Chapter 1

Basic terminology

1.1 The dependability concept

The trustworthiness of a computer system is characterized by its *dependability.* A system is dependable if "reliance can justifiably be placed the service it delivers" [Lap92]. Different attributes of the dependability are defined.

Availability is the readiness for usage. It is the measure of the delivery of correct service with respect to the alternation of correct and incorrect service.

Reliability is the continuity of service. It is measure of the continuous delivery of the correct service, or equivalently, the time to failure.

Safety is the non-occurrence of catastrophic consequences.

Security means the prevention of unauthorized access.

The reliability of a system is a time dependent function, namely the probability that the lifetime of the system exceeds a predefined time (assumed that at zero system time its was operational). Availability is the probability that the system is operational at the given point in time.

A system *failure* occurs when the service of the system does not comply with the specification, the behavior of the system deviates from that required by the user. The cause of a system failure is an erroneous transition of the system state, caused either the failure of a system component or an erroneous design. An *error* is that part of the system state which is liable to lead to failure. Whether or not an error leads to a failure depends on the system composition (applied redundancy) or on the activity of the system (an error may be overwritten). The adjudged or hypothesized cause of an error is a *fault.*

The development of a dependable computing system is supported by a set of methods which can be classed into:

Fault prevention: how to prevent fault occurrence or introduction

Fault removal: how to reduce the presence of faults

Fault tolerance: how to ensure a service capable of fulfilling the system's requirements in presence of faults

Fault forecasting: how to estimate the present number, future incidence and consequences of faults.

The notions of dependability are summarized in Table 1.1.

Dependability		
Impairment	Means	Attributes
faults	fault prevention	reliability
errors	fault tolerance	availability
failures	fault removal	safety
	fault forecasting	security

Table 1.1: Notions of dependability

1.2 On the impairments of dependability

The impairments of dependability (fault, error, failure) are of primary importance. They classification is summarized as follows:

A fault is *active* when it produces an error, otherwise *dormant.* Faults and their sources are diverse, a classification is possible on the nature, origin and persistence:

- The *nature* of a fault is accidental or intentional.
- The *origin* of a fault can be viewed from three viewpoints. *Phenomenologically* physical or human-made faults can be distinguished.
  Taking into account the *system boundaries*, internal faults are parts of the system state, external faults result from the interaction of the system with its physical environment (temperature, radiation) or human environment (operator).
  The *phase of creation* means design faults (during the development of the system) or operational faults (during the system's exploitation).
- On the basis of the *temporal persistence* of a fault *permanent faults* (presence is not related to internal or external pointwise conditions) and *temporary faults* (present for a limited amount of time) can be distinguished.

Temporary external faults originating from the physical environment are referred to as *transient* faults, while temporary internal faults are referred to as *intermittent* faults (e.g. marginal timing, load or pattern sensitive situations which occur rarely).

An error may be *latent* or *detected.* An error is latent if it has not been recognized, it is detected by an error detection mechanism or algorithm. An error may propagate and create more errors.

A failure occurs when an error passes through the system boundary and affects the service delivered by the system. Domain of a failure may be value or timing, consequences on the environment vary from benign to catastrophic. Perception of a failure by several users may be consistent or inconsistent (Byzantine).

The manifestation mechanism of faults, errors and failures is summarized by the chain

$$\text{fault} \rightsquigarrow \text{error} \rightsquigarrow \text{failure} \rightsquigarrow \text{fault} \rightsquigarrow \ldots$$

A subcomponent fault leads to a component failure, which can be interpreted as fault of the component which results in system failure etc. Failure modes of a system component can be viewed as fault types for the system.

1.3 On the attributes of dependability

The properties allowing the dependability are emphasized in different applications to a greater or less extent. Availability is always required, while safety and security may not be required according to the application.

Sometimes also other attributes of dependability are defined [Lyu96]:

Integrity is defined as the avoidance of improper alterations of informations. It is directly related to a specific class of faults, i.e. intentional faults, but relate to accidental faults as well. Besides integrity of data, integrity of programs is also important.

