TTTech

Part 2:

Basic concepts and terminology

Def.: Dependability (Verlässlichkeit)

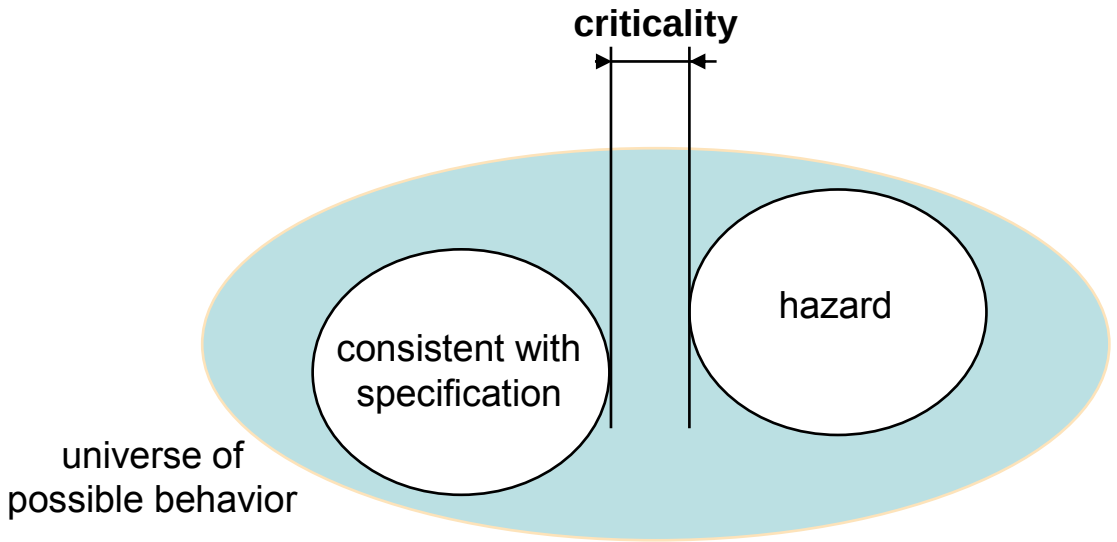
is defined as the trustworthiness of a computer system such that reliance can justifiably be placed on the services it delivers. [Laprie 1992, IFIP WG 10.4]

Two aspects of *reliance:*

- reliance that the system performs according to its service specification
- reliance that the system avoids hazards, i.e., behaviour which may lead to undesired consequences

Criticality of dependable systems

- Informally speaking, the “closer” the service specification and hazards are, the higher is the criticality of a system.
- this distance defines an application-inherent fault-tolerance margin

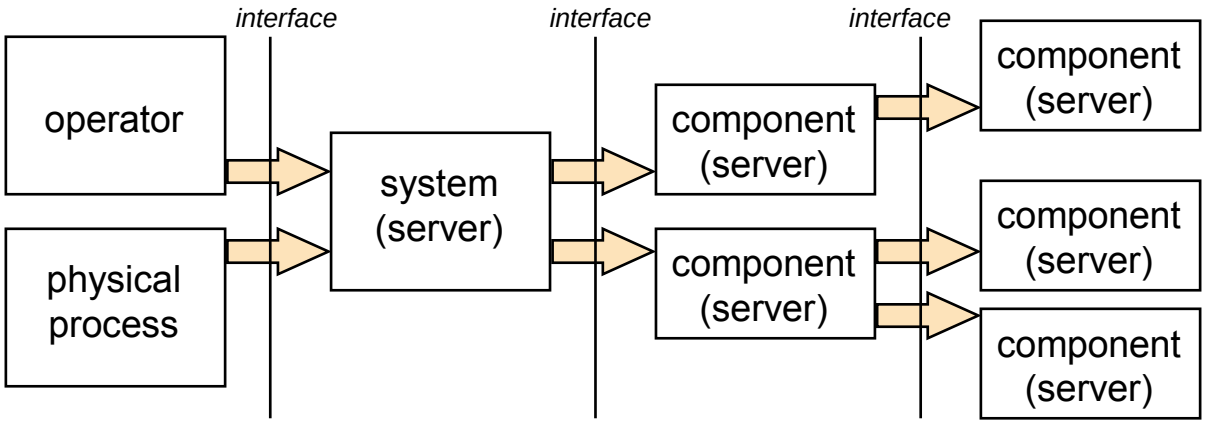


System

- a system is defined by its structure
- and by the behavior of its constituting components

