Dependable Systems

Definitions and Metrics

Dr. Peter Tröger

Sources:

J.C. Laprie. Dependability: Basic Concepts and Terminology
Dependability

- **Umbrella term** for operational requirements on a system

  - IFIP WG 10.4: "[..] the trustworthiness of a computing system which allows reliance to be justifiably placed on the service it delivers [..]"
  
  - IEC IEV: "dependability (is) the collective term used to describe the availability performance and its influencing factors: reliability performance, maintainability performance and maintenance support performance"
  
  - Laprie: „Trustworthiness of a computer system such that reliance can be placed on the service it delivers to the user “

- Adds a third dimension to system quality

- General question: How to deal with unexpected events?

- In German: ’Verlässlichkeit‘ vs. ’Zuverlässigkeit‘
System Type Examples

- **Dependable (reliable) system**
  - Delivers a required service during its lifetime

- **Fault-tolerant computer system**
  - Continues correct service provisioning in the presence of faults

- **Real-time computer system**
  - Deliver a service within given time constraints (physical time, duration, ...)

- **Responsive computer system**
  - Fault-tolerant real-time system
System Integration Levels

- Dependability has to be considered at every level
- Decomposition approach influences dependability success
Dependability Stakeholders

- **System** - Entity with function, behavior, and structure
  - A number of components or subsystems, which interact under the control of a design [Robinson]

- **Service** - System behavior abstraction, as perceived by the user

- **User** - Human or physical system that interacts with the systems service

- **Specification** - Definition of expected service and delivery conditions
  - On different levels, can lead to specification fault

- Reliance demands assessment of **non-functional dependability attributes**

- Provide ability for trustworthy service delivery by **dependability means**

- Undesired (maybe expected) circumstances form **dependability threats**
Dependability Tree (Laprie)
Dependability Threats

• **System failure** - *'Ausfall'*
  • Event that occurs when the service no longer complies with the specification / deviates from the correct service.

• **System error** - *'Fehler(zustand)'*
  • Part of system state that can lead to subsequent failure
  • Some sources define errors as active faults - not in this course ...

• **System fault** - *'Fehler(ursache)'*
  • Adjudged or hypothesized cause of an error
  • Failure occurs when error state alters the provided service
  • Systems are build from connected components, which are again systems
  • Fault is the consequence of a failure of some other system to deliver its service
Chain of Dependability Threats (Avizienis)

Fault → Error → Failure → Propagation → Activation → Fault → Causation → Fault
Faults

• High diversity in possible sources and types

• Fault nature
  
  • Accidental faults (‘Zufallsfehler’) vs. intentional faults (‘Absichtsfehler’)

  • Intentional faults are created deliberately, presumably malevolently

• Fault origin viewpoints (not exclusive)
  
  • Phenomenological causes: Physical / natural faults vs. human-made faults

  • System boundaries: Internal faults (part of system state that produces an error) vs. external faults (interference with the environment)

  • Phase of creation: Design faults vs. operational faults

• Temporal persistence
  
  • Permanent faults vs. temporary faults
Observations on Faults

- An external fault is a design fault - inability or refusal to foresee all situations
- Design faults are created during system development, system modification, or operational procedure creation and establishment
- Just replacing broken version of the same component leads to recurrent faults
- Physical faults are accidental faults
- Temporary external accidental physical faults are also called transient faults
- Temporary internal accidental faults are also called intermittent faults
  - Examples: Pattern-sensitive memory hardware, system overload
  - Arbitrary concept - Permanent faults with unknown activation condition
- Intentional and design faults are human-made faults, might be malicious faults
- Hardware production defects are typically physical faults
Observations on Faults

• A fault is **active** when it produces an error

• A non-active internal fault is a **dormant / passive fault** („inaktive Fehlerursache“)
  • Origin in hardware fault analysis - often cycling between dormant and active

• Many specialized versions of the term ‚fault‘, e.g. **bug**
  • **Heisenbug** - Intermittent software fault, **Bohrbug** - Permanent software fault
  • **Mandelbugs** - Appear chaotic due to many dependencies

• **Fault-tolerant system design** is a contradiction
  • Design demands specification, faults are non-specified cases
  • Solution: Specification for fault-free case + additional fault specification

• Fault can mean performance or timing faults (derivation from expected load / timing)
Fault Characterization (Laprie & Kanoun)

Diagram showing a hierarchical classification of faults based on nature, phenomenological cause, system boundaries, phase of creation, and persistence. The diagram includes categories such as accidental, intentional, physical, human, internal, external, design, operation, permanent, and temporary faults.
Fault Characterization (Laprie & Kanoun)
Fault Characterization (Laprie & Kanoun)
Fault Characterization (Laprie & Kanoun)
Fault Characterization (Laprie & Kanoun)

