Threats to dependability

Faults

Faults

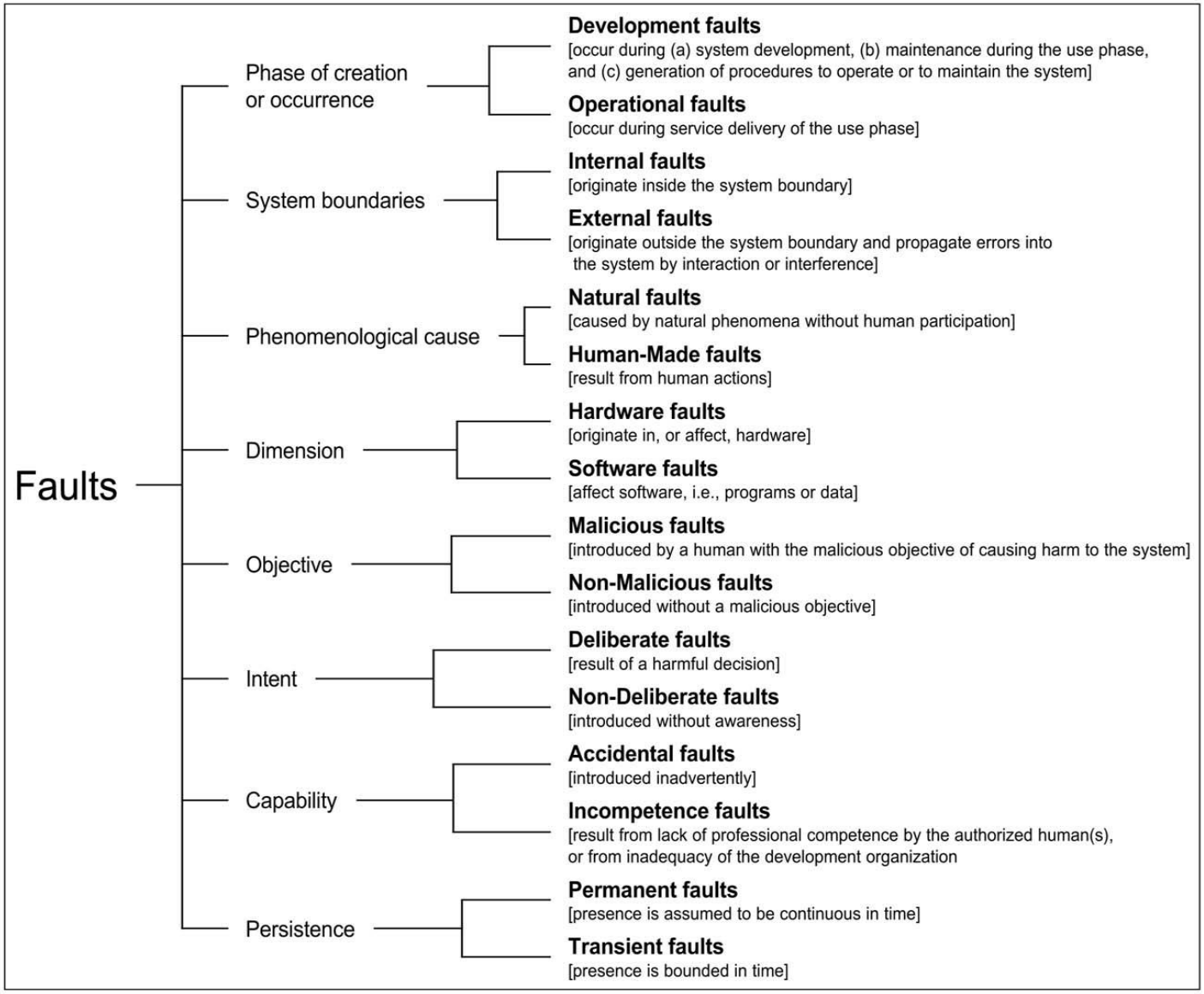
All different faults that may affect a system during its life cannot be enumerated

We can classify faults. Classifiction of faults is important because we can identify which mechanisms protect us from a given class of faults.

Faults are classified according to basic viewpoints

Classification of faults according to eight basic viewpoints

From A. Avizienis, J.C. Laprie, B. Randell, C. Landwehr. Basic Concepts and Taxonomy of Dependable and Secure Computing, IEEE Transactions on Dependable and Secure Computing, Vol. 1, N. 1, 2004



Phase of creation

this viewpoint identifies when the fault or the reason for the fault was created

The life cycle of a system consists of two phases:

- development phase

from specification to development and testing. At the end of this phase the system is ready to deliver service

The **development environment** of a system consists of the following elements:

1. the physical world with its natural phenomena
2. human developers, some possibly lacking competence or having malicious objectives
3. development tools: software and hardware used by the developers to assist them in the developmen process
4. production and test facilities

- use phase

The use phase begins when the system is accepted for use and starts the delivery of its services to the users.
Use consists of alternating periods of
- correct service delivery (service delivery)
- service outage (period in which incorrect service is delivered including no service)
- service shutdown (intentional halt of the service by a authorized entity)

The **use environment** of a system consists of the following elements:

1. the physical world with its natural phenomena
2. Administrators including maintainers
3. Users (receive service from the system at the interface)
4. Providers (deliver service to the system at the interface)
5. The infrustructure (provides specialized services to the system e.g., GPS)
6. Intruders : malicious entities (humans or other systems) that could alter the service

Development faults:
faults occur during the system development or maintenance during the use phase

Operational faults:
the fault occur during the service delivery of the use phase

System boundary

Internal faults: originate inside the system boundary

External faults: originate outside the system boundary and propagates errors into the system by interaction or interference

Phenomenological cause

This viewpoint identifies the nature of faults: caused by natural phenomena and/or humans

Natural faults (physical faults) relevant in hw

faults that are caused by natural phenomena without human participation.