Recursive nature of the *depends* (⇨) relation

- service users depend on the services provided by the system (server)



Attributes of dependability

- **reliability** (Funktionsfähigkeit)
  dependability with respect to continuity of service
- **availability** (Verfügbarkeit)
  dependability with respect to readiness for usage
- **safety** (Sicherheit)
  dependability with respect to avoidance of catastrophic consequences
- **security** (Vertraulichkeit)
  dependability with respect to prevention of unauthorized access and/or handling of information

Quantitative definitions for the attributes of dependability

- **Reliability** $R(t)$
  is the probability that the system will conform to its specification throughout a period of duration *t*.
- **Availability** A or V
  is the percentage of time for which the system will conform to its specification.
- **Safety** $S(t)$
  is the probability that the system will not exhibit a specified undesired behavior throughout a period of duration *t*.
- **Security**
  no quantitative definition possible,
  – Secrecy Who can read information?
  – Integrity Who can change things and how?
  – Availability cf. 2nd point

Additional attributes of dependability

- Usability (Verwendbarkeit)
- Recoverability (Wiederherstellbarkeit)
- Maintainability (Wartbarkeit)
- Extendability (Erweiterbarkeit)
- Trustability (Vertrauenswürdigkeit)
- *others*

Reliability vs. safety

- correct service
- functional specification
- no severe consequences in case

- avoidance of hazards
- non-functional specification
- catastrophic consequences of failures

- different costs of failures
- different methodologies for system construction
- conflicting goals
- safety $\subseteq$ reliabilty

Reliability vs. safety

- **Railway signalling:**
  - red signal is a safe system state
  - safe system state is unreliable
  - safety ≠ reliability
- **Fly-by-wire airplane control:**
  - after take off there is no safe (non-functional) system state
  - safety ≈ reliability
    (degraded modes of operation are possible)

- often there is a conflict between safety and reliability

Reliability vs. availability

- **Factory automatization:**
  - the computer has to assure proficient manufacturing
  - availability is most important parameter
  - reliability is not that important
- **Satellite:**
  - once put into operation there is no possibility for maintenance
  - mission reliability is most important parameter

- availability is only relevant if maintenance is possible

Specification

The definition of all dependability attributes is based on specifications. A *good* specification must be:

- exact
- consistent
- complete
- authoritative

Importance of specification

Together with the analysis of possible behavior and its consequences, system specification is the most difficult part of building a dependable system.

Multiple levels of specifications

To consider the different aspects and attributes of dependable systems, usually different levels of specifications exists.

An example

level	*specification*
functional	“all commands have to be carried out correctly”
reliability	“either correct commands or warning indicator”
safety	“recorded info may not be corrupt”

The underground train

- an electronically controlled underground train had the following buttons:
  - to open and close doors
  - to start the train
- it was specified that “the train only may start if and only if the start button is pressed and all doors are closed”
- a driver blocked the start train button by means of a tooth pick to start the train immediately if the doors were closed

What happened?

- one day a door was blocked and the driver went back to close the door, and of course, the train left the station without the driver

What went wrong?

- it was the drivers fault to block the start button with a tooth pick
- but it was also a specification fault since the correct specification should have read: “the train only may start if and only if the start button changes it state to start and all doors are closed”
- in that example it made a big difference whether *state* or *event*-semantics are implemented

Requirements for automotive electronics

- Reliability 0 *km*/0 *h* failures: < 500 10^{-9}
  failures within 1st year: < 1000 (500) 10^{-9}
- often no difference between reliability and safety
- System lifetime 3500 *h*
- Warranty ≥ 1 year, spare parts ≥ 10 years
- Environmental conditions: -40 – +85°*C*
- Vibration 10 *Hz* – 1 *kHz*, noise 5 *g*, sine 2 – 5 *g*
- Shock 30 *g*
- Supply voltage 8 – 16 *V*,
  start with 6 *V* (-40 – 85°*C*), 18 *V* for 2 *h*, 24 *V* for 1 *min*,
  reversed polarity 13.5 *V* for 1 *min*

Stress tests for automotive electronics

- Function test
  8, 13.5, 16 *V* at -40, 25, 85 °*C*
- Heat test
  85 ± 2 °*C* for 16 *h* at 16 *V* and 6000 *rpm*
- Cold test
  -40 ± 3 °*C* for 2 *h*
- Heat storage
  85 ± 2 °*C* for 504 *h*
- Temperature shock
  -40 to 85 °*C* changeover time 30 *s* for 25 times
- Temperature change
  -40 to 85 °*C*, 3 ± 0.6 °*C*/*min* for 2 cycles