Confidentiality means the non-occurrence of unauthorized disclosure of information. Security is usually defined as the combination of integrity, confidentiality and availability.

Maintainability is the ability to undergo repairs and evolution. It goes beyond corrective maintenance which relates to repairability only. Adaptive maintenance means adjusting the system to environmental changes, perfective maintenance improves the function of the system by responding to customer demands.

Performability is a performance related measure of dependability. A system may automatically decrease its level of performance to compensate for hardware or software faults.

Testability is the ability to test for certain attributes within the system. It characterizes the ease with which certain tests can be performed.

1.4 On the means for dependability

Only the combination of the means (fault prevention, fault removal, fault tolerance and fault forecasting) can lead to a dependable computing system. Despite of fault prevention (proper design methodologies, construction rules) faults are created, this way fault removal is needed.

Fault removal itself is imperfect, just like the components of the system, hence the importance of fault forecasting. As our dependence on computing systems increases, fault tolerance becomes a key factor. It turns to construction rules and design methodologies again. The process is recursive since complex systems are designed and implemented using other (computerized) systems, which themselves have to be dependable.

Fault tolerance, i.e. how to prevent faults and errors leading to a failure, includes four phases as follows [LA90]:

Error detection finds the part of the system state which can lead to a failure. The basics are reviewed in a subsequent chapter.

Damage confinement and assessment finds out the consequences of a fault and assesses the extent to which the system state was damaged. An easy damage confinement requires structuring the interactions of the system, avoiding the uncontrolled spread of the erroneous state of a component to others. Mechanisms include atomic actions, list or ticket based protection mechanisms, locking systems [LA90].

Error recovery restores the error-free state state of the system. Anticipated damage of the system state can be recovered by *forward recovery*, unanticipated damage (not necessarily foreseen by the designer) can be handled in general by *backward recovery*. Sometimes the system state contains enough redundancy to be able to provide error-free service, this way no recovery is needed (error compensation).

Forward recovery produces a new, error-free state of the system depending on the damage assessment. It is a limited, system-specific solution, but very cost-effective.

Backward recovery restores a prior error-free state of the system. It is a more general concept, usually independent of the damage assessment. Provision is that recovery points have to be established (points in time for which the state needs to be restored). State based recovery means that the complete state of the system is stored at recovery points referred to as *checkpoints*. Operation based recovery logs the activities of the system in a so-called *audit trail*. In distributed systems, multiple recovery points are managed by recovery protocols which assure consistency and commitment during the establishment, discarding and restoring operations. Disadvantage of backward recovery is that the state restoration is time and resource consuming.

Fault treatment and continued service process identifies the fault; if it was a transient one then continues the system service otherwise repairs/reconfigures the system. What is damage assessment and error recovery for errors, that is fault identification and system repair for faults. Fault identification is performed by diagnostic checking, system repair and reconfiguration is usually manual or semi-automatic process. The normal service of the system is continued from a restart or retry point. If the system can continue the service in the case of some disabled components, *graceful degradation* is achieved.

1.5 Redundancy techniques

If fault tolerance or error detection is required then some form of redundancy is also required [Pra96]. Redundancy implies the addition of information, resources or time beyond what is needed for normal operation of the system. Its use can provide additional capabilities of the system, but has also an impact on areas of performance, size, weight, cost etc.

Redundancy can take forms of

- hardware,
- time,
- information and
- software.

Hardware redundancy

The physical replication of hardware is the most common form of redundancy. There are three basic types:

- **Passive redundancy** uses fault masking to hide the occurrence of a fault. It relies upon a voting mechanism (majority voting). The most common form is the modular redundancy technique.

  TMR (triple modular redundancy) triplicates the hardware and performs a majority vote to determine the output of the system. If one of the modules becomes faulty then the two remaining modules can mask the result of the faulty one. The replicated modules are typically processors, computing nodes or other (active) hardware components. The critical point in TMR is the voter, which is a *single point of failure*: if it fails then the complete system fails. Several techniques are known to overcome this problem.