Hw mainly breaks due to physical effects

Human-made faults (relavant in sw)

Result from human actions

Sw faults are related to programming

Prevention technologies are completely different:

Human-made faluts: rigorous development, testing, …

Natural faults: high quality material, optimize operation condition (temperature), shield the hardware, cooling ….

Dimension

Hardware faults: originate in or affect the hardware

Software faults: affect the software (programs or data)

Objective

Non-malicious faults: introduced by a human without malicious objectives

Malicoius faults: introduced by a human with the malicious objective of causing harm to the system

Intent

Deliberate faults: result of a harmful decision

Non-deliberate faults: introduced without awareness

Capability

Accidental faults: introduced inadvertently

Incompetence faults: result from a lack of professional competence by the authorized humans or by inadequacy of the development organization

Persistence

Temporary faults: a fault that can appear and disappear within a very short period of time. Faults that go away from themselves (e.g., short power outages)

Permanent faults: a fault continuous and stable.
It remains in existence if no corrective action is taken (e.g., disk sector damage).

Classes of combined faults

Not all viewpoints are applicable to all fault classes
(e.g. natural faults cannot be classified by Intent)

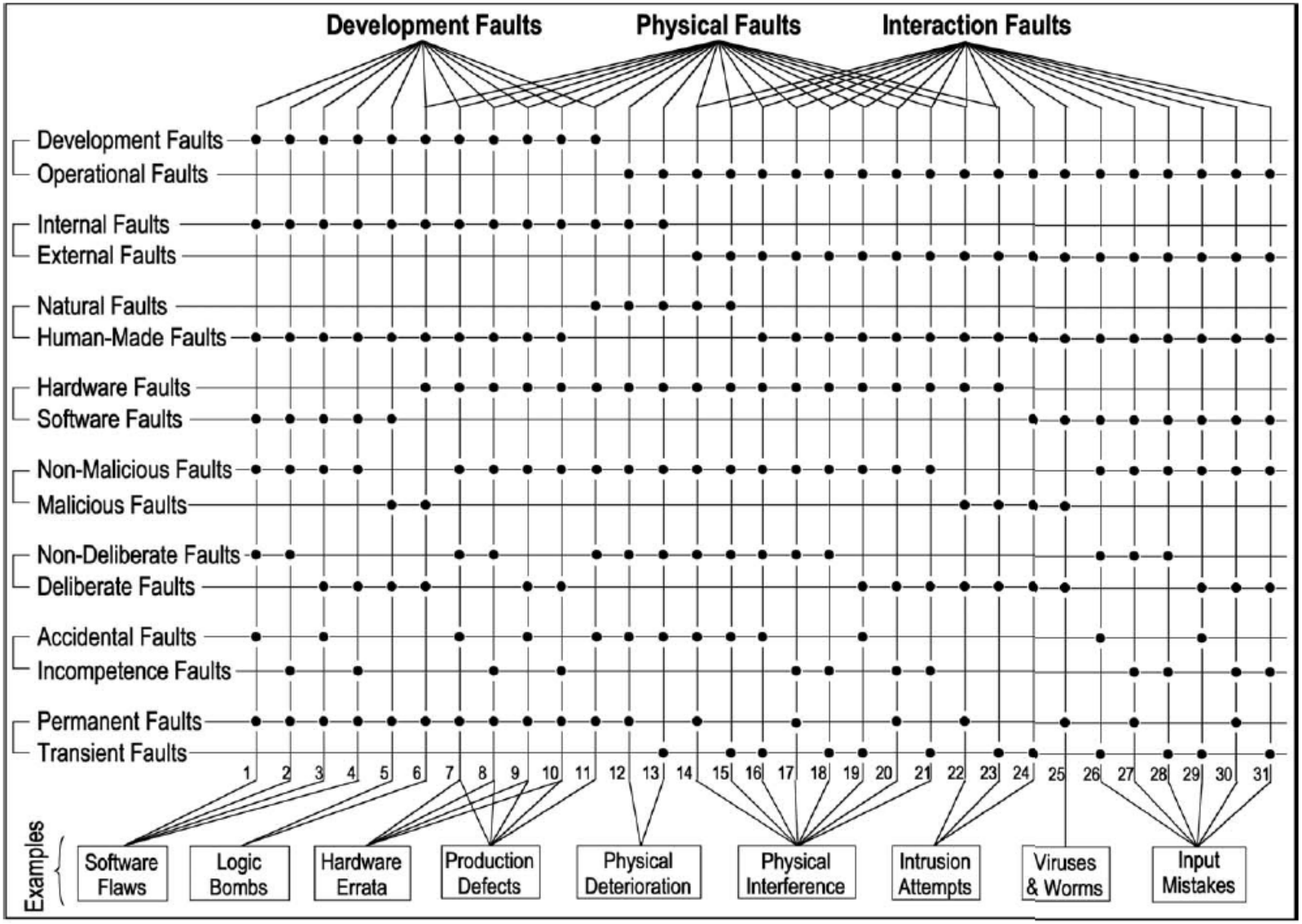
31 identified combinations
that belong to three major partially overlapping groupings

