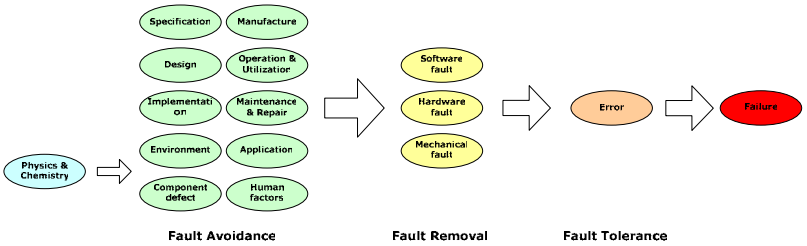
Concepts for Dependable and Secure Computing

References

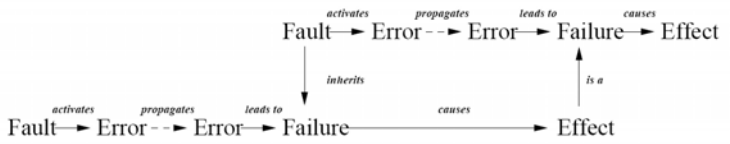
- Algirdas Avizienis, Fellow, IEEE, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. *Basic Concepts and Taxonomy of Dependable and Secure Computing*, in IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, VOL. 1, NO. 1, JANUARY-MARCH 2004

Fault, Error and Failures



3

Threads to Dependability



- Fault.
  - The cause of a failure is a fault that ranges from specification and design defects to physical or human factors.
- Error.
  - An error is a design flaw or a deviation from the desired or intended state of a system.
- Failure.
  - A failure is defined as the manner in which a component, subsystem, or system could potentially fail to meet or deliver the intended function.
- Effect.
  - The effect is the actual consequence of a system behavior in the presence of a failure.

4

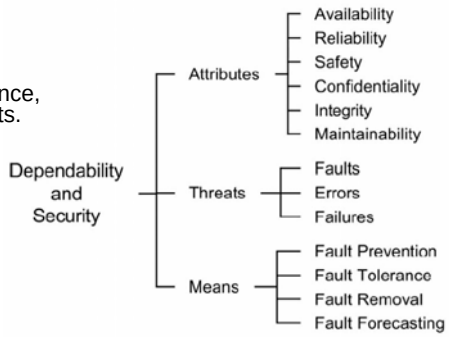
Dependability Attributes

- Availability
  - readiness for correct service.
    mean time to failure /
    mean time to failure + mean time to repair
- Reliability
  - continuity of correct service.
    mean time to failure
- Safety
  - absence of catastrophic consequences on the user(s) and the environment.
- Integrity
  - absence of improper system alterations.
- Maintainability
  - ability to undergo modifications and repairs.

5

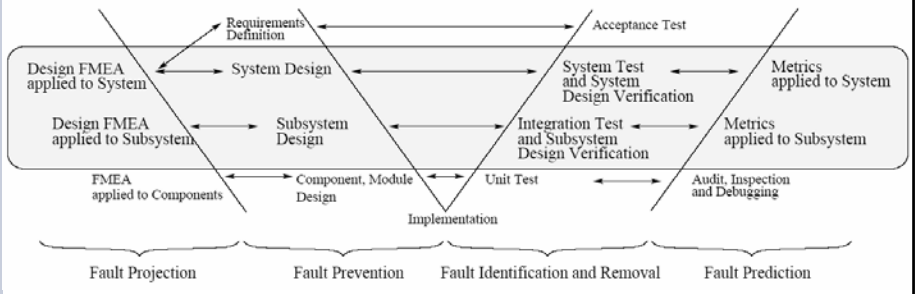
Means to attain Dependability

- Fault projecting
  - projecting failure modes in software development is used to establish a fault hypothesis and estimate the presence of faults, the future incidence, and the likely consequences of faults.
- Fault prevention / avoidance
  - means to prevent the occurrence or introduction of faults.
- Fault removal
  - means to reduce the number and severity of faults.
- Fault forecasting / prediction
  - tries to identify complex structures that are likely to become a source of faults.
- Fault tolerance
  - means to avoid service failures in the presence of faults.



