Graceful degradation might have multiple levels of reduced functionality.
• or in case of a detected error-condition: dynamic parts which are swapped in.
• static parts of the system and connected via a voting/masking/comparing system

Sophisticated synchronization and communication systems.
• the same specification
• different computer systems (CPUs, buses, memory systems, drives)
• different operating systems
• ... environments (N-Version programming)
• and by restricting the communication between the different developer teams

For some high-risk systems this approach is applied in forms of redundant sub-systems with:
• watch-dog timers, limit switches, additional physical sensors, transient-recording-systems (emergency system dump), overload-backup-systems, or even in-circuit emulators.

The system continues to operate in the presence of ‘foreseeable’ error conditions,
Targeted failure probability: $< 10^{-10}$ (e.g. UK Seizewell B nuclear reactor (emerg.): $< 10^{-9}$)

System identification
• Investigate the system
• physical applications and specifications
• constraints of the employed controller network
• constraints of the underlying run-time system
• dynamic application specifications (requested real-time behaviour)

Achieving reliability

Fault avoidance at hardware-level:
• use reliable hardware components — consider the environmental demands!
• use an adequate (hardware) system design — shock, humidity, interference,
• ensure proper assembly and encapsulation — weak connectors, bad connections, ...
• Fault avoidance at software design level:
• strict system specifications (employ format methods if applicable)
• use proven software-engineering and design methodologies
• Employ languages and run-time environments with reasonable support for the requirements.

Fault removal
• find and remove errors from the previous stage.
• Team programming methods like extreme programming or rigorous testing may help here.

Fault tolerance
• Component & communication system failures

Real-Time & Embedded Systems
Reliability
Faults on different levels
• Inconsistent or inadequate specification
  — very frequent source for disastrous faults
• Software design errors
  — very frequent source for disastrous faults
• Component & communication system failures
  — use and mostly predictable

Faults in the time domain
• Transient faults
  — many communication system failures, electric interferences, etc.
• Intermitent faults
  — transient errors which occur more than once (e.g. overheating effects)
• Permanent faults
  — stop in the system until they are repaired by some means

Real-Time & Embedded Systems
Reliability
System identification

Real-Time & Embedded Systems
Reliability
Fault avoidance
Fault avoidance at software design level:
• use reliable hardware components — consider the environmental demands!
• use an adequate (hardware) system design — shock, humidity, interference,
• ensure proper assembly and encapsulation — weak connectors, bad connections, ...
• Fault avoidance at software design level:
• strict system specifications (employ format methods if applicable)
• use proven software-engineering and design methodologies
• Employ languages and run-time environments with reasonable support for the requirements.

Hardware redundancy
• any hardware redundancy adds to the overall system complexity!

Real-Time & Embedded Systems
Reliability
Fault detection

Real-Time & Embedded Systems
Reliability
Fault tolerance

Real-Time & Embedded Systems
Reliability
Fault prevention

Real-Time & Embedded Systems
Reliability
Terminology

References for this chapter
N-version programming — Voting issues

- **Integer arithmetic:** integer (or any discrete sub-type) based results will be identical ✔
- **Real arithmetic:** Real-valued results will usually be different ☞ Comparisons need to consider tolerances.
- **Slight:** if processor is not fully continuous (thresholds, quantizations, bifurcations) ☞ Comparisons need to revalidate the whole process in order to evaluate similarities ✔/identical ☞ specify the system.
- **Multiple solutions:** the solution space still allows for multiple correct, but different solutions ☞ progress the system.

Dynamic redundancy — Error diagnosis

**Confinement:**
- How to avoid the transfer of fault-effects between system parts.
- **Modular decomposition**
- **Partial effects**
- **Global:** check system structural integrity (e.g., lack of file-systems)
- **Assessment:**
  - **resulting from the location of the detected error state and the possible paths traversed by the system which are all leading to this error state.
  - **a fine-granular system structure (error-confinement) limits the length of these possible paths.

Dynamic redundancy — Error recovery

**Backward error recovery:**
- set checkpoints and stop the system at each passing of a checkpoint.
- how can system consistent checkpoints be ensured?
- if a system state is changed back to the last consistent checkpoint.
- **forward error recovery:**
  - methods of choice for most error critical parts of real-time and embedded systems.