Development faults
that include all fault classes occurring during development

Physical faults
that include all fault classes that affect hardware

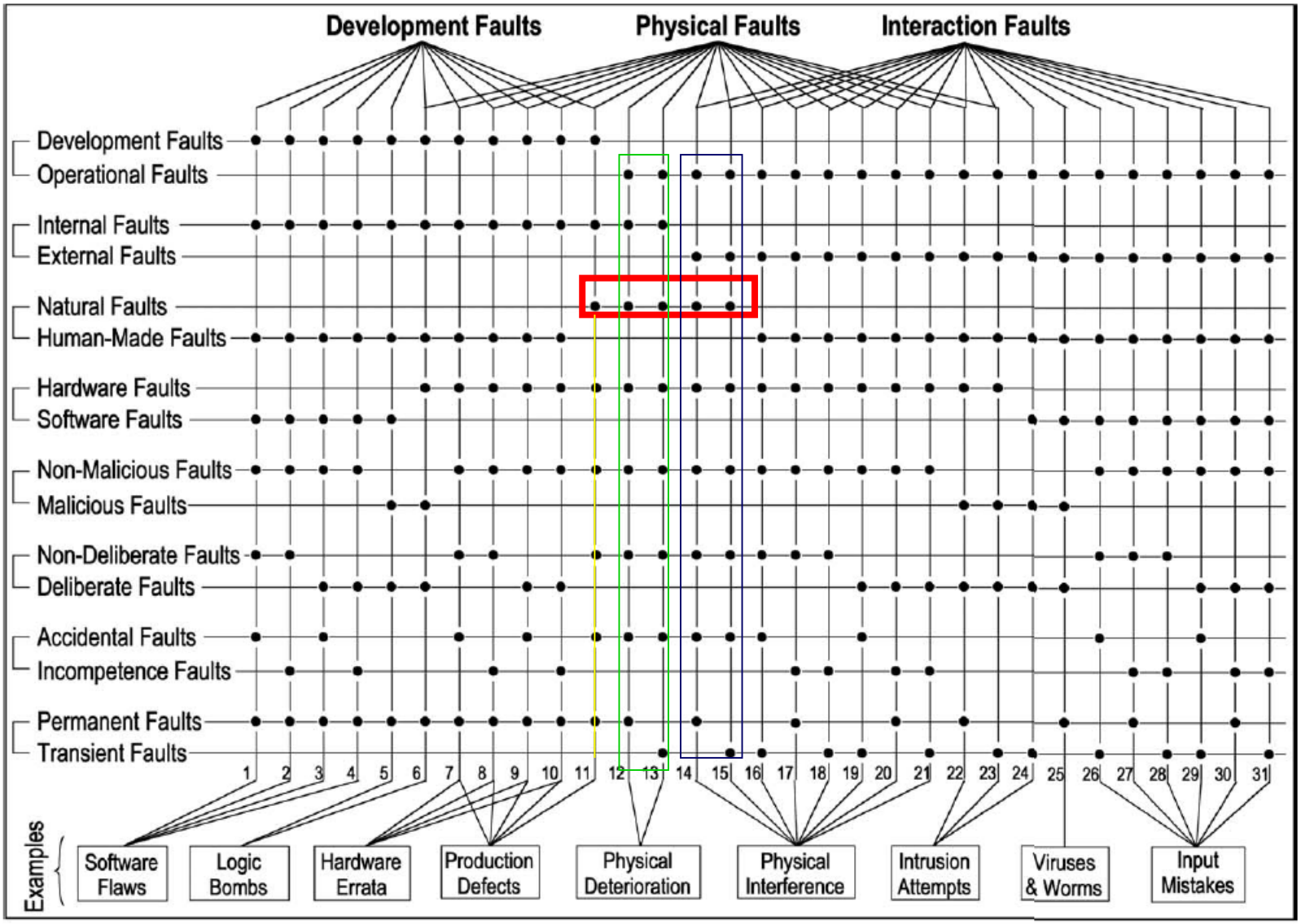
Interaction faults
that include all external faults.

