## 3. Fault, Error, and Failure:

- A ***fault*** is the identified cause of an error. It can be viewed as simply the consequence of a failure of some other system that has delivered or is now delivering a service to the given system. It is the manifestation of an error in software leading to an abnormal condition that can cause an element or an item to fail.
- An ***active fault*** is one that produces an error.
- An ***error*** is part of the system state that is liable to lead to a failure. An error may propagate, that is, produce other errors. Faults are known to be present when errors are detected.
- A ***failure:*** When a system or module is designed, its behavior is *specified*. When in service, we can *observe* its behavior. When the observed behavior differs from the specified behavior, we call it a *failure*.

  It is the Termination of the ability of an element to perform a function as required.

  **So, with software fault tolerance, the aim is to prevent failures by tolerating faults whose occurrences are known when errors are detected.**

  The cycle: **…failure→fault→error→failure→fault…**
  indicates their general causal relationship. The causal order is maintained, however the generality is shown when, for example, an error leads to a fault without an observed failure.

Whether or not an error will actually lead to a failure depends on three factors:

1. **The system composition**, and especially the nature of the existing redundancy:
   - **Intentional redundancy** (introduced to provide fault tolerance) which is explicitly intended to prevent an error from leading to failure,

- **Unintentional redundancy** (it is practically difficult if not impossible to build a system without any form of redundancy) which may have the same - unexpected – result as intentional redundancy.

2. **The system activity:** an error may be overwritten before creating damage.
3. **The definition of a failure from the user's viewpoint:** what is a failure for a given user may be a bearable nuisance for another one.

**Faults can be *hard* or *soft* (transient):**

- A module with a **hard fault** will not function correctly, it will continue with a high probability of failing – until it is repaired. A hard fault could be a device with a burnt-up component. This will certainly not fix itself.
- A module with a **soft fault** appears to be repaired after the failure. A soft fault could be electrical noise interfering with the computer. If this has made a data transport on a bus fail, a second attempt could work if there is no noise at that time.

*Example:*

A failure occurs because of an error, caused by a fault. The time between the occurrence of an error and the resulting failure is the *error latency*.

```
FUNCTION div(a, b: REAL):REAL
BEGIN
    div := a/b;
END;
```

This function has a fault; it does not check the value of the variable b. This fault results in a latent error in the function div. If the function is executed with a zero-value as b-argument, that is an *error*. When the division is executed, we have a program failure.