Diagram showing the classification of faults based on nature, phenomenological cause, system boundaries, phase of creation, and persistence.
Fault Characterization (Laprie & Kanoun)
Fault Characterization (Laprie & Kanoun)
Fault Characterization (Laprie & Kanoun)
Fault Model

- Faults can be classified into different categories on different abstraction levels
  - Physics
  - Circuit level / switching circuit level
    - Interesting for hardware design research (not this course)
    - Investigate logical signals on connections
      - stuck-at-zero, stuck-at-one, bridging faults, stuck-open
  - Register transfer level
  - Processor-memory-switch (PMS) level
  - Hardware system level
  - ... (Software) ...
Physical Faults [Goloubeva]

• Highly energized particles originate from space, atmospheric, or ground radiation
  • Cosmic radiation, solar heavy ions, solar protons, ...

• Interaction of particle that strikes a circuit - atomic displacement, direct ionization, indirect ionization created by nuclear reactions

• Smaller structures are sensitive to ionization effects from all kinds of particles

• **Single Event Upset (SEU)** - injected charge changes content of a memory bit

• Dynamic random access memory (DRAM) - typical building blocks for main memory
  • No inherent refreshing, influence on storage capacitor changes value

• Static random access memory (SRAM), for caches, registers, pipeline, ...
  • Impact on restoring transistor leads to invalid refresh operation
Physical Faults [Goloubeva]

- Logic circuits: Shrinking size, reduction of power supply, increase of frequency
  - Noise margin is extremely reduced, single-event strike impacts circuit lines
- **Single Event Transient (SET):** Particles modify voltage in a combinational circuit
  - Can be modeled at gate level as erroneous transition on the gate output

![Logic Circuit Diagram]
Fault Model for Semiconductor Memories

- **Stuck-at-1** or **stuck-at-0** (hard) faults, **transition / bit-flip faults** \((0\rightarrow1, 1\rightarrow0)\)
- **Open and short circuits** - Too much or too little metallization; Also open bonds
- **Input and output leakage** - Leakage current in excess of the specified limit
- **Multiple writing** - Data written into more than one cell when writing into one cell
- **Pattern sensitivity** - Device does not perform reliably with certain test pattern(s)
- **Refresh dysfunction** - Data are lost during the specified minimum refresh time
- **Write recovery** - Write followed by reading/writing at different location resulting in reading/writing at same location
- **Sense amplifier recovery** - Data accessed for a number of cycles are the same and then suddenly changed, sense amplifier tends to stay in the same state
- **Sleeping sickness** - Memory loses information in less than the stated hold time (typically tens of milliseconds)
Fault Model for Semiconductor Memories

- **Decoder malfunction** - Inability to address same portions of the array
  - No cell accessed by certain address, multiple cells accessed by certain address
  - Certain cell not accessed by any address
  - Certain cell accessed by multiple addresses

- **Bridging fault** - Short between cells, AND type or OR type

- **State coupling fault** - Coupled (victim) cell is forced to 0 or 1 if coupling (aggressor) cell is in given state

- **Inversion coupling fault** - Transition in coupling cell inverts coupled cell

- **Idempotent coupling fault** - Coupled cell is forced to 0 or 1 if coupling cell transits from 0 to 1 or 1 to 0

- **Disturb fault** - Victim cell forced to 0 or 1 if we read or write aggressor cell (may be the same cell)
System-Level Fault Model

• Idea from hardware background, meanwhile also in software
  • Usage: How many faults of different classes can occur? What do I tolerate?
• Process as black box, only look on input and output messages
• Link faults are mapped to the participating components
• Timing of faults: Fault delay, repeat time, recovery time, reboot time, ...
• Every participating component would need a fault model - pick the most urgent ones
System-Level Fault Model [Cristian]

• **Fail-Stop** Fault: Processor stops all operations, notifies the other ones

• **Crash** Fault: Processor looses internal state or stops without notification

• **Omission** Fault: Processor will break a deadline or does not react to some task at all
  - Send / Receiver Omission Fault: Necessary message was not not sent / not received in time

• **Timing** Fault / **Performance** Fault: Processor stops / reacts to a task before its time window, after its time window, or never

• **Incorrect Computation** Fault: No correct output on correct input

• **Byzantine** Fault / **Arbitrary** Fault: Every possible fault
  - Authenticated Byzantine Fault: Every possible fault, but authenticated messages cannot be tampered
Errors

- State of the system, not an event!
- Escalates to failure depending on
  - Intentional / unintentional redundancy
  - System activity
  - User’s definition of a failure
    - Examples: Maximum outage time, acceptable delay, retransmission rate
- System activity can overwrite the error state before damage is happening
- **Latent** (not recognized) vs. **detected** error coming from an active fault
- Hardware often contains unintentional redundancy, makes it difficult to test
Hardware Error Models [Goloubeva]