6

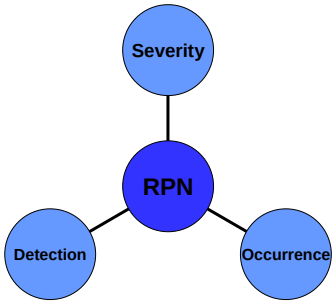
Analysis and Testing Lifecycle



7

Fault projection

- Anticipate potential scenarios of failure as soon as possible
- Focus on high-risk components
  - Identification of Failure Modes, Effects, Causes, Design Controls
  - Assessment with Risk Priority Number (RPN)



Effects → **Severity**

Causes → **Occurrence**

Error → **Detection**

8

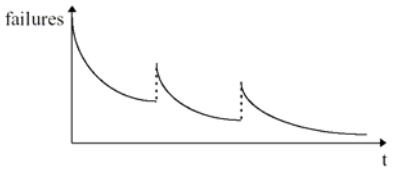
Fault avoidance

- Fault avoidance
  - use of formal methods, semi-formal methods, structured methods and object-oriented methods.
- Impose discipline and restrictions on the designers of a system.
  - Hinder the designers from making too complex designs and
  - provide means to model and predict the behavior of their designs.

9

Fault Removal

- Software does not wear out over time. It is therefore reasonable to assume that as long as errors are uncovered reliability increases for each error that is eliminated.
- *The failure rate of software decreases when errors are removed.*
  However, new errors are introduced when the software is changed.



10

Fault Tolerance: Phases

1. error detection
   - presence of fault is detected
2. damage confinement
   - damage due to a failure must be delimited
3. error recovery
   - correction of error
4. fault treatment and continued service
   - fault or fault component has to be identified and remove the component or use it differently

11

Error detection: Timing check

- Timing checks typically set a timer with a value presenting the timing constraints of the component.
- If the timer times out, it means that the timing constraints are violated.
- a timing violation often implies that the component behaves incorrectly

12

Error detection: Structural and Coding checks

- **Semantic checks**
  - ensure that the value is consistent with the rest of the system
- **Structural check**
  - internal data structure is as it should be
- **Coding checks**
  - E.g. with checksums

13

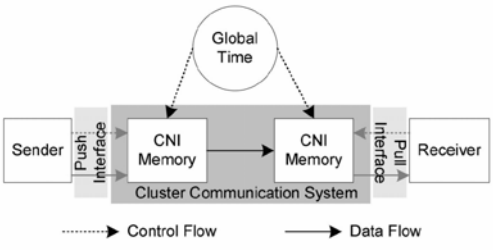
Damage confinement

- **Error detected**
  - Prevent error from propagating through the system
- **Firewalls**
  - Design firewalls into the system to ensure that no information flow takes place across the walls.

14

Communication Network Interface (CNI)

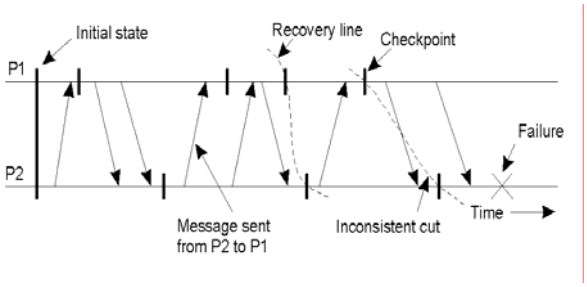
- A receiver that is working on a time-critical task is never interrupted by a control signal from the communication system.
- Since no control signals cross the CNI, propagation of control errors is prohibited by design.



15

Error recovery

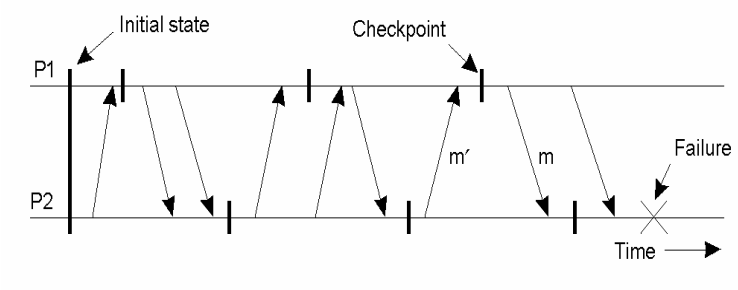
- **Backward recovery**
  - system state is restored to an earlier state, hoping that the earlier state is error-free.