Classification schema



names of some illustrative fault

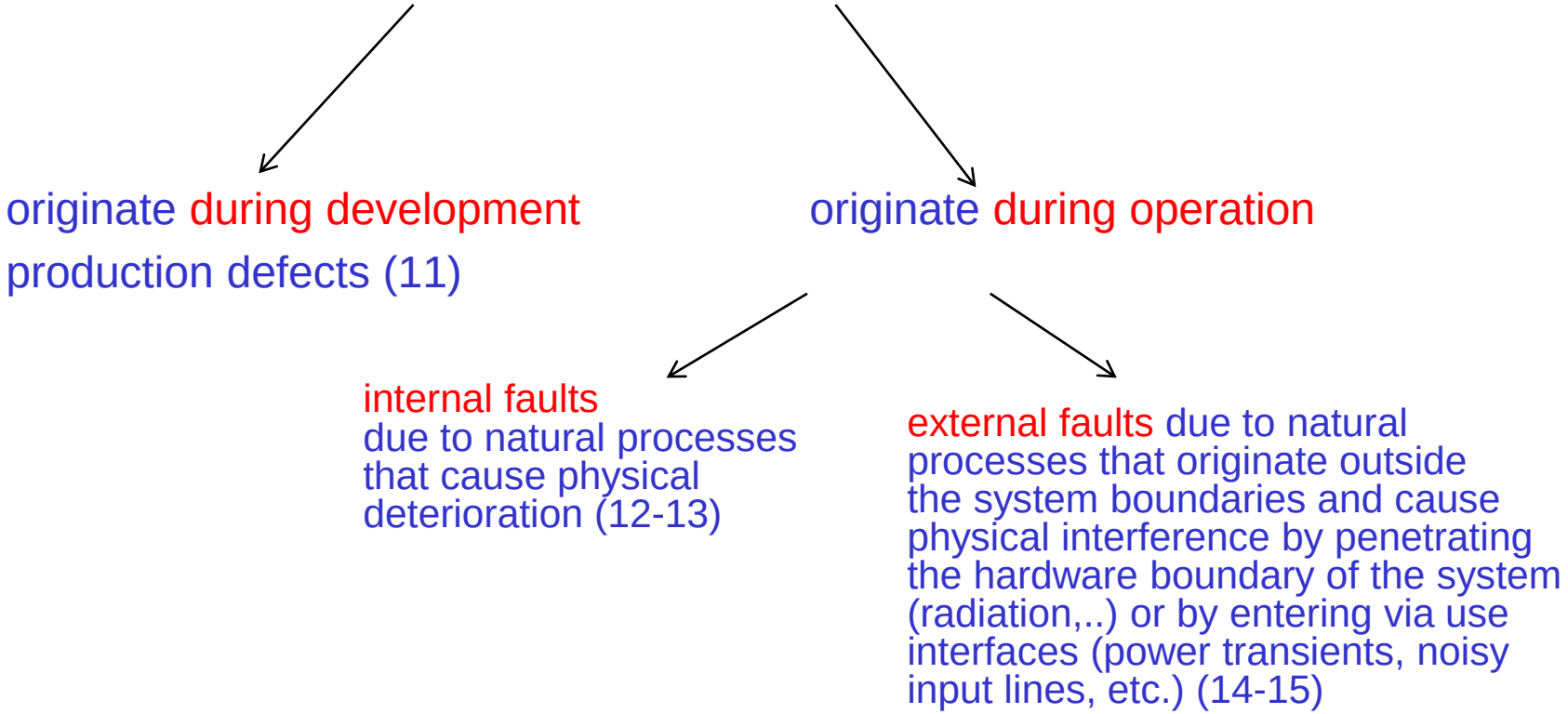
Natural faults



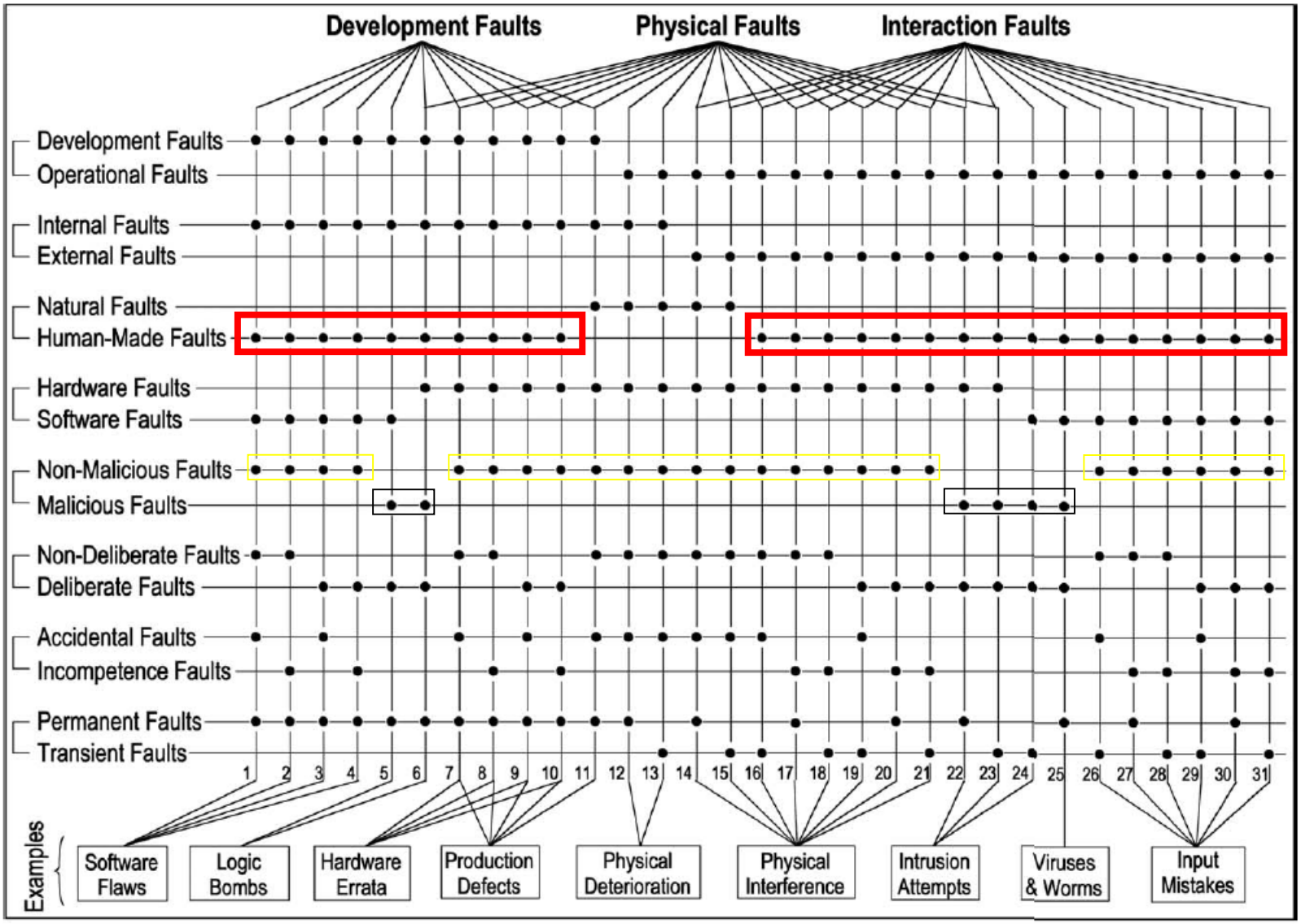
From A. Avizienis, J.C. Laprie, B. Randell, C. Landwehr. Basic Concepts and Taxonomy of Dependable and Secure Computing, IEEE Transactions on Dependable and Secure Computing, Vol. 1, N. 1, 2004

Natural faults

Natural faults (11-15) are hardware faults



Human-Made Faults



From A. Avizienis, J.C. Laprie, B. Randell, C. Landwehr. Basic Concepts and Taxonomy of Dependable and Secure Computing, IEEE Transactions on Dependable and Secure Computing, Vol. 1, N. 1, 2004

Human-Made Faults

result from human actions

Malicious faults, introduced during either system development with the objective to cause harm to the system during its use (5-6), or directly during use (22-25).

Non-malicious faults introduced without malicious objectives (1-4, 7-21, 26-31)

Deliberate faults: faults that are due to bad decisions, that is, intended actions that are wrong and cause faults (3, 4, 9, 10, 19-21, 29-31).

Non-deliberate faults that are due to mistakes, that is, unintended actions of which the developer, operator, maintainer, etc. is not aware (1, 2, 7, 8, 16-18, 26-28);

Development faults (3, 4, 9, 10) result generally from tradeoffs, either aimed at preserving acceptable performance, at facilitating system utilization, or induced by economic considerations

Interaction faults (19-21, 29-31) may result from the action of an operator either aimed at overcoming an unforeseen situation, or deliberately violating an operating procedure without having realized the possibly damaging consequences of this action.