• Hardware faults effect state information, e.g. register values
  • Stuck-at and other hardware faults therefore can also be denoted as error
• More interesting to investigate resulting effects on system-level
  • Single data error - Program data is corrupted (in cache, memory, or register)
  • Single code error - Effect on one instruction of the code
    • Type 1/2 - Instruction modification without / with change of control flow
• Nature of error state can confirm to the nature of the originating fault
  • Transient vs. permanent, static vs. dynamic, single vs. multiple
  • Influence from utilized dependability means
Hardware Error Models [Goloubeva]

• Mapping of hardware-level single bit-flip error to other layers

  • **Memory data segment, processor data cache**: System-level single data error

  • **Memory code segment, processor code cache**: System-level single code error of type 1 (modification of target register) or type 2 (modification of branch target)

  • **Memory stack segment**: System-level data error or type 2 code error

• **Processor register**: Depending on processor architecture and register type

  • Single data error if register holds data interpreted by the application

  • Single type 1 code error, if register holds address used by load/store operation

  • Single type 2 code error, if register holds address of a branch target

• **Processor control register**: Everything could happen ...
Hardware Error Models - Code Errors [Goloubeva]

MOV R0, 10
MOV R1, 1
LOOP: ADD R1, R1
SUB R0, 1
BNZ LOOP

MOV R0, 10
MOV R1, 1
LOOP: SUB R1, R1
SUB R0, 1
BNZ LOOP

MOV R0, 10
MOV R1, 1
LOOP: ADD R1, R1
SUB R0, 1
BNZ LOOP

MOV R0, 10
MOV R1, 1
LOOP: ADD R1, R1
SUB R0, 1
BNZ FOOBAR

MOV R0, 10
MOV R1, 1
LOOP: ADD R1, R1
SUB R0, 1
BNZ LOOP

MOV R0, 10
MOV R1, 1
LOOP: ADD R1, R1
SUB R0, 1
BZ LOOP
Software Error Models [Goloubeva]

• Similar terminology, but completely different semantics

• Syntactical errors are handled by compiler, semantical errors occur at runtime
  • Static vs. dynamic, permanent vs. temporary errors

• Example for C programming language
  • Errors affecting assignments (missing / wrong local variable values)
  • Errors affecting conditional instructions (wrong boolean or iteration condition)
  • Errors affecting function call / return (wrong parameters, return statement)
  • Errors affecting algorithms (missing statements or function calls, wrong operators)

• Under research in the software engineering field - field studies, automated code analysis, developer interviews
Error Propagation
Error Propagation [Goloubeva]
Error Message Occurrence (Hansen & Siewiorek)

- Same fault can lead to different (detected or undetected) errors
- Errors become detected by error detection mechanisms
  - Some undetected errors are detected by several detectors
  - Some detectors report several undetected errors together
  - Some undetected errors are not detected at all
- Detected errors might not be logged, if the system stops too fast
Failures

• Non-compliance with the specification - arbitrary failure (‘willkürlicher Ausfall’)

• System failures can be further categorized in failure modes
  
  • Fail-silent / crash failure mode - incorrect results are not delivered
  
  • Fail-stop mode - constant value is delivered

• Failure mode view points
  
  • Failure mode domain - what is influenced
    
    • Service result - value failures, service timeliness - timing failures
    
    • Service availability - stopping failures
  
  • User perception in this mode - consistent / inconsistent for all users
  
  • Failure consequences in this mode - allow ordering of failure modes
Failure Severity (‘Schweregrad des Ausfalls’)  

- Denotes consequences of failure  

- **Benign failures** (‘unkritische Ausfälle’)  
  - Failure costs and operational benefits are similar  
  - Sometimes also umbrella term for failures only detected by inspection  
  - A system with only such failures is **fail-safe**  

- **Catastrophic failures** (‘kritische Ausfälle’)  
  - Costs of failure consequences are much larger than service benefit  

- **Significant / serious failures** - Intermediate steps expressing reduced service  

- Grading of failure consequences on overall system depends on application  
  - Flying airplane - Catastrophic stopping failure, Train - Benign stopping failure  

- **Criticality** - Highest severity of possible failure modes in the system
Critcality Levels Example: DO-178B Standard

• Software Considerations in Airborne Systems and Equipment Certification
  • Mature document, developed for more than 20 years

• Definition of severity of failure conditions for airplane, crew, and passengers
  • Catastrophic - Loss of ability to continue safe flight and landing
  • Major - Reduced airplane or crew capability to cope with operating conditions
    • Reduction in safety margins and functional capabilities
    • Higher workload or physical distress for the crew
  • Minor - Not significantly reduced airplane safety, slight increase in workload
  • No effect - Failure results in no loss of operational capabilities and no increase in crew workload
Example: DO-178B Standard
Failure Types

