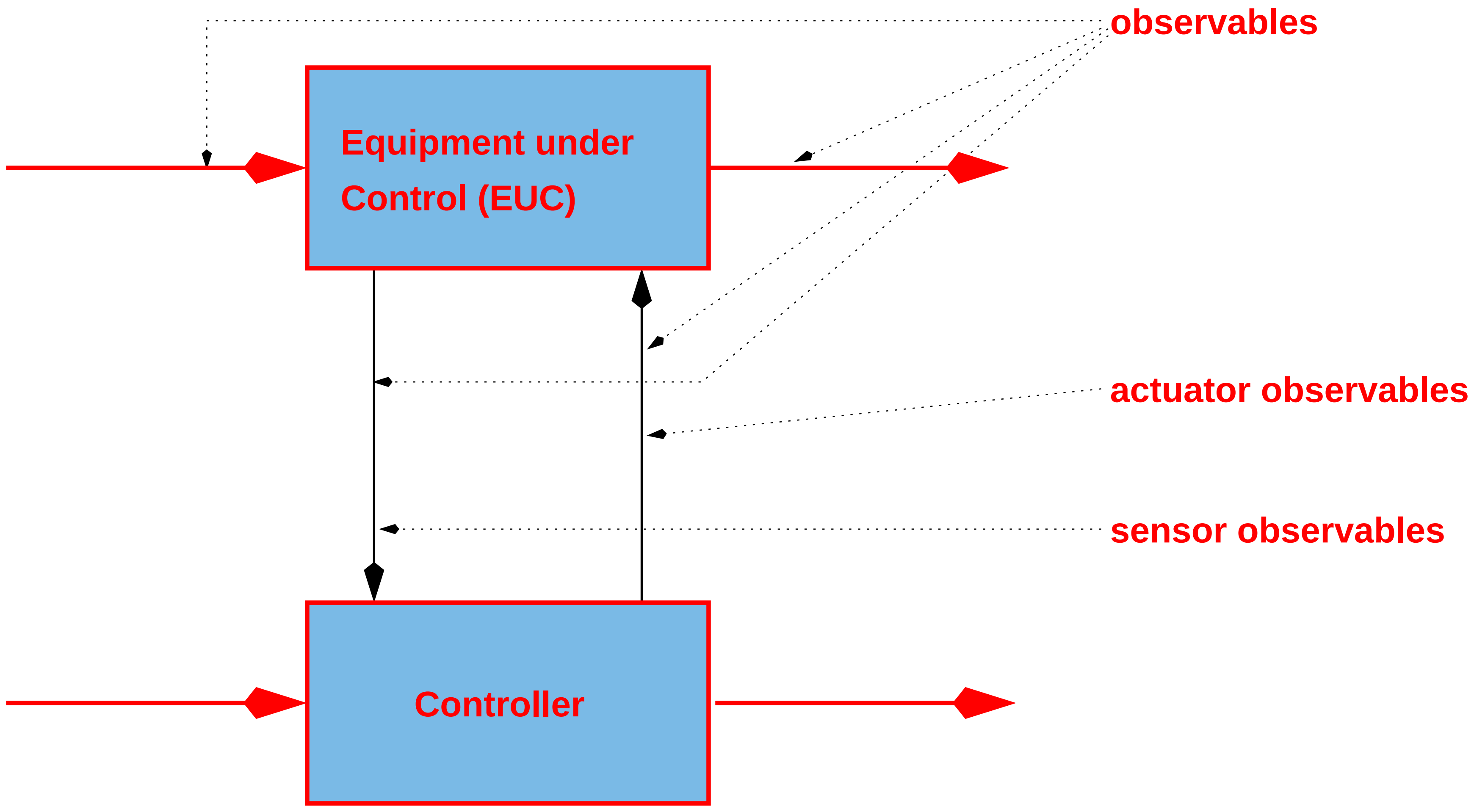
Safety-Related Standards – Risk Diagram



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	+	+	·	-	·	+

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

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?