Deliberate, nonmalicious faults are often recognized as faults only after an unacceptable system behavior (failure).

The developer(s) or operator(s) did not realize at the time that the consequence of their decision was a fault.

Malicious faults are all DELIBERATE faults

Development physical (hardware) faults: microprocessor faults discovered after production (named Errata). They are listed in specification updates

Human-Made Deliberate Non-malicious Faults

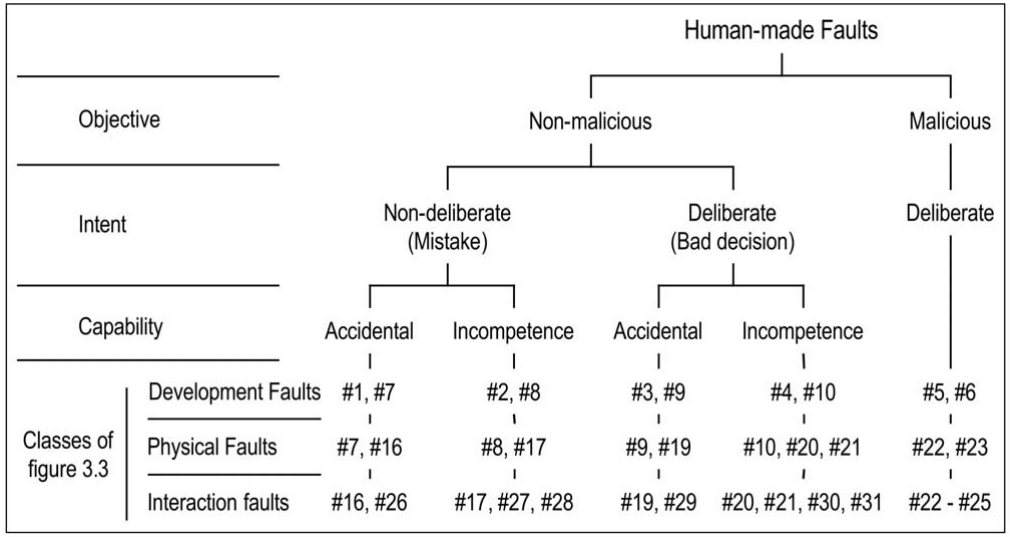
Not all mistakes and bad decisions by nonmalicious persons are accidentals.

A further partitioning is introduced (Capability viewpoint):

accidental faults

incompetence faults
result from lack of professional competence
or inadequacy of the development organization

How to recognize incompetence faults? Important when consequences that lead to economic losses, injuries or loss of human life.



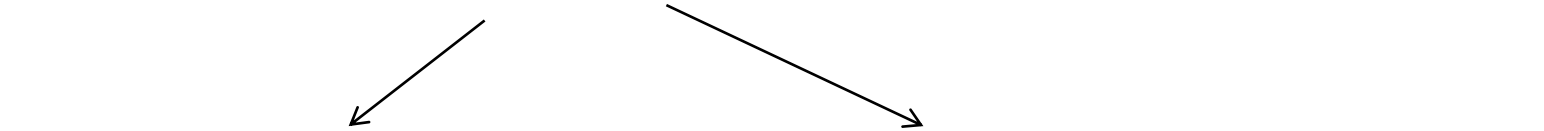
From A. Avizienis, J.C. Laprie, B. Randell, C. Landwehr. Basic Concepts and Taxonomy of Dependable and Secure Computing, IEEE Transactions on Dependable and Secure Computing, Vol. 1, N. 1, 2004

Malicious faults

Malicious human-made faults are introduced with the malicious objective to alter the functioning of the system during use.

The goals of such faults are:

- to disrupt or halt service, causing denials of service;
- to access confidential information; or
- to improperly modify the system.