• Duration of the failure
  • **Permanent** failures - no possibility for repairing or replacement
  • **Recoverable** failures - back in operation after a fault is recovered
  • **Transient** failures - short duration, no major recovery action

• Effect of the failure
  • **Functional** failures - system does not operate according to its specification
  • **Performance** failures - performance or SLA specifications not met

• Scope of the failure
  • **Partial** failure - only parts of the system become unavailable
  • **Total** failure - all services go down
Swiss Cheese Model (Prof. Reason)

- Origins in medical research
- Defenses, barriers, and safeguards might be penetrated by fault trajectory
Observations on Failures

• Failures vs. Load
  • Typically positive correlation
    • Increasing load can lead to wear-out - increasing failure rate
    • Higher load can show up failure causes
    • Detected faults lead to recovery activities - load increases
  • Feedback effects possible
  • Related faults (attributed to a common cause) can lead to common-mode failures
Chain of Dependability Threats

Sources of Errors

[from Siewiorek and Swarz]
Security - Vulnerability Assessment [Johnston]

• Different dependability attribute targets might lead to different terminology

• Example: Vulnerability assessment for nuclear security

  • **Threat**: Who might attack against what asset, using what resources, with what goal in mind, when / where / why, with what probability

  • **Threat assessment (TA)**: Attempting to predict the threats - proactive security

  • **Vulnerability**: Specific weakness in security that could be exploited

  • **Vulnerability assessment (VA)**: Attempting to discover / demonstrate them

  • **Risk management**: Deploy, modify, and re-assign security resources, based on TA results, VA results, assets, security breach consequences, and costs (time, money, human resources)

  • **Attack**: Attempt to harm valuable asset by exploiting one or more vulnerabilities
Security - Vulnerability Assessment [Johnston]

• Threats and vulnerabilities are different concepts, and must be treated separately
  • Vulnerabilities without threats are not interesting
  • Vulnerabilities do not define threats (bad locks do not imply thieves to show up)
• No one-to-one mapping, different attacks can exploit the same vulnerability
• TA involves mostly speculation about unknown people, so VA is more important
• Correct VA should identify large amount of issues with cheap countermeasures
• System features can become a vulnerability only in combination with an attack
• TA and VA are not pass / fail certifications
Means for Dependability

- **Fault prevention** - Prevent fault occurrence or introduction
- **Fault tolerance** - Provide service matching the specification under faults
- **Fault removal** - How to reduce the presence of faults
- **Fault forecasting** - Estimate the present number, future incidence, and the consequences of faults
- Combined utilization

![Diagram showing dependability attributes and means]
Dependability Means (Laprie)

- Offline / online techniques
  - Fault intolerance techniques
    - **Fault prevention** - Prevent fault occurrence or introduction
    - **Fault removal** - Reduce the presence of faults
    - 100% fault-free servicing for the whole life time is not possible
  - Fault tolerance techniques
    - **Fault forecasting** - Estimate the present number, future incidence, and the consequences of faults
    - **Fault tolerance** - Provide service complying with specification in spite of faults
- Problems with **coverage** and **validation of the validator**
Dependable System Design (Echtle)
Fault Prevention

• Specific approaches for avoiding faults
  • Specialized specification formalisms and techniques
  • Specialized development / manufacturing process to prevent design faults
  • Shielding
    • Only use ultra-reliable components

• General engineering approaches
  • Software engineering procedures
  • Quality management regulations and enforcement
  • Training and organization of maintenance departments
Fault Removal

• Make faults disappear before fault tolerance becomes relevant

• Step 1: Verification
  • Check if the system adheres to verification conditions; if not, take next steps
  • Static verification: Static analysis, data flow analysis, compiler checks
  • Dynamic verification: Symbolic execution or verification testing

• Step 2: Diagnosis
  • Find the faults that influenced the verification conditions

• Step 3: Correction
  • Fix the problem, repeat the steps (regression)

• Fault removal during operation: Corrective maintenance (curative / preventive)
Testing

• Selecting test inputs is driven from different view points
  
  • Testing purpose: **conformance testing**, **fault-finding testing**
  
  • System model: **functional testing** (with functional model) or **structural testing**
  
  • Fault model: enables **fault-based testing**
  
• Deterministic testing vs. random testing

• Structural testing of hardware is fault-finding, fault-based, structural testing

• Structural testing of software is fault-finding, non-fault-based, structural testing

• **Golden unit**: Reference system for comparison of output for a given input
Fault Tolerance

• Fault tolerance is the ability of a system to operate correctly in presence of faults.

or

• A system S is called **k-fault-tolerant** with respect to a set of algorithms \( \{A_1, A_2, \ldots, A_p\} \) and a set of faults \( \{F_1, F_2, \ldots, F_p\} \) if for every k-fault F in S, \( A_i \) is executable by a subsystem of system S with k faults. (Hayes, 9/76)

or

• Fault tolerance is the use of **redundancy** (time or space) to achieve the desired level of system dependability - costs!