## 4. Dependable Fault-tolerant Systems

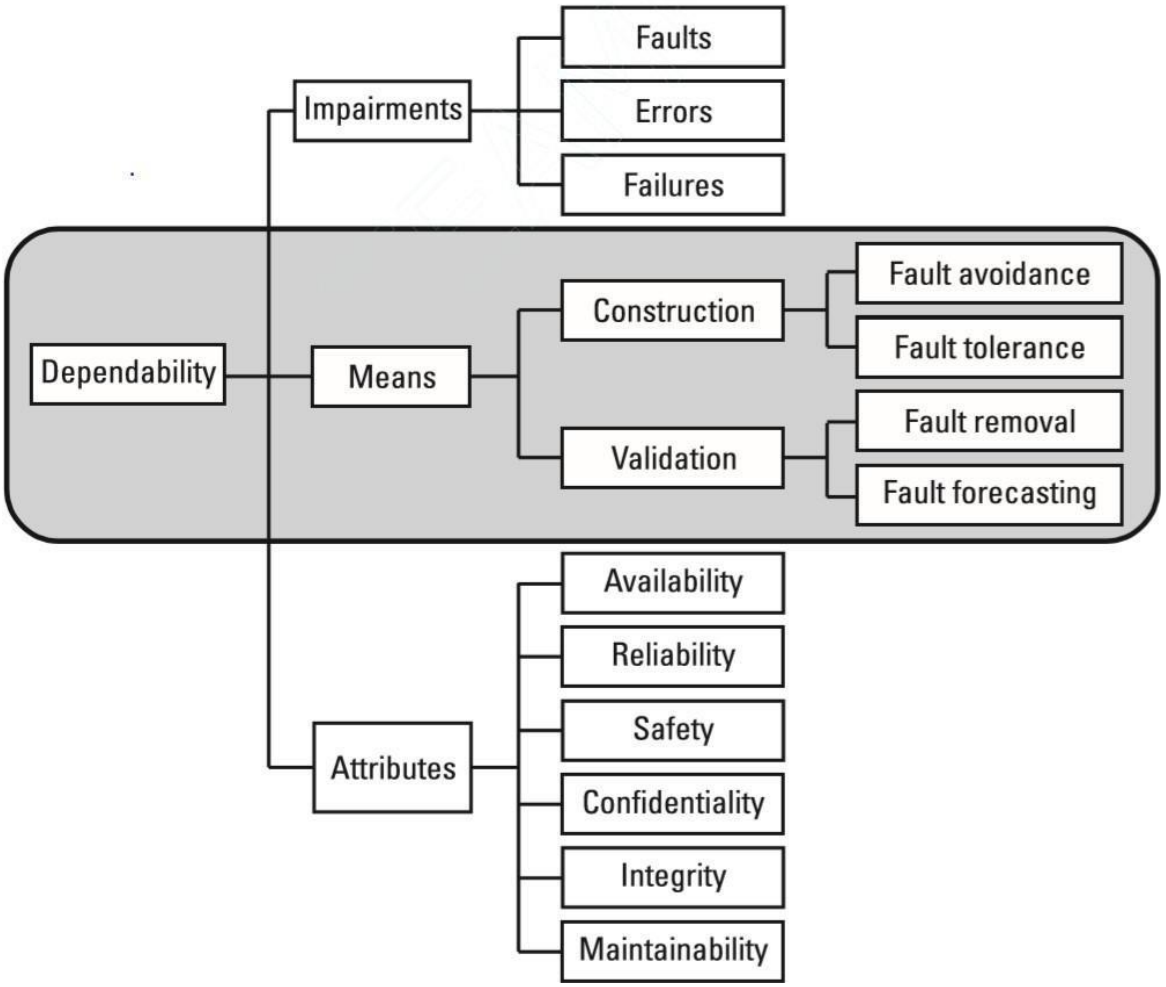
Dependability is that property of a computer system such that reliance can justifiably be placed on the service it delivers. The service delivered by a system is its behavior as it is perceived by its user(s); a user is another system (physical, human) which interacts with the former.

Depending on the application(s) intended for the system, dependability may be viewed according to different properties which enable the attributes of dependability to be defined:

- the readiness for usage leads to ***availability***;
- the continuity of service leads to ***reliability***;
- the non-occurrence of catastrophic consequences on the environment leads to ***safety***;
- the non-occurrence of unauthorized disclosure of information leads to ***confidentiality***;
- the non-occurrence of improper alterations of information leads to ***integrity***;
- the aptitude to undergo repair and evolution leads to ***maintainability***;

The development of a dependable computing system demands the combined utilization of a set of methods, these methods can be classed into:

- **Fault Prevention**: how to prevent fault occurrence or introduction;
- **Fault Tolerance**: how to provide a service that fulfills the system function in spite of faults;
- **Fault Removal**: how to reduce the presence (number, seriousness) of faults;
- **Fault Forecasting**: how to estimate the present number, the future incidence, and the consequences of faults.




The notions of Dependability can be grouped into three classes:

1. **The Attributes of Dependability**: enable the properties which are expected from the system to be expressed, and allow the system quality resulting from the impairments and the means opposing to them to be assessed.

   They are: availability, reliability, safety, confidentiality, integrity, maintainability.

2. **The Means for Dependability**:

The means to achieve dependability falls into two major groups:

- **Construction:** those that are employed during the software construction process (fault avoidance and fault tolerance), and