Developent faults (5,6)
|
Internal
|
Permanent

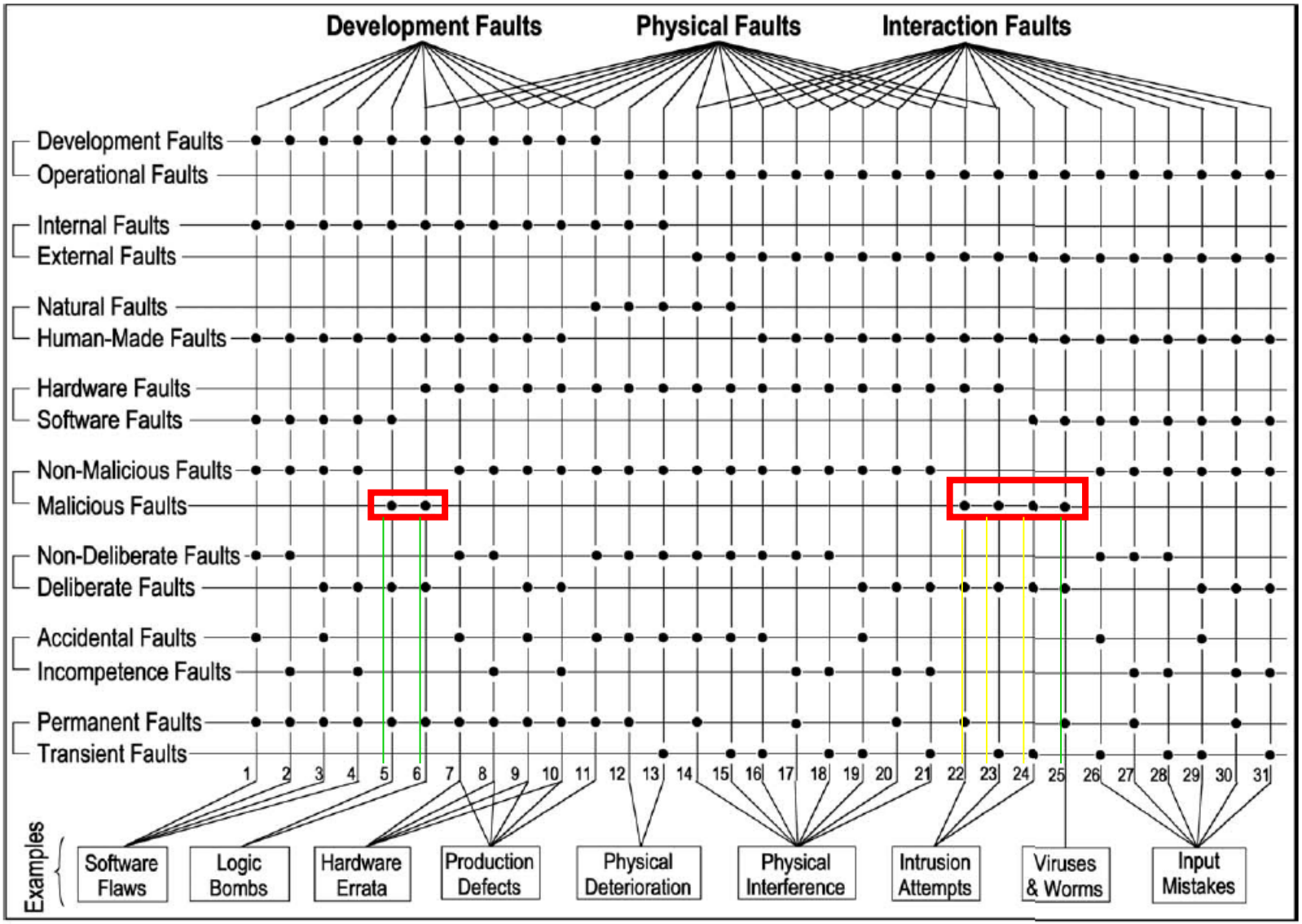
(5) SW
(6) HW
Trojan horses, logic bombs, trapdoors, ….

Operational faults (22-24)
|
External
| |
Permanent Transient
(22) intr attempts (HW) (23) intr attempts HW
(25) Viruses or worms (SW) (24) intr attempts SW

intrusion attempts:
- may be performed by system operators or administrators who are exceeding their rights
- may use physical means to cause faults: power fluctuation, radiation, wire-tapping, heating/cooling, etc.

Malicious logic faults: (5,6) + (25)
Intrusion attempts: (22,23,24)

Human-Made Malicious faults



From A. Avizienis, J.C. Laprie, B. Randell, C. Landwehr. Basic Concepts and Taxonomy of Dependable and Secure Computing, IEEE Transactions on Dependable and Secure Computing, Vol. 1, N. 1, 2004

Examples

An “exploit” is a software script that will exercise a system vulnerability and allow an intruder to gain access to, and sometimes control of, a system. Invoking the exploit is an operational, external, human-made, software, malicious interaction fault (24-25).

The vulnerability that an exploit takes advantage of is typically a software flaw (e.g., an unchecked buffer) that could be characterized as a developmental, internal, human-made, software, nonmalicious, nondeliberate, permanent fault (1-2).

Heating the RAM with a hairdryer to cause memory errors that permit software security violations would be an external, human-made, hardware, malicious interaction fault (22-23).

Interaction Faults

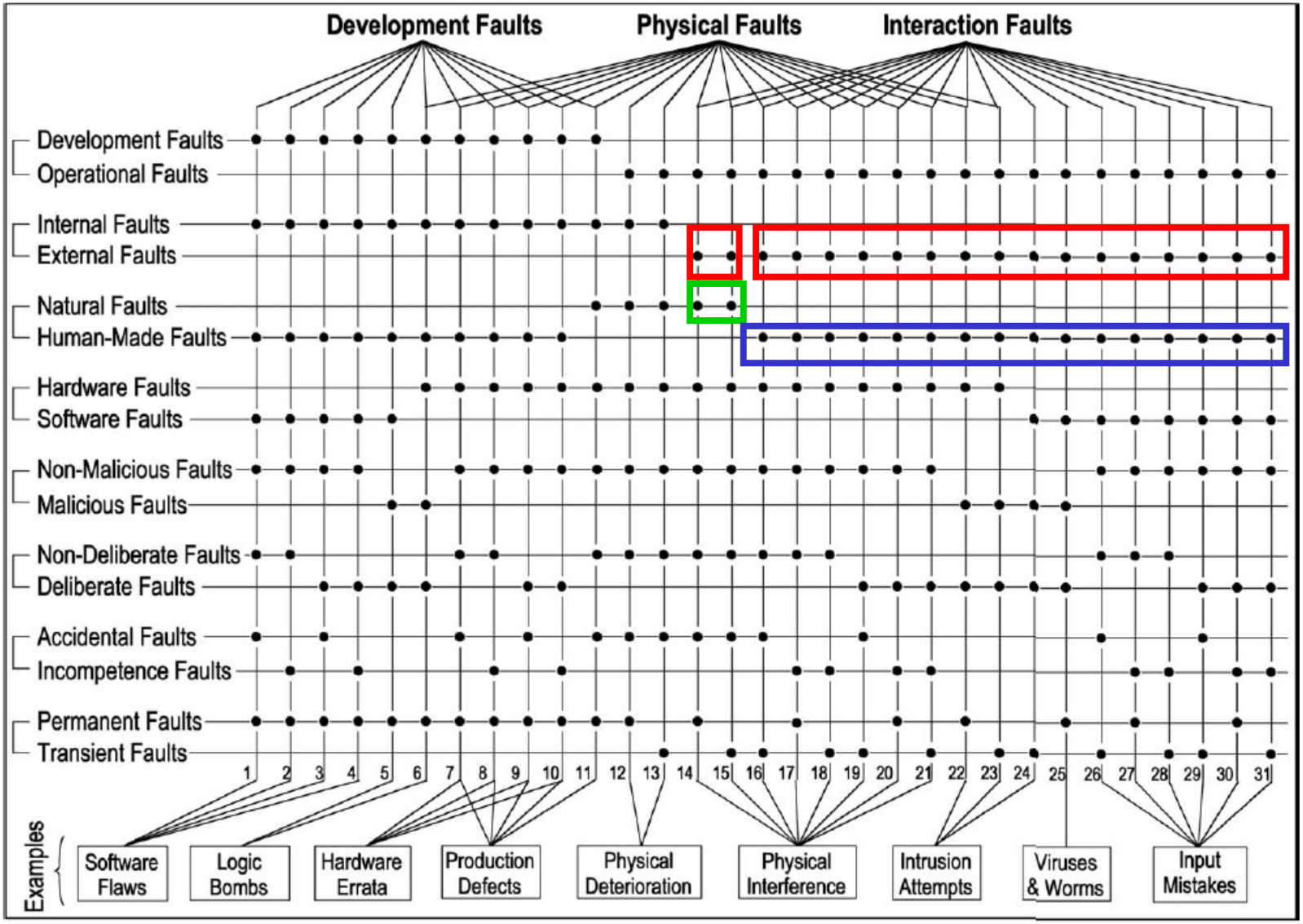
occur during the **use phase**, therefore they are all **operational faults**. They are caused by elements of the use environment interacting with the system; therefore, they are all **external**.