• Accepts that an implemented system will not be fault-free

• Implements automatic recovery from errors

• Is a recursive concept (voter replication, self-checking checkers, stable memory)
Fault Tolerance

• Typical design methodology in many technical and biological systems
  • Spare wheel in cars, redundant organs, ...

• Fault tolerance mechanisms need to be evaluated by dependability attributes
  • Minimum, maximum, average reliability and availability
  • Easy to formulate and understand, hard to prove - failure rate remains unknown
  • Quantitative limits based on fault model (which faults in which components)

• Typically 'one-fault-at-a-time' assumption

• Different attributes of fault tolerance implementation to be checked
  • Functional verification, sensitivity analysis, minimum amount of resource resp. computational overhead, implementation performance, transparency, portability
Phases of Fault Tolerance (Hanmer)
Decomposition of Fault Tolerance (Lee & Anderson)

• Error detection
  • Presence of fault is deducted by detecting an error in some subsystem
  • Implies failure of the according component

• Damage confinement
  • Delimit damage caused due to the component failure

• Error processing - recovery / compensation
  • System recovers from the effect of an error

• Fault treatment
  • Ensure that fault does not cause again failures
Fault Tolerance - Error Detection

• Replication check
  • Output of replicated components is compared / voted
  • Independent failures, physical causes -> many replicas possible (e.g. HW)
  • Finds also design faults, if replicated components are from different vendors

• Timing checks (‘watchdog timers‘)
  • Timing violation often implies that component output is also incorrect
  • Typical solution for node failure detection in a distributed system

• Reasonableness checks - Run-time range checks, assertions
  • Structural and coding checks, diagnostics checks, algorithmic checks

• Ideal: Self-checking component with clear error confinement areas
Fault Tolerance - Error Detection

- Replication checks are powerful and expensive, examples:
  - Execute identical copies on different hardware (component failures)
  - Execute separate and different versions (assumes independent design faults)
  - Execute same copies different times (transient faults)
  - Replicate only portion of the system
  - Works for both hardware and software

- Signaling aspect in the error detection task
  - Typical software model are exceptions, a way for implementing forward recovery

- Combination fault detection and fault location
Fault Tolerance - Damage Confinement (Taylor)

- **System decomposition**
  - Every communication link might enable damage spreading
  - Introduce mutual suspicion
  - Hardware-based separation of software components
  - OS-based separation (processes, runtime monitors, special shells)

- **Law-governed architecture**
  - Externalize contrains on interaction by runtime rules

- **Strongly-typed language**
  - Language guarantees the absence of unintended control flows
Preventing Error Propagation

- Especially relevant when single components communicate their data
  - **Single-source information** - local clock, sensor data, transaction status ...
  - Non-failed component must find an **agreement** how to treat received information
    - Special topic in distributed systems
    - Atomic broadcast, clock synchronization, membership protocols
Fault Tolerance - Error Processing Through Recovery

- **Forward error recovery**
  - Error is masked to reach again a consistent state (*fault compensation*)
  - Corrective actions need detailed knowledge (*damage assessment*)
  - New state is typically computed in another way
    - Examples: error correcting codes, non-journaling file system check, advanced exception handlers, (voters)

- **Backward error recovery**
  - Roll back to previous consistent state (*recovery point / checkpoint*)
  - Very suitable for transient faults
  - Computation can be re-done with same components (*retry*), with alternate components (*reconfigure*), or can be ignored (*skip frame*)
Fault Tolerance - Fault Treatment

- **Fault diagnosis** - determine error cause's location and nature
- **Fault passivation** - (remove faulty component & reconfigure system
  - Error processing might already remove the fault - *soft fault*
  - Typical example are temporary faults
- Fault tolerance manager
  - Careful diagnosis with hardware support
  - Damage assessment by disabling faulty components automatically
  - Example: IBM mainframe architecture
- Software rejuvenation
  - Gracefully terminating an application and immediately restarting it at a clean internal state
Fault Tolerant Mindset (Hanmer)

• What can go wrong in any given situation?
  • Mindset to be applied in all development stages

• “Every problem in computer science boils down to tradeoffs“ [Henschen]
  • Fault prevention vs. fault tolerance vs. failure severity

• KISS principles, leave out „bells and whistles“

• Incremental additions of reliability - long-term products

• Defensive Programming
  • Simple error handling; fix root cause, not symptoms; make data auditble; make code maintainable;
Fault Tolerant Design Methodology (Hanmer)