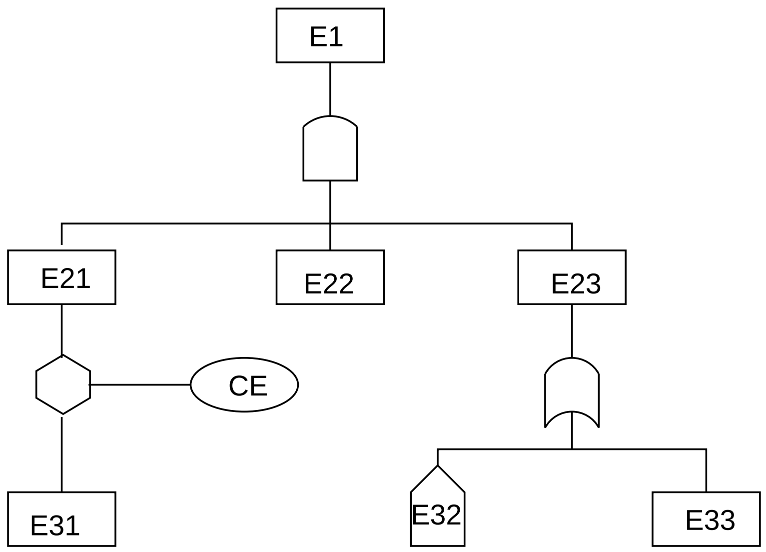
Risk Assessment: What are the probabilities that a hazard occurs, provided the hazard analysis is consistent and complete?

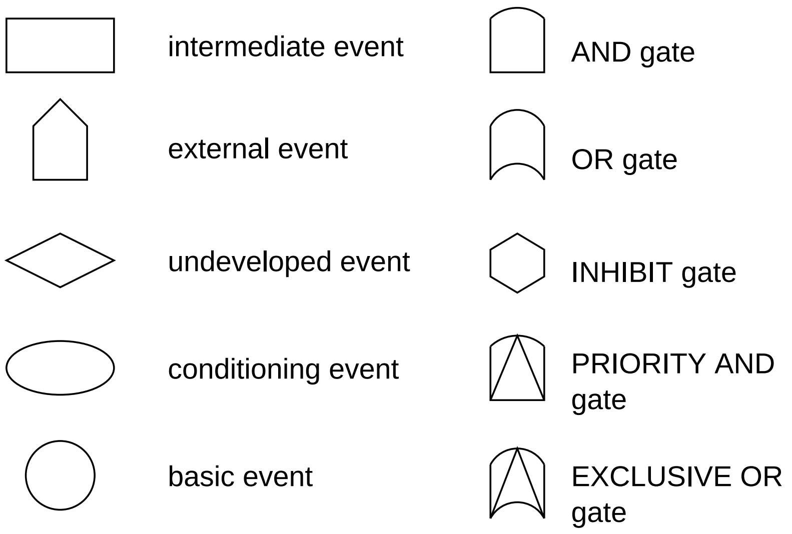
Modelling Techniques for Hazard Analysis

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

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

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

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

Deriving Safety Requirements from Hazard Analysis

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

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

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

Reliability: Probability for a component to operate correctly in interval $[0, t]$.

$$R(t) = \int_0^t \varrho(t)\,dt, \quad \text{with } \varrho(t) \text{ a density function.}$$

$R(t)$ 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 time t.

Risk Assessment - General Definitions

Unreliability:

Probability for a component to fail in interval $[0, t]$.

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

$Q(t)$ 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 $[0, t]$.

Risk Assessment - General Definitions

Failure Density:

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

$$f(t) = \frac{dF(t)}{dt}$$

for differentiable F.

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 $\Delta t \to 0$:

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

Risk Assessment - General Definitions

Mean Time To Failure:

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

$$MTTF = \int_0^\infty R(t)\,dt$$

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.

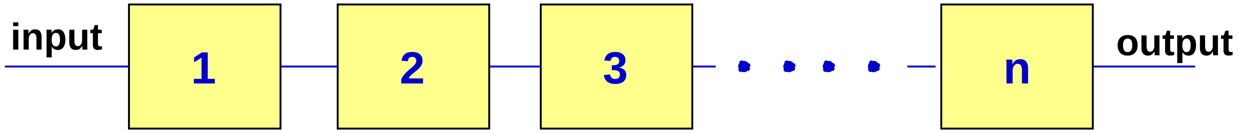
$$\text{Availability} = \frac{MTTF}{MTTF + MTTR} = \frac{MTTF}{MTBF}$$