Human-made (16-31)
most classes originate due to some human action in the use environment

External natural faults (14-15)
caused by cosmic rays, solar flares, etc. nature interacts with the system without human participation.

Configuration faults (i.e., wrong setting of parameters that can affect security, networking, storage, middleware, etc.): a broad class of human-made operational faults. Such faults can occur during configuration changes performed during adaptive or augmentative maintenance performed concurrently with system operation

Interaction faults



From A. Avizienis, J.C. Laprie, B. Randell, C. Landwehr. Basic Concepts and Taxonomy of Dependable and Secure Computing, IEEE Transactions on Dependable and Secure Computing, Vol. 1, N. 1, 2004

A common feature of interaction faults is that, in order to be "successful," they usually necessitate the prior presence of a vulnerability, i.e., an internal fault that enables an external fault to harm the system.

Vulnerabilities can be development or operational faults; they can be malicious or nonmalicious, as can be the external faults that exploit them.

A vulnerability can result from a deliberate development fault, for economic or for usability reasons, thus resulting in limited protections, or even in their absence.

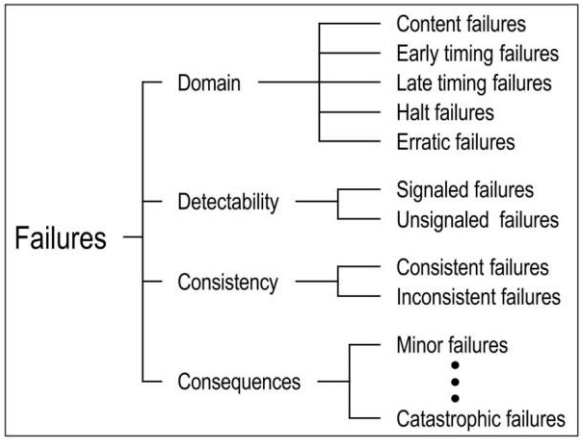
Failures

Service Failures

A service failure (failure) is defined as an event that occurs when the delivered service deviates from correct service (function of the system).

The failure modes characterize the deviation of the incorrect service according to four viewpoints:

1. the failure domain,
2. the detectability of failures,
3. the consistency of failures, and
4. the consequences of failures on the environment.



From A. Avizienis, J.C. Laprie, B. Randell, C. Landwehr. Basic Concepts and Taxonomy of Dependable and Secure Computing, IEEE Transactions on Dependable and Secure Computing, Vol. 1, N. 1, 2004

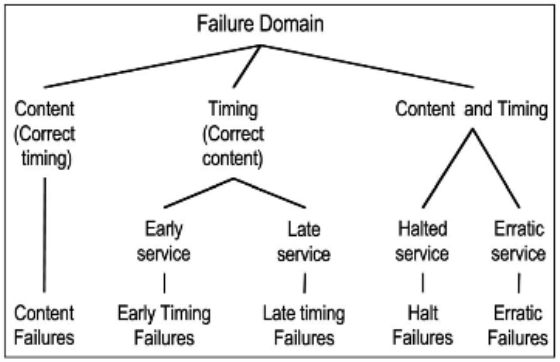
Functional specification: description of the function of the system

Service compliant with the specification may be unacceptable for users (due to specification faults)

Specification faults: the specification not adequately describe the system function

-misinterpretations, unwarrented assumptions, inconsistencies, ….

Failures



1. Failure domain viewpoint:

content failures
the content of the information delivered at the service interface deviates from implementing the system function.

timing failures
the time of arrival or the duration of the information delivered at the service interface deviates from implementing the system function.

When information and time are incorrect we have:

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); a special case of halt is silent failure, or simply silence, when no service at all is delivered at the service interface (e.g., no messages are sent in a distributed system).

erratic failures
when a service is delivered (not halted), but is erratic (e.g., babbling - the system repeatedly fails).

2. Detectability viewpoint:

signaled failures: when the failures are detected and signaled by a warning signal to the users (based on a detecting mechanisms in the system that check the correctness of the delivered service).

unsignaled failures: otherwise.

The failure detecting mechanism may fail!! Two possible failure modes:

- false alarm (signaling a loss of function when no failure has actually occurred)
- unsignaled failure (not signaling a function loss).

When the occurrence of service failures result in reduced modes of service, the system signals a degraded mode of service to the user(s).

3. Consistency viewpoint (when a system has two or more users):

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, Byzantine failures.

Failures

4. Consequences viewpoint:

grading the consequences of the failures upon the system environment enables failure severities to be defined.