- Assess things that can go wrong with the system (e.g. fault trees).
  - Find potential risks and according system failures.
- Define strategies to mitigate the identified risks.
  - Failure avoidance options, prevent faults from activation
- Create a mental model of the system design with redundancy.
- Design error detection and error processing capabilities.
- Design in the failure mitigation capabilities.
- Design human-computer interactions and modes of management.
Dependable Design Strategies (Malek)

- Decompose the system
  - Identify fault classes, fault latency and fault impact for the components
  - Identify “weak spots” and assess potential damage
  - Integrate partial recovery / reintegration / restart

- Determine qualitative and quantitative specs for fault tolerance and evaluate your design in specific environment

- Develop / utilize fault and error detection techniques and algorithms
- Develop / utilize fault isolation techniques and algorithms
- Refine fault tolerance, iterate for improvement
- Re-use proven components, but be aware of integration issues
Attributes of Dependability

• **Non-functional attributes** such as reliability and maintainability

• **Complementary nature of viewpoints** in the area of dependability

• In comparison to functional properties
  
  • ... hard to define
  
  • ... hard to abstract
  
  • ... ‘Divide and conquer‘ does not work as good
  
  • ... difficult interrelationships
  
  • ... often probabilistic dependencies
Attributes of Dependability

- **Reliability ("Funktionsfähigkeit")** - Continuity of service
  - Initial goal for computer system trustworthiness; other disciplines have different understanding
  - "Reliability is not doing the wrong thing." [Gray85]
  - "Reliability: Ability of a system or component to perform its required functions under stated conditions for a specified period of time" [IEEE]
  - "Reliability is the probability that an item will not fail." [Misra]

- **Availability ("Verfügbarkeit")** - Readiness for usage
  - "Probability that a system is able to deliver correctly its service at any given time." [Goloubeva]
  - "Maintainability is the probability that the item can be successfully restored to operation after failure; and availability ... is a function of reliability and maintainability." [Misra]
Observations on Dependability Attributes

• Availability is always required
• Reliability, safety, and security may be optional
• Reliability might be analyzed for hardware / software components
• Availability is always from the system view point
Attributes of Dependability

• **Safety** - Avoidance of catastrophic consequences on the environment
  • Critical applications
  • Specification needs to describe things that should not happen

• **Security** - Prevention of unauthorized access and/or information handling
  • Became especially relevant with distributed systems

• **Confidentiality** - Absence of unauthorized disclosure of information

• **Integrity** - Absence of improper system alteration
  • With respect to either accidental or intentional faults

• **Maintainability** - Ability to undergo modifications and repairs
In Detail

• **Reliability** - Function $R(t)$
  - Probability that a system is functioning properly and constantly over time period $t$
    - Assumes that system was fully operational at $t=0$
    - Denotes failure-free interval of operation

• **Availability** - Fraction of / points in time were a system is operational
  - Describe system behavior in presence of error treatment mechanisms (fault tolerance, repairing)

• **Instantaneous availability (at $t$)** - Probability that a system is performing correctly at time $t$; equal to reliability for non-repairable systems: $A_i(t) = R(t)$

• **Steady-state availability** - Probability that a system will be operational at any random point of time, expressed as the fraction of time a system is operational during its expected lifetime: $A_s(t) = \frac{Uptime}{Lifetime}$
Reliability Definition: PDF & CDF

- Probability density function \(pdf\) for random variable \(X\)
  - Discrete variable: Probability that \(X\) will be \(x\)
  - Continuous variable: Probability that \(X\) is in \([a, b]\)
    - Computed as area under the density function in this range
    \[
    P(a \leq X \leq b) = \int_{a}^{b} f(x) \, dx \quad \text{and} \quad f(x) \geq 0 \text{ for all } x
    \]

- Cumulative distribution function \(cdf(x)\): Probability that the value of \(X\) is at most \(x\)
  \[
  F(x) = P(X \leq x) = \int_{0, -\infty}^{x} f(s) \, ds
  \]
  - Limits of integration depend on the nature of the distribution function

- Value of \(cdf\) at \(x\) is always area under \(pdf\) up to \(x\)
PDF Examples

- Well-known statistical distributions, each describing a random variable behavior
- Parameters of the distribution derived from data, complete description then by \( pdf \)

Normal distribution (mean, variance)

Exponential distribution (rate parameter)

Probability density function

Cumulative distribution function
The Reliability Function $R(t)$

- Reliability: Probability $R(t)$ that a component works for time period $[0,t]$

- Idea: Express time period of correct operation as **continuos random variable** $X$  
  -> **time to failure**

  - $\text{cdf}(t)$ **of this variable**: Describes probability of failure before $t$ -> **Unreliability Function** $F(t)$

  - $1-\text{cdf}(t)$: Describes probability of a failure after $t$ -> **time to failure** -> **Reliability Function** $R(t)$

    - This works because working / non-working is a binary decision

- Typically, failures are modeled as Poisson process

  - Poisson properties lead to exponential distribution for the time between events

  - This time therefore only depends on failure rate parameter
Failure Rate

- Treat pdf for time-to-failure random variable X as **failure density function**
  - Can be computed as derivative of the unreliability function

\[ f(t) = \frac{dF(t)}{dt} \]

- **Failure rate** / hazard rate function - mean frequency of failures at time t
  - Conditional probability of a failure between a and b, given the survival until t

\[ \lambda(t) = \frac{f(t)}{R(t)} = \lambda \text{ for constant failure rate} \]
Why Exponential?