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

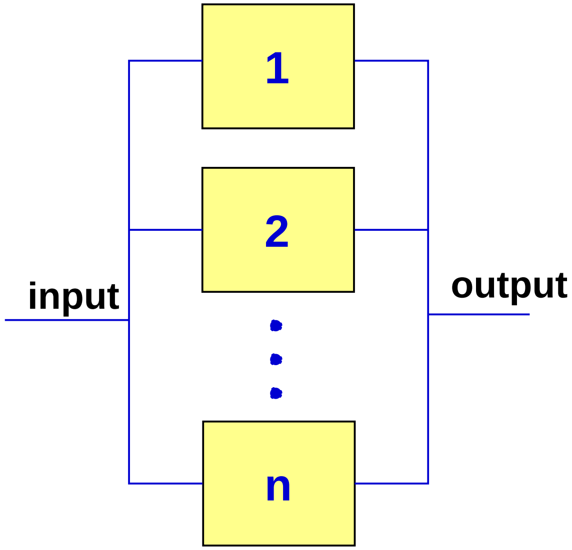
$$R(t) = \prod_{i=1}^{n} R_i(t)$$

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

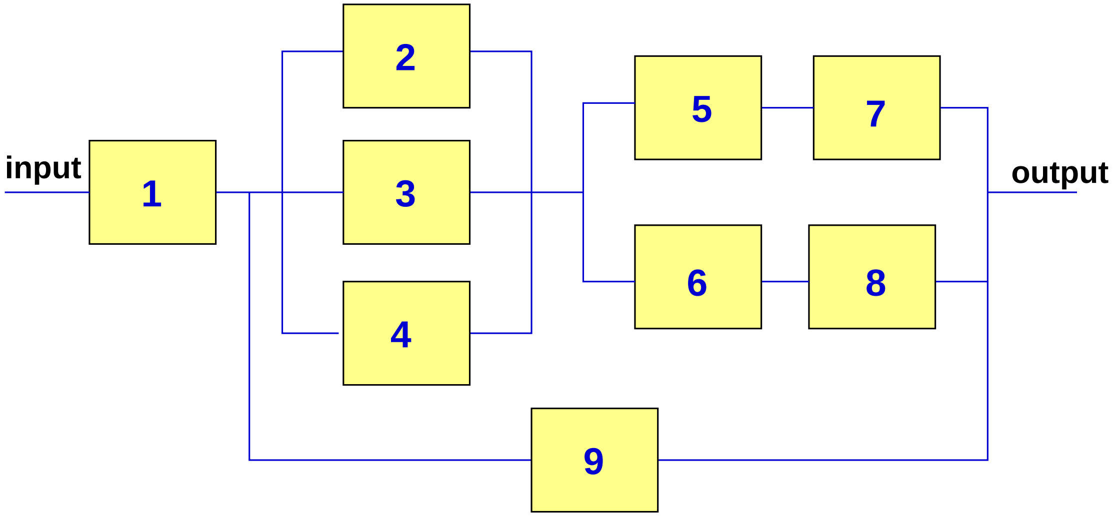
$$R(t) = 1 - \prod_{i=1}^{n}(1 - R_i(t))$$

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

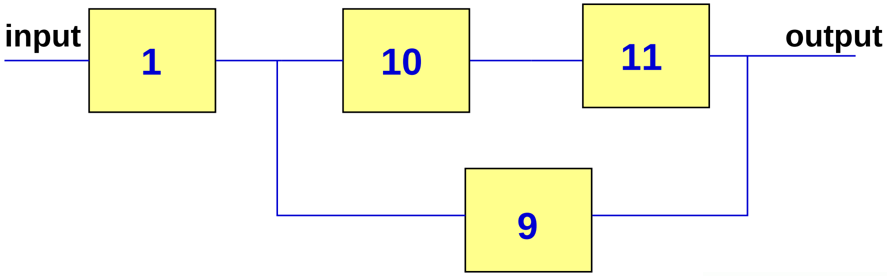
$$R(t) = 1 - (1 - 0.999)^3 = 0.999\,999\,999$$