For each severity level, we have an associated maximum acceptable probability of occurrence

Two severity limiting levels can be defined according to the relation between the benefit provided by the service delivered in the absence of failure, and the consequences of failures:

minor failures

the harmful consequences are of similar cost to the benefits provided by correct service delivery

catastrophic failures

the cost of harmful consequences is orders of magnitude, or even incommensurably, higher than the benefit provided by correct service delivery

Development failures

Development failures:

development process terminated before the system is accepted for use and placed into service

Two aspects:

- budget failure
- schedule failure (project delivery delayed to a point in which the system is obsolete, or functionally inadequate)

Principal causes:

incomplete or faulty specifications

excessive number of user specification changes

inadequate design

inadequate fault removal capability

……..

Dependability failures

Dependability specification

- identifies classes of faults that are expected
- the use environment in which the system will operate
- specific modes of failures of the system
- acceptable occurrence rates of failures according to severity level

example: fail-stop system

failures are halt failures (constant service delivery)

1) fail-sailent : service absent

2) fail-passive: stuck-at failure

example: fail-safe system: all failures are minor failures

May include:

- requests for tolerating specific situations
- requests for the inclusion of specific fault prevention or fault tolerance techniques

States goals for each attribute: Availability, Reliability, Safety

Dependability failure:

service failure more frequently or more severe than expected

Dependability failures

Dependability specification may contain faults:

- Omission faults in the description of the environment
- Choices of classes of faults to be tolerated
- Unjustified choice for very high requirements for one or more dependability attributes

  outage limit: 3 sec for year

  5min for year

Errors

Errors

An error can be:

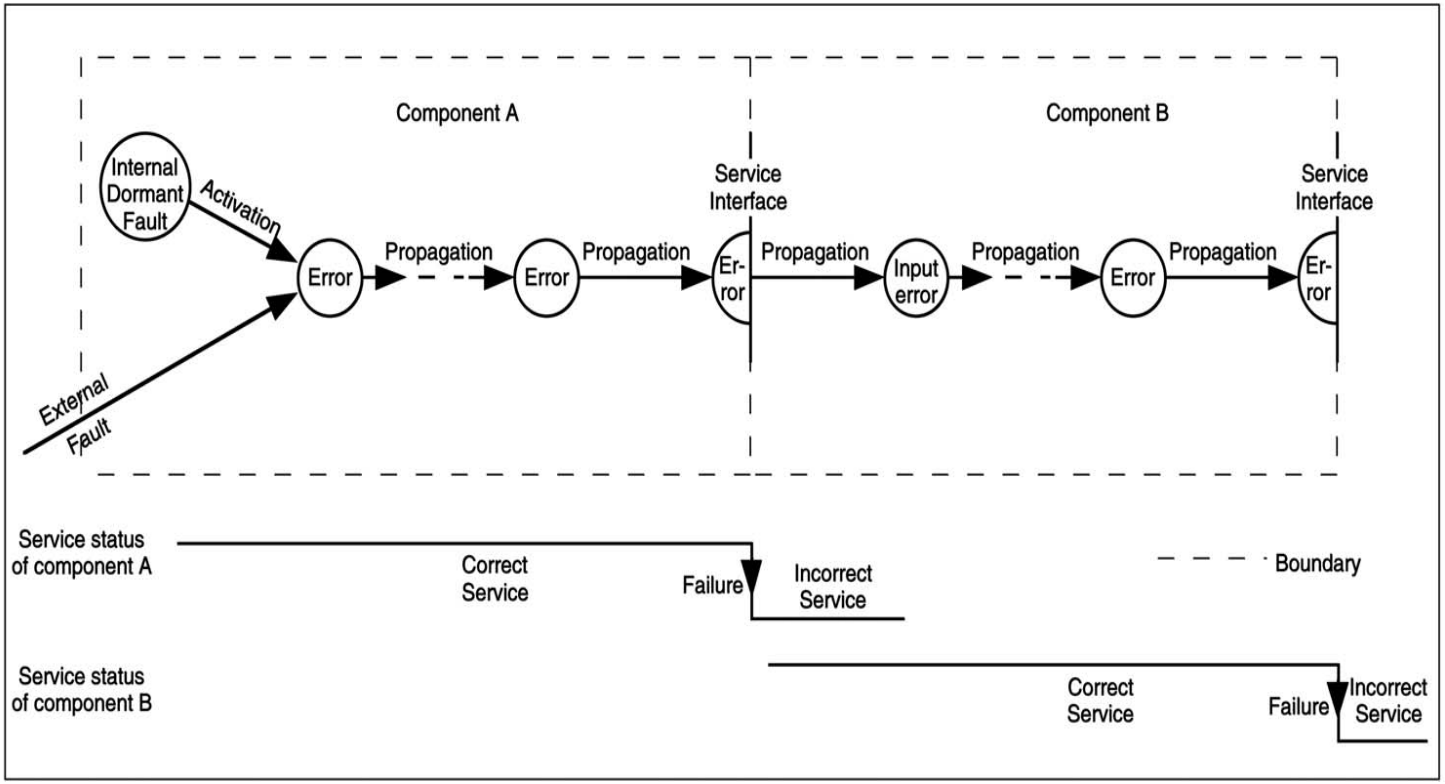
- detected if its presence is indicated by an error message or error signal.
- latent if it is present but not detected

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

1. The structure of the system, and especially the nature of any redundancy that exists in it: protective redundancy, introduced to provide fault tolerance, that is explicitly intended to prevent an error from leading to service failure. Unintentional redundancy (it is in practice difficult if not impossible to build a system without any form of redundancy) that may have the same presumably unexpected result as intentional redundancy.
2. The behavior of the system: the part of the state that contains an error may never be needed for service, or an error may be eliminated (e.g., when overwritten) before it leads to a failure.

Some faults (e.g., a burst of electromagnetic radiation) can simultaneously cause errors in more than one component. Such errors are called multiple related errors. Single errors are errors that affect one component only