- Distribution function that models (beside others) the **memoryless property** of the Poisson process

  - $P(T > t + s|T > t) = P(T > s)$, e.g. $P_{\text{Failure}}(5 \text{ years}|T > 2 \text{ years}) = P_{\text{Failure}}(3 \text{ years})$
  
  - Failure is not the result of wear-out
  
  - Models „intrinsic failure“ part of the bath-tube curve, were most components spend the majority of their life time

  - Weibull distribution can also model tear-in and wear-out

- Some natural phenomena have constant failure rate (e.g. cosmic ray particles)

- Example: Product support determines an outage rate of 0.5% per day, independent from age

  - Failure rate is 0.005, so mean time to failure = 200 days

  - Numerical formulation for „law of small numbers“
The Reliability Function $R(t)$

- Failures occur continuously and independently at a constant average rate (Poisson process).
- Increasing probability of failure with increasing $t$ - cdt function.
- Failure rate $\lambda$ from experience or complexity measures.
- Cumulative distribution function:
  \[ F(x; \lambda) = \begin{cases} 1 - e^{-\lambda x}, & x \geq 0, \\ 0, & x < 0. \end{cases} \]
- Reliability function (survival probability) for exponential failure distribution:
  \[ R(t) = P(X > t) = 1 - F(t) = e^{-\lambda x} \text{ with } F(x) = 1 - e^{-\lambda x} \]
Variable Failure Rate in Real World

- Failure rate is treated as constant parameter of the exponential distribution

- (maybe invalid) simplification, mostly combined solution in practice:
  
  - Exponential distribution when failure rate is constant
  
  - Weibull distribution when failure rate is monotonic decreasing / increasing
Hardware Failure Rate

![Graph showing the failure rate over time with different phases: Early Life, Useful Life, Wearout. Key points include λ_e(t), λ_u(t), and λ_w(t).](chart.png)
Software Failure Rate

- Industrial practice

- When do you stop testing?  - No more time, or no more money ...

(C) Malek
Failure Rate Examples

- Standards from experience provide base data for component reliability
- Society of Automotive Engineers (SAE) reliability model

\[ \lambda_p = \lambda_b \prod_{i=1}^{b} \pi_i \]

- Predicted failure rate \( \lambda_p \)
- Base failure rate for the component \( \lambda_b \)
- Various modification factors \( \pi_i \)
  
  - Component composition
  - Ambient temperature
  - Location in the vehicle
Example: Item-Level Sparing Analysis [Misra]

- Sparing analysis challenges
  - How many spares do you need to keep the system available at the desired rate?
  - When are you going to need to spares (manufacturing time)?
  - Where the spares should be kept?
  - What system level you want to spare at?
Steady-State Availability

- **Mean time to failure (MTTF)** - Average time it takes to fail
- **Mean time to recover / repair (MTTR)** - Average time it takes to recover
- **Mean time between failures (MTBF)** - Average time between two successive failures

![Diagram showing MTTF, MTBF, and MTTR concepts](image-url)
Steady-State Availability and MTBF

- Expressing reliability with MTBF 'should' imply a repairable system
  - If all failures can be repaired, the MTBF estimate can become constant as time tends to infinity
  - In reliable systems, the downtime is short in comparison to uptime, so the steady-state condition holds earlier

- \( MTBF = MTTF + MTTR \)

- **Availability** = \( MTTF \) (accumulated up time) / \( MTBF \) (accumulated life time) = \( \frac{MTTF}{MTTF + MTTR} \)

- Expressing reliability with MTTF 'should' imply a non-repairable system
  - **MTBF (mean time BEFORE failure)** = MTTF -> typical source of confusion
  - If time to failure is exponentially distributed, then the reciprocal of the rate parameter is equivalent to the distribution mean

\[
\lambda = \frac{1}{MTTF}
\]
Example

- Test population with 50 HDDs and 100 hours of testing, 2 drives fail during the test
  - As usual, we assume exponential distribution of the time to failure
  - Reliability at t=100 is known to be 98%
  - Reciprocal of the according failure rate is the MTTF

\[ R(t) = P(X > t) = 1 - F(t) = e^{-\lambda x} \text{ with } F(x) = 1 - e^{-\lambda x} \]

\[ R(100\text{hours}) = e^{-\lambda^{100}} = 0.98 \]

\[ \lambda = -\frac{ln0.98}{100} = 0.000202 \]

\[ MTTF = \frac{1}{\lambda} = 4949.831\text{hours} \]
MTBF / MTTF in Practice

• Often express average failure behavior (statistics) for a component population

• Good for relative comparison, not for expected life time expectation of one unit