Reliability Modelling - Combinational Models

Series-parallel combinations



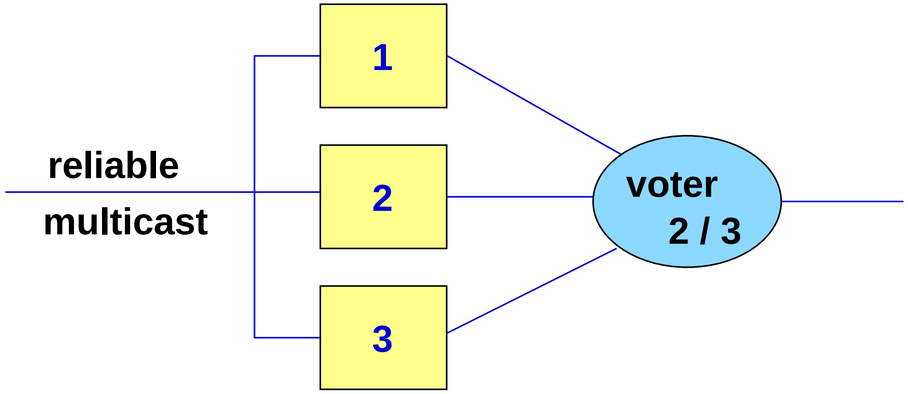
From the above formulas, the system reliability is easily computed in two steps, using the following abstraction:



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.



$$\begin{aligned} R(t) &= R_1(t)R_2(t)R_3(t) + (1 - R_1(t))R_2(t)R_3(t) \\ &\quad + R_1(t)(1 - R_2(t))R_3(t) + R_1(t)R_2(t)(1 - R_3(t)) \end{aligned}$$

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 $R(t) = 0,352$.

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.

Design Criteria for Safety-Critical Systems

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:

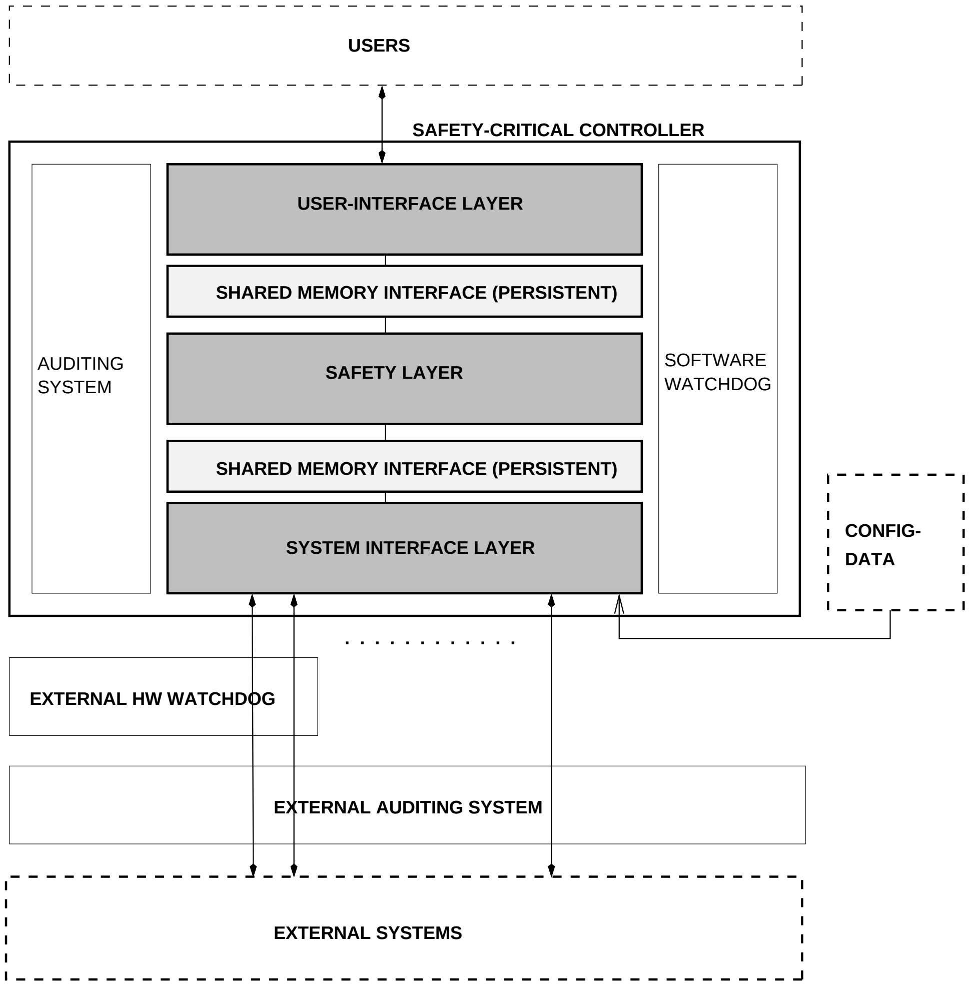
- **Funds Transfer Framework:** generic model for transaction management of bank accounts
- **Safety Architecture Framework:** generic architecture for safety-critical fail-stop controllers without controller redundancy (explained in more detail below)
- **Master-Standby Framework:** generic architecture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