16

Error recovery

- **Independent checkpointing**
  - Domino effect



17

Error recovery

- **Forward recovery**
  - no previous state is available. Instead the system attempts to go forward trying to make the system error-free by taking corrective actions

18

Fault tolerance design principles

- **Fault tolerant designs can be divided into two categories**
  - robust designs
    - Robust systems are designed to cope with unexpected inputs, changed environmental conditions and errors in the model of the external system. A robust design can for example, be a servo feedback loop in a control system.
  - redundant designs

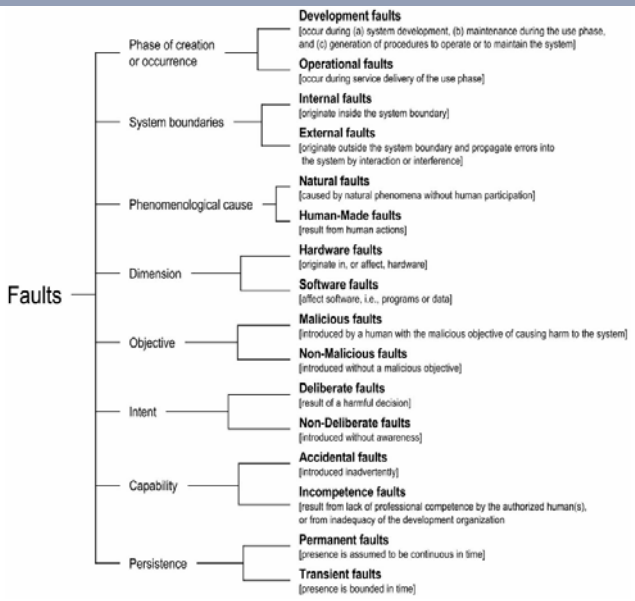
19

Redundant design

- **Information Redundancy:**
  - For example, checksums or double-linked lists are/make use of redundant *information*. Data structures that make use of redundant information are usually referred to as robust data structures. If, for example, a double linked list is used – and one link is corrupted, the list can be regenerated using the other link.
- **Time Redundancy**
  - Redundancy in *time* can be realized for example, by allowing a function to execute again if a previous execution failed.
- **Physical Redundancy**
  - *Redundancy* in space is called *replication*. The concept is founded on the assumption that parts that are replicated fail independently. A common use of replication is for example, to use several sensors, networks or computers in parallel.
- **Model Redundancy**
  - *Model-based* redundancy uses properties of a known model, e.g., physical laws. If for example, a revolution counter for a wheel, in a four wheel drive vehicle fails, it is possible to estimate the revolution speed based on the other wheels' speeds.

20

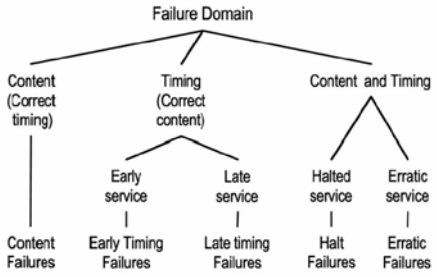
Faults



21

Failures

- Content failures
  - The content of the information delivered at the service interface (i.e., the service content) deviates from implementing the system function.
- Timing failures
  - The time of arrival or the duration of the information delivered at the service interface (i.e., the timing of service delivery) deviates from implementing the system function.



- Halt failure
  - or simply halt when the service is halted (the external state becomes constant, i.e., system activity, if there is any, is no longer perceptible to the users)
- Erratic failures
  - i.e., when a service is delivered (not halted), but is erratic (e.g., babbling).

22

Failure Consistency

- consistent failures
  - The incorrect service is perceived identically by all system users.
- inconsistent failures
  - Some or all system users perceive differently incorrect service (some users may actually perceive correct service); inconsistent failures are usually called, after, Byzantine failures.