Safety requirements for aircrafts

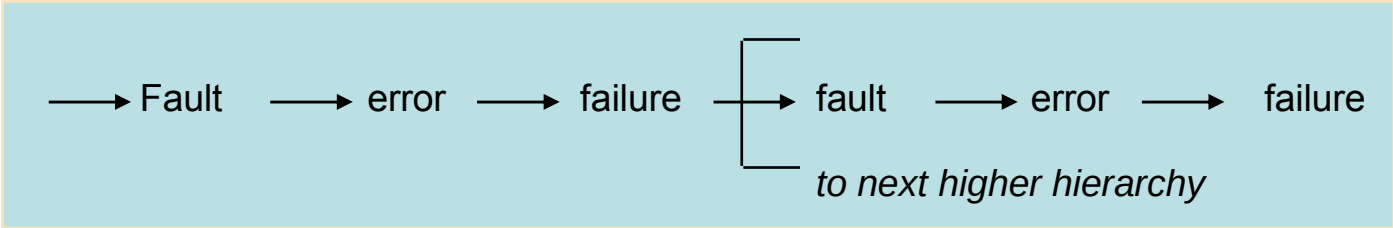
AMJ 25.1309 classifies failure conditions by their severity and specifies a maximum permissible probability

- **Minor:** 10^{-5} per flight hour or greater
  no significant reduction of aeroplane safety, a slight reduction in the safety margin
- **Major:** between 10^{-5} and 10^{-7}
  significant reduction in safety margins or functional capabilities, significant increase in crew workload or discomfort for occupants
- **Hazardous:** between 10^{-7} and 10^{-9}
  large reduction in safety margins or functional capabilities, causes serious or fatal injury to a relatively small number of occupants
- **Catastrophic:** less than 10^{-9}
  these failure conditions would prevent the continued safe flight and landing of the aircraft

Impairments to dependability

- **failure** (Ausfall):
  Deviation of the delivered service from compliance with the specification.
  (Transition from correct to incorrect service delivery)
- **error** (Fehlzustand):
  Part of the system state which is liable to lead to a failure.
  (Manifestion of a fault in a system)
- **fault** (Fehlerursache):
  Adjudged or hypothesized cause of an error.
  (Error cause which is intended to be avoided or tolerated).

Fault/failure chain



fault → error

- a fault which has not been activated by the computation process is *dormant*
- a fault is *active* when it produces an error

error → failure

- an error is *latent* when it has not been recognized
- an error is *detected* by a detection algorithm/mechanism

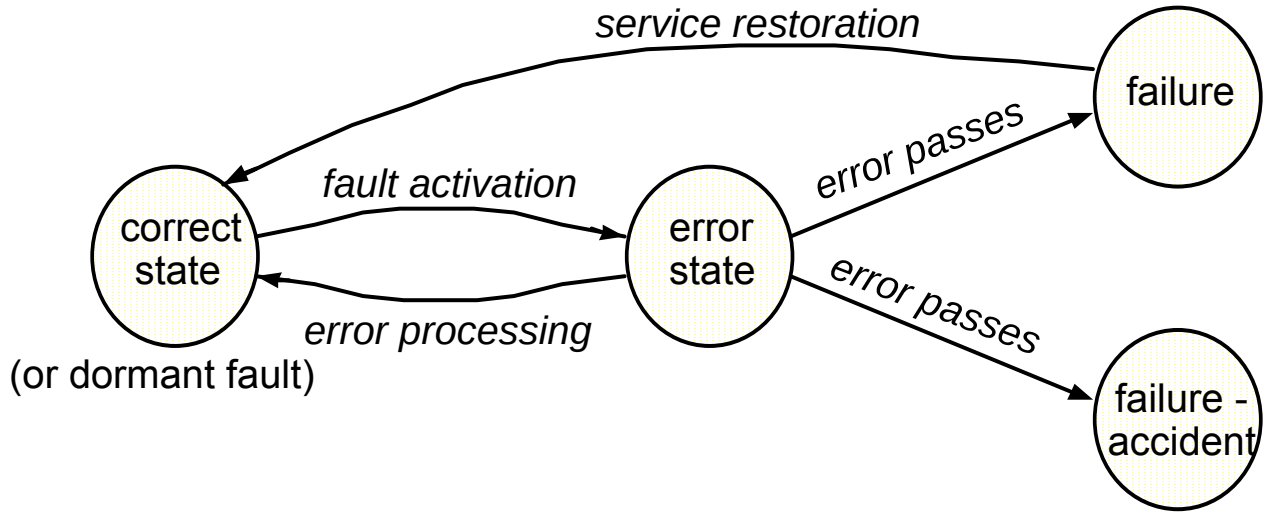
failure → fault

- a failure occurs when an error “passes through” and affects the service delivered
- a failure results in a fault for the system which contains or interacts with the component