Reliability

**Reliability**

- **Safety**
  - Freedom from those conditions that can cause death, injury, occupational illness, damage to (or loss) of equipment (or property), or environmental harm (Jaffe, 1986).
  - Cannot be directly measured.
  - **Safety**
  - **availability** — can be used.
  - **Reliability** — can be measured.
  - **Safeness** — can be measured.
  - **Faults** — can be measured.
  - **Dependability** — can be measured.
  - **Integrity** — can be measured.
  - **Maintainability** — can be measured.

- **Safety**
  - Freedom from those conditions that can cause death, injury, occupational illness, damage to (or loss) of equipment (or property), or environmental harm (Jaffe, 1986).
  - Cannot be directly measured.
  - **Safety**
  - **availability** — can be used.
  - **Reliability** — can be measured.
  - **Safeness** — can be measured.
  - **Faults** — can be measured.
  - **Dependability** — can be measured.
  - **Integrity** — can be measured.
  - **Maintainability** — can be measured.

- **Safety**
  - Freedom from those conditions that can cause death, injury, occupational illness, damage to (or loss) of equipment (or property), or environmental harm (Jaffe, 1986).
  - Cannot be directly measured.
  - **Safety**
  - **availability** — can be used.
  - **Reliability** — can be measured.
  - **Safeness** — can be measured.
  - **Faults** — can be measured.
  - **Dependability** — can be measured.
  - **Integrity** — can be measured.
  - **Maintainability** — can be measured.

- **Safety**
  - Freedom from those conditions that can cause death, injury, occupational illness, damage to (or loss) of equipment (or property), or environmental harm (Jaffe, 1986).
  - Cannot be directly measured.
  - **Safety**
  - **availability** — can be used.
  - **Reliability** — can be measured.
  - **Safeness** — can be measured.
  - **Faults** — can be measured.
  - **Dependability** — can be measured.
  - **Integrity** — can be measured.
  - **Maintainability** — can be measured.

- **Safety**
  - Freedom from those conditions that can cause death, injury, occupational illness, damage to (or loss) of equipment (or property), or environmental harm (Jaffe, 1986).
  - Cannot be directly measured.
  - **Safety**
  - **availability** — can be used.
  - **Reliability** — can be measured.
  - **Safeness** — can be measured.
  - **Faults** — can be measured.
  - **Dependability** — can be measured.
  - **Integrity** — can be measured.
  - **Maintainability** — can be measured.
Real-Time & Embedded Systems

Reliability

• Terminology
  - Faults, failures, fault-tolerance

• Faults
  - Fault avoidance, removal, prevention = Fault tolerance

• Redundancy
  - Static (RBX, NMR) and dynamic redundancy

• Newcomer programming and dynamic redundancy in software design

• Reduce & Formulate
  - API level

• Real-Time Logic

Real-Time & Embedded Systems

Reliability

• Assertions on sequenced and orders of states
  - employ predicate logic & a set of new operators:
    • \( \mathcal{Q} A \): is true for all future states
    • \( \mathcal{A} A \): is true for the following state

Examples:
\[ \mathcal{Q} \text{Collision Warning} \Rightarrow \mathcal{Q} \text{Collision Prevention} \]
\[ \mathcal{Q} \text{Collision Prevention} \Rightarrow \mathcal{Q} \text{Collision Warning} \]

assuming that there is a sequence of distinguishable states (or "time").

Real-Time & Embedded Systems

Reliability

Linear Temporal Logic of Real Numbers (LTR)

\[ e = \begin{cases} p_1 & \text{if } p \in E(t) \\ \exists i > 0 : p_{i+1} \neq p_{i+2} \text{ and } \forall j \neq i : p_j = p_{i+2} & \text{if } (t, t) \end{cases} \\ (t, t) \end{cases} \\ (t, t) \end{cases} \\ (t, t) \end{cases} \\ (t, t) \end{cases} \\ (t, t) \end{cases} \\ (t, t) \end{cases} \]

\([t, t] \text{ is eventually true if } t \]