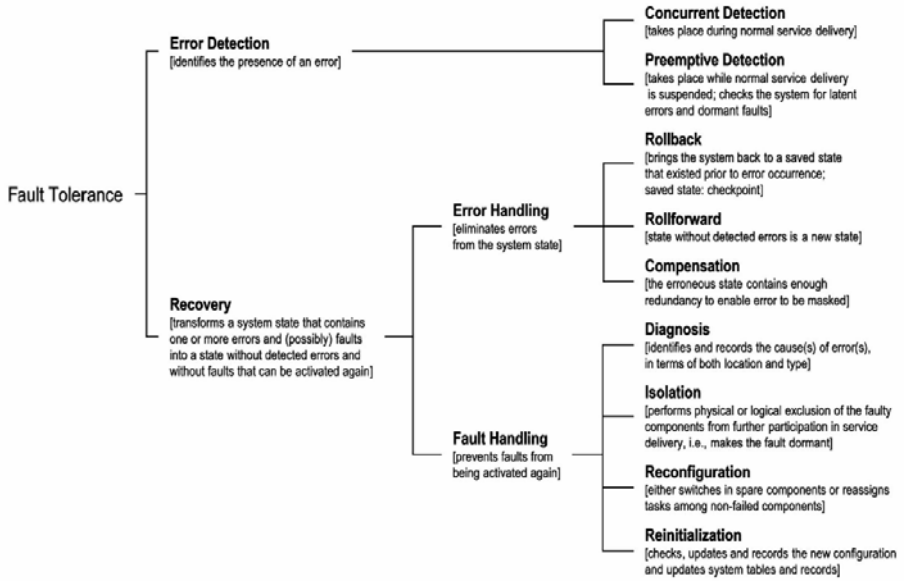
23

Development Failure

- Budget failure.
  - The allocated funds are exhausted before the system passes acceptance testing.
- Schedule failure
  - The projected delivery schedule slips to a point in the future where the system would be technologically obsolete or functionally inadequate for the user's needs.
- Partial Development Failures
  - Overruns
    - Budget or schedule overruns occur when the development is completed, but the funds or time needed to complete the effort exceed the original estimates.
  - Downgrading
    - The developed system is delivered with less functionality, lower performance, or is predicted to have lower dependability or security than was required in the original system specification.

24

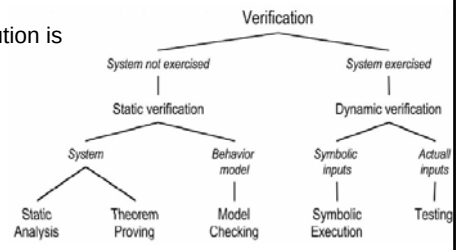
Means to attain Dependability



25

Fault Removal and Verification

- Verifying a system without actual execution is static verification.
  Such verification can be conducted:
  - static analysis
    - (e.g., inspections or walk-through, data flow analysis, complexity analysis, abstract interpretation, compiler checks, vulnerability search, etc.)
  - theorem proving;
    - on a model of the system behavior, where the model is usually a state-transition model (Petri nets, finite or infinite state automata), leading to model checking.

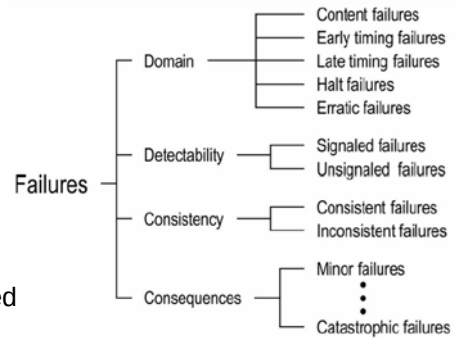


- Verifying a system through exercising it constitutes dynamic verification
  - Testing
    - Idea: Assumption that there exist a piecewise continuous relationships between the input and the output of a system. Only a few tests, for each continuous piece, needs to be performed. The behavior of the system intermediate to the samples can be interpolated.

26

Failure severity

- for availability
  - the outage duration
- for safety
  - the possibility of human lives being endangered;
- for confidentiality
  - the type of information that may be unduly disclosed
- for integrity
  - the extent of the corruption of data and the ability to recover from these corruptions.



27