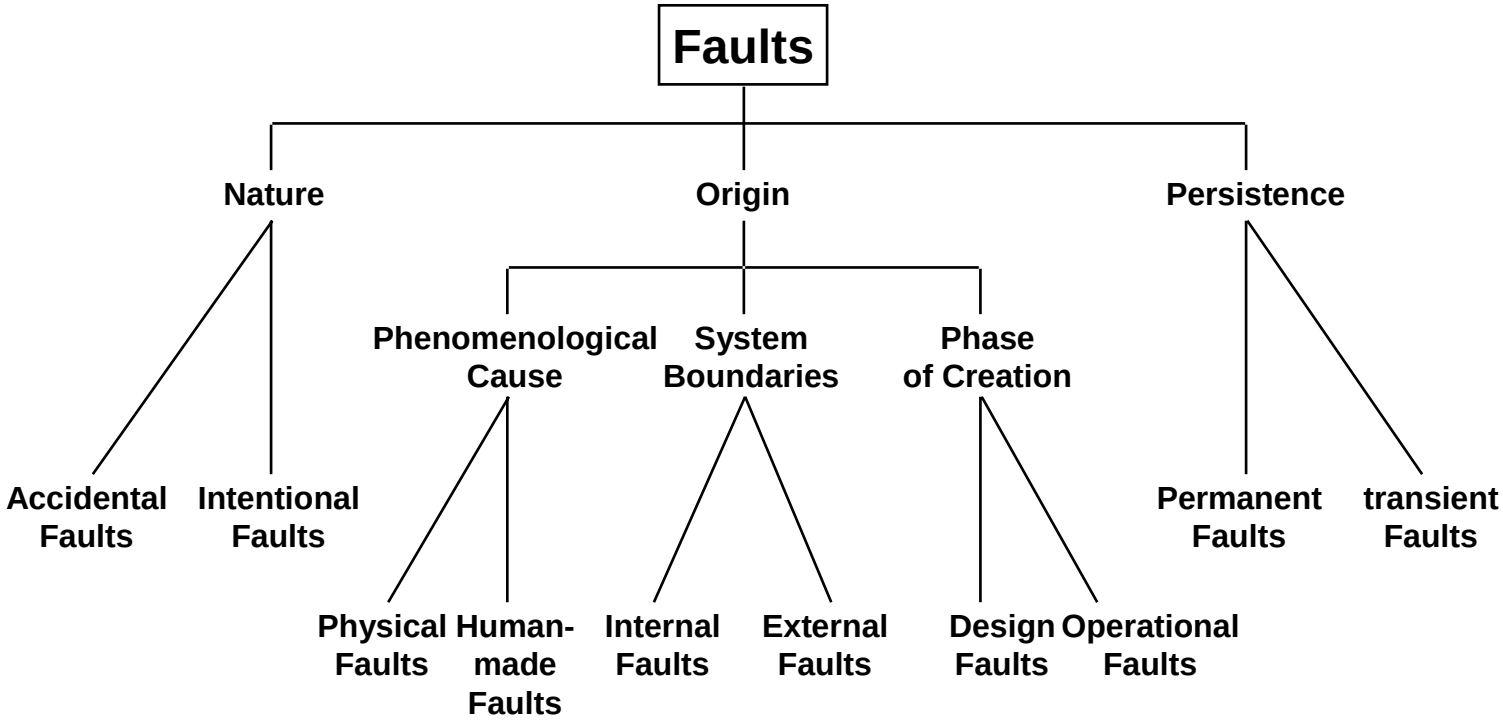
Examples for fault/failure chain

- **Program error (software):**
  - a dormant *fault* in the written software (instruction or data)
  - upon activation the fault becomes active and produces an *error* (system state)
  - if the erroneous data affects the delivered service, a *failure* occurs
- **Electromagnetic interference (hardware):**
  - leads to *faulty* input value (either digital or analog)
  - by reading the input the fault becomes active and produces an *error*
  - if the erroneous input value is processed and becomes visible at the interface a *failure* occurs