  NMR (N-modular redundancy) is the generalization of the TMR approach. N (usually an odd number) modules are used in a majority voting arrangement.

- **Active (dynamic) redundancy** achieves fault tolerance by detecting the existence of faults and performing actions to remove the faulty hardware from the system (reconfiguration). Active redundancy is used in systems where the temporary erroneous results are acceptable as long as the system reconfigures itself.

  One example is the *standby spare* scheme. One module is operational while the others serve as stand-by spares. Error detection mechanisms are used to determine when the active module becomes faulty. The faulty module is removed from operation and replaced with one of the spares. Disruption in processing is minimized by *hot standby* spares which operate in synchrony with the active module and can take over the execution at any time. *Cold standby spares* are unpowered, *warm spares* perform some marginal tasks in reduced activity mode until they needed to replace the faulty module.

- **Hybrid redundancy** combines the attractive features of passive and active redundancy. Fault masking is used to prevent erroneous result to be generated. Fault detection, location and recovery also used to remove the faulty hardware and replace it with spares.

  Most hybrid technique is based on the NMR scheme. A basic core is arranged as an NMR with majority voting. In addition, spares are provided to replace failed units in the core.

Information redundancy

Information redundancy is the addition of redundant information to data. Good examples of information redundancy are

- parity codes: addition of an extra bit to a data word such that the resulting word has odd (or even) number of 1s.
- m-of-n codes: codewords are defined that are n bits in length and contain exactly m 1s.
- duplication codes: the original information is duplicated.
- checksums: a quantity of information added to a block of data to achieve error detection capability.
- cyclic codes, arithmetic codes, Hamming- and Berger-codes: the basic concept is the same, as in all other coding techniques. Data is encoded before the operation (transmission) is performed, then the resulting codewords are checked if they are valid.
- self-checking concepts: a circuit is self-checking if it has the ability to automatically detect the existence of a fault without any external stimulus.

Time redundancy

Time redundancy techniques attempt to reduce the extra hardware needed by the previous two techniques at the expense of additional time. (Some systems can involve additional execution time easier than additional hardware.)

The basic concept of time redundancy is the repetition of the computation in order to detect (and tolerate) the fault. Fundamentally, the same computation is executed in two or more times and the results are compared. If a discrepancy is detected then the computation is performed again to see if the disagreement remains or disappears. In this way transient faults can be detected and tolerated. The main problem is, that it is difficult to repeat the computation in the presence of a fault (efficient state restoration techniques are needed).

Software redundancy

Many error detection and fault tolerance techniques can be implemented in software (using the same underlying hardware as the normal operation). Redundant software can occur in many forms:

- **Consistency (acceptability) checks** use a-priori knowledge about the properties of the results to verify the correctness. For example, the range of variables or the required time to solve the problem could provide information to examine whether the output is acceptable or not.
- **Capability checks** verify that the system possesses the expected capability. Examples of capability checks are memory or ALU tests, tests of communication.
- **Fault tolerant software schemes** like N-version programming, recovery blocks, N self-checking programming are examined in details in a subsequent chapter.

Bibliography

[LA90] P. A. Lee and T. Anderson. *Fault Tolerance - Principles and Practice*, volume 3 of *Dependable Computing and Fault Tolerant Systems*. Springer Verlag, 1990.

[Lap92] J. C. Laprie. *Dependability: Basic Concepts and Terminology*, volume 5 of *Dependable Computing and Fault Tolerant Systems*. Springer Verlag, 1992.

[Lyu96] M. R. Lyu. *Handbook of Software Reliability Engineering*. McGraw-Hill, New York, 1996.

[Pra96] D. K. Pradhan. *Fault-Tolerant Computer System Design*. Prentice Hall PTR, New Jersey, 1996.