• Example: Hard disk with MTTF of 500,000 hours and 5 years of expected operation ('service life')
  
  • Drive of this type is expected to run 5 years without problems
  
  • Large group of such drives will (on average) have one failed drive after 500,000 hours of **accumulated** life time

• What to buy: Model with longer MTBF or longer warranty time?
Operational Availability Calculation [Misra]

- **Uptime** elements: Standby time, operating time

- **Downtime** elements
  - *Logistic*: Spares availability, spares location, transportation of spares
  - *Preventive maintenance*: Inspection, servicing
  - *Administrative delay*
    - Finding personnel, reviewing manuals, complying with supply procedures, locating tools, setting up test equipment
  - *Corrective maintenance*
    - Preparation time, fault location diagnosis, getting parts, correcting faults, testing
MTTR Examples

- Hardware MTTR with spares onsite
  - Operator available - 30min
  - Operator on call - 2 hours
  - Operator available during working hours - 14h
  - Without spares - at least 24h

- SW MTTR with watchdog
  - Reboot from ROM - 30s
  - Reboot from disk - 3 min
  - Reboot from network - 10 min
Steady-State Availability

\[ A = \frac{Uptime}{Uptime + Downtime} = \frac{MTTF}{MTTF + MTTR} \]

<table>
<thead>
<tr>
<th>Availability</th>
<th>Downtime per year</th>
<th>Downtime per week</th>
</tr>
</thead>
<tbody>
<tr>
<td>90.0 % (1 nine)</td>
<td>36.5 days</td>
<td>16.8 hours</td>
</tr>
<tr>
<td>99.0 % (2 nines)</td>
<td>3.65 days</td>
<td>1.68 hours</td>
</tr>
<tr>
<td>99.9 % (3 nines)</td>
<td>8.76 hours</td>
<td>10.1 min</td>
</tr>
<tr>
<td>99.99 % (4 nines)</td>
<td>52.6 min</td>
<td>1.01 min</td>
</tr>
<tr>
<td>99.999 % (5 nines)</td>
<td>5.26 min</td>
<td>6.05 s</td>
</tr>
<tr>
<td>99.9999 % (6 nines)</td>
<td>31.5 s</td>
<td>0.605 s</td>
</tr>
<tr>
<td>99.99999 % (7 nines)</td>
<td>0.3 s</td>
<td>6 ms</td>
</tr>
</tbody>
</table>
MTTR >> MTTF [Fox]

• Armando Fox on 'Recovery-Oriented Computing‘

  • \[ A = \frac{MTTF}{MTTF + MTTR} \]

  • 10x decrease of MTTR as good as 10x increase of MTTF ?

  • MTTF’s are not claimable, but MTTR claims are verifiable

  • Proving MTTF numbers demands system-years of observation and experience

  • Lowering MTTR directly improves user experience of one specific outage, since MTTF is typically longer than one user session

• HCI factor of failed system

  • Miller, 1968: >1sec “sluggish”, >10sec “distracted” (user moves away)

  • 2001 Web user study: ~5sec „acceptable”, ~10sec „excessively slow“
MTTR >> MTTF [Fox]

- Proposal: Utility curve for recovery time
  - Factors: Length of recovery time, level of service availability during error state
  - Key distinction between interactive (session-based) and non-interactive systems
- If error state leads to some steady-state latency
  - For how long will users tolerate temporary degradation?
  - How much degradation is acceptable?
  - Do they show a preference for increased latency vs. worse QOS vs. being turned away and incentivized to return?
- Long recovery times are often reasoned by stateful components
  - Utilize alternative architecture concepts
Availability

USER FRIENDLY by J.D. "Iliad" Frazer

HOW RELIABLE IS OUR NETWORK?

AS FAR AS OUR CUSTOMERS ARE CONCERNED, FIVE NINES.

WHAT DOES "FIVE NINES" MEAN?

99.999% UPTIME.

WAIT... WHY?!

SO WOULD "RELIABLE TO NINE FIVES" IN OUR NEWSPAPER AD BE NOT VERY GOOD?

I HOPE FOR YOUR SAKE THAT YOU GOT THE AD FIXED.

I DID I JUST GOT OFF THE PHONE WITH THE NEWSPAPER.

AND I TOLD THEM TO BE EXPLICIT ABOUT THE FIVE NINES.

EXPLICIT?

NOW WITH 9.9999% UPTIME!