Fault/failure state transition chart



Classification of faults



Classification of Faults

Fault	Nature		Origin						Persistence	
			Phenomenological cause		System Boundaries		Phase of creation			
	Accidental Fault	Intentional Faults	Physical Fault	Human made Fault	Internal Fault	External Fault	Design Fault	Operational Fault	Permanent Fault	Temporary Fault
Physical Faults										
Transient Faults										
Intermittent Faults										
Design Faults										
Interaction Faults										
Malicious Logic										
Intrusion										

Errors

Whether or not an error actually leads to a failure depends on the following facts:

- the system **composition** and the existence of **redundancy** (intentional or unintentional redundancy)
- the system **activity** after the introduction of an error (the error may get overwritten)
- the definition of a failure by the **user's viewpoint**

The means for dependability

- **fault prevention:**
  Methods and techniques aimed at preventing the introduction or occurrence of faults.
- **fault removal:**
  Methods and techniques aimed at reducing the number and seriousness of faults.
- **fault forecasting:**
  Methods and techniques aimed at evaluating and modelling the present number, future incidents, and severity of faults.
- **fault tolerance:**
  Methods and techniques aimed at providing a service that is consistent with its specification in spite of faults.

$\Rightarrow$**fault avoidance:**
fault prevention and fault removal

Fault prevention

- **hardware components:**
  - environment modifications (temperature)
  - quality changes, use “better” components
  - component integration level, higher integration
  - derating, reduction of electrical, thermal, mechanical, and other environmental stresses
- **software components:**
  - software engineering methodologies
  - OOD and OO languages
  - design rules
  - CASE tools
  - formal methods

Fault removal

- **verification:**
  to check, whether the system adheres to the specification.
  - Static analysis: inspections, walk-throughs, data flow analysis, complexity analysis, compiler checks, correctness proofs, petri net models, finite state automata.
  - Dynamic Analysis*:* testing, black-box, white-box, conformance, fault-finding, functional, timeliness, structural, deterministic, random or statistical
- **diagnosis:**
  diagnosing the fault which prevented the verification from succeeding
- **correction:**
  perform corrective actions to remove the fault $\Rightarrow$ regression verification

Fault forecasting

- performing an evaluation of the system with respect to faults
- evaluation of aspects such as:
  - reliability
  - availability
  - maintainability
  - safety
- see chapter “Fault-tolerance and modelling”

Fault tolerance

There are four phases, which, taken together, provide the general means by which faults are prevent from leading to system failures.

- **error detection:**
  errors are the manifestations of faults, which need to be detected to act upon
- **damage confinement and assessment:**
  before any attempt is made to deal with the detected error, it is necessary to assess and confine the extent of system state damage
- **error recovery:**
  error recovery is used to transform the currently erroneous system state into a well defined error-free system state
- **fault treatment and continued service:**
  even if the error-free system state has been recovered it is often necessary to perform further actions to prevent the fault from being activated again

Error recovery

two possibilities to transform the currently erroneous system state into an error-free system state:

- **Backward recovery:**
  - system state is reset to a previously store error-free system state
  - reexecution of failed processing sequence
  - typical for data base systems
    (it is not possible to predict valid system states)
- **Forward recovery:**
  - system state is set to a new error-free system state
  - typical for real-time systems with period processing patterns
    (it *is* possible to predict valid system states)

Fault-tolerance and redundancy

A system requires some kind of **redundancy** to tolerate faults. This redundancy can be implemented in three different domains:

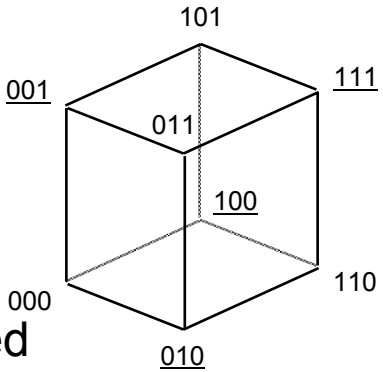
- **Domain of information:**
  redundant information e.g. error correcting codes, robust data structures
- **Domain of space:**
  replication of components, e.g. 2 CPU's, UPS (uninterruptable power supply)
- **Domain of time:**
  replication of computations, e.g. calculate results by same (or different) algorithm a second time, sending messages more than once