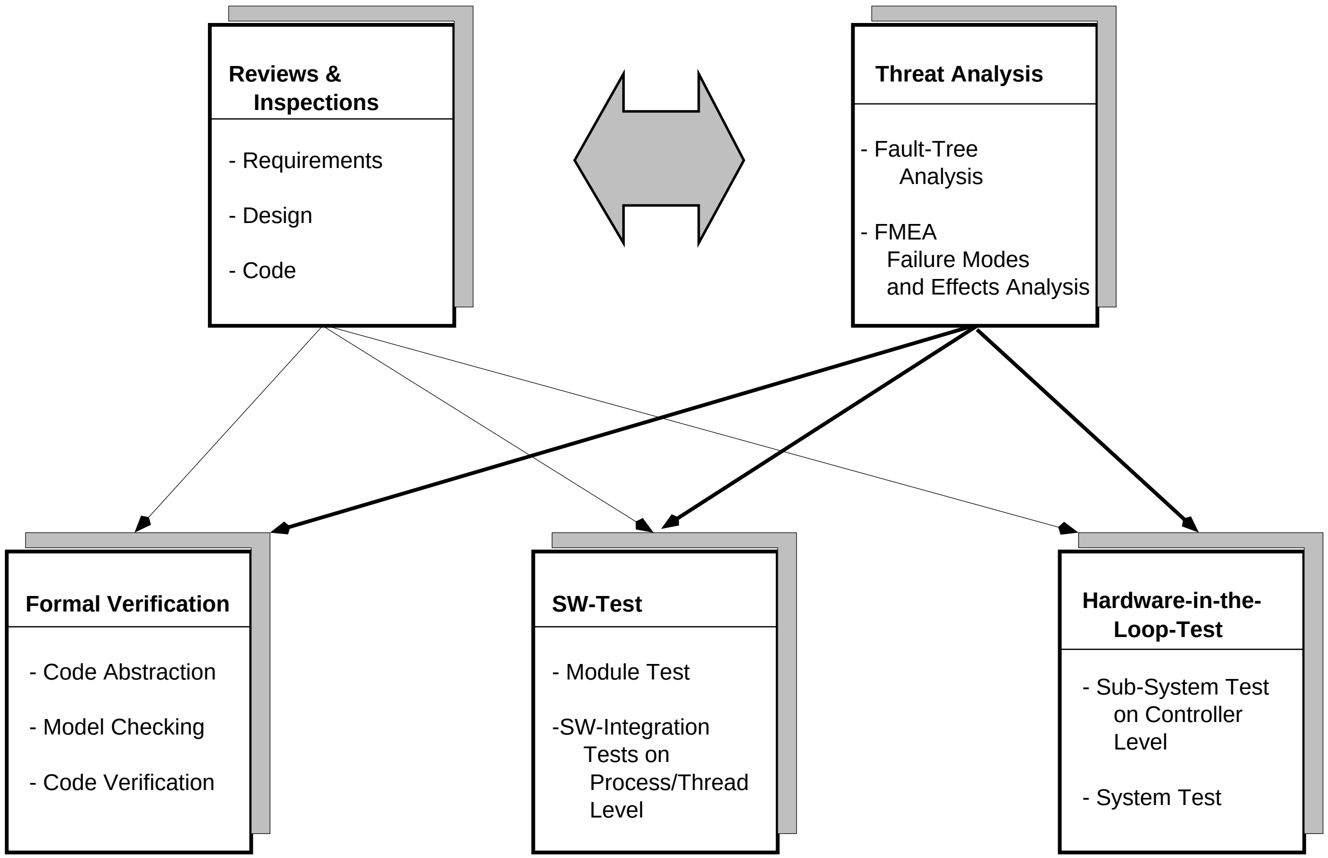
Hazard-Analysis-Driven Selection of VVT-Methods