Fault-tolerance in the domain of information

- **error correcting codes:**
  - for all error correcting codes (ECC)

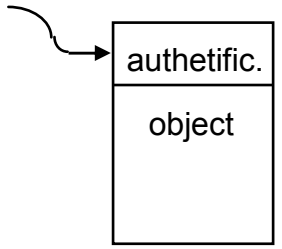
    $(2t + p + 1) \leq d$

    d .. Hamming distance of code
    t .. number of single bit errors to be tolerated
    p .. number of additional errors that can be detected



3 bit code, d = 1

- **robust data structures:**
  - store the number of elements
  - redundant pointers
    (e.g. double linked chains with status)
  - status or type information
    (e.g. authenticated objects)
  - checksum or CRC

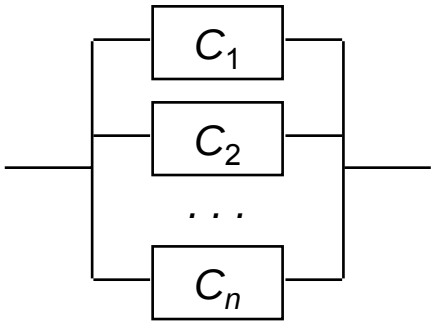


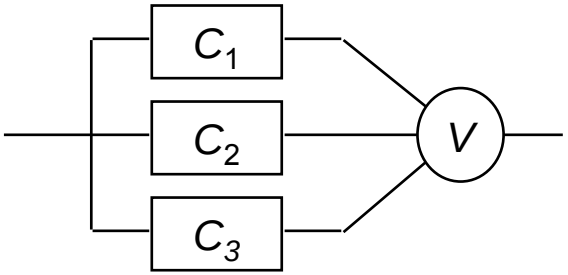
pointer to authentificated object

- **application specific knowledge**

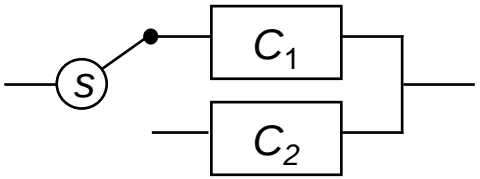
Fault-tolerance in the domain of space

- **active redundancy**
  - – parallel fail-silent components
  - – voting, triple modular redundancy (TMR)





- **passive or standby redundancy**
  - – hot standby: standby component is operating
  - – cold standby: standby components starts only in case of a failure



Fault-tolerance in the domain of time

allows tolerance of temporary faults

- **multiple calculation:**
  - a function is calculated n times with the same inputs
  - the result is checked by an acceptance test
  - or the multiple results are voted
- **sending messages multiple times:**
  - message transmission is repeated n times
  - retransmission only in case of failures (positive acknowledge retransmit PAR)
  - retransmission always n times (reduces temporal uncertainty for real-time systems)

Summary

- **Dependability** is defined as the trustworthiness of a computer system such that reliance can justifiably be placed on the services it delivers.
- **Reliability** is dependability with respect to continuity of service
- **Availability** is dependability with respect to readiness for usage
- **Safety** is dependability with respect to avoidance of catastrophic consequences
- **Security** is dependability with respect to prevention of unauthorized access and/or handling of information and/or availability
- **Specifications** are important, they are required to be **exact**, **consistent**, **complete**, and **authoritative**
- **Multiple levels** of specifications

Summary (cont.)

- A system **fails** if the delivered service deviates from the service specification.
- An **error** is that part of the system state which is liable to lead to a failure.
- A **fault** is the adjudged or hypothesized cause of an error.
- The **fault/failure chain**:
  $\rightarrow$ fault $\rightarrow$ error $\rightarrow$ failure $\rightarrow$ fault $\rightarrow$ error $\rightarrow$ failure
- Fault **classification** according to nature, origin and persistency
- To build a dependable computer system it is important to have a consistent model and acceptable terminology to reason about the mechanisms and possibilities how the system can fail.

Summary (cont.)

- the **means for dependability**:
  - fault prevention
  - fault removal
  - fault forecasting
  - fault tolerance
- the **4 phases of fault-tolerance**:
  - error detection
  - damage confinement and assessment
  - error recovery
  - fault treatment and continued service
- fault-tolerance requires **redundancy**
- the **domains of redundancy** are:
  - information
  - space (active or passive replication)
  - time