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**

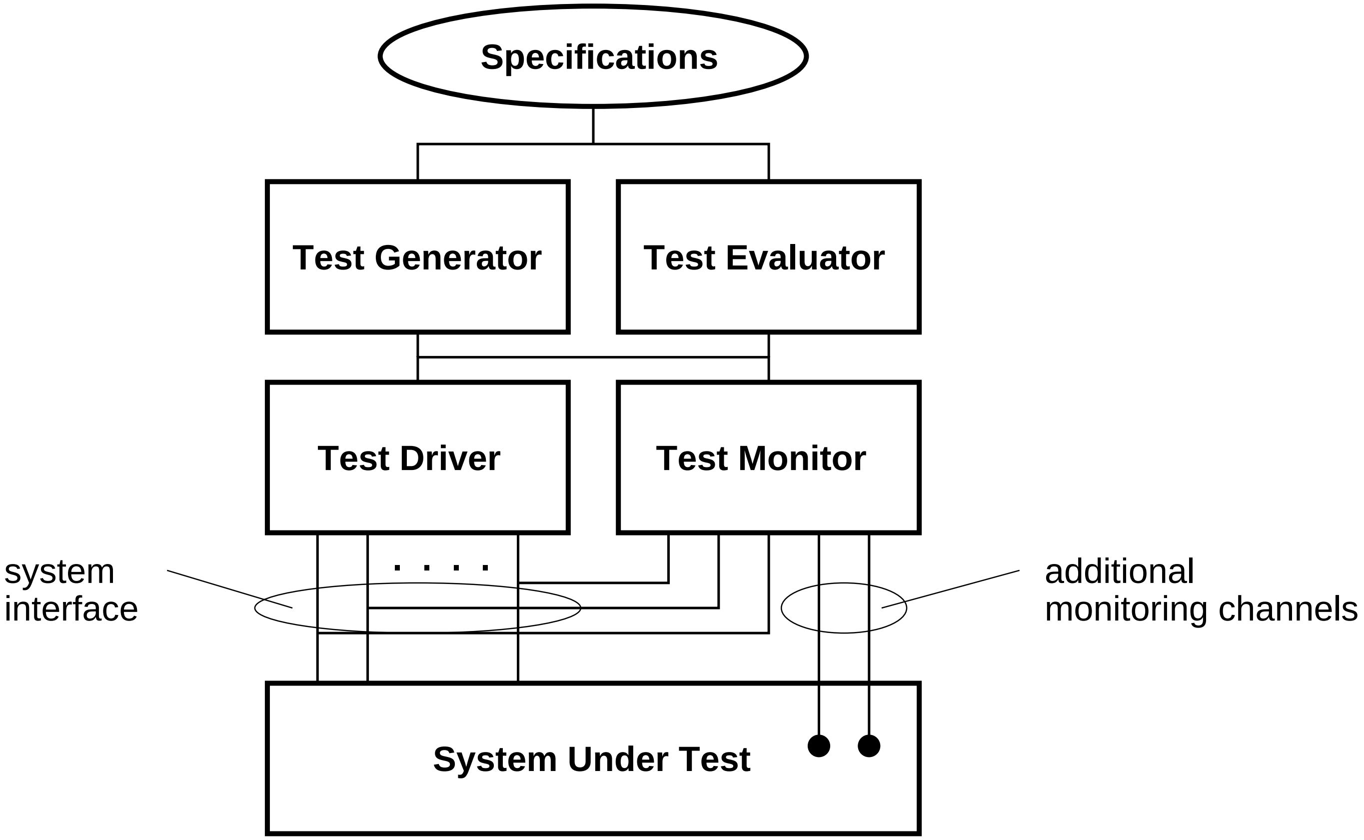
Remark: When to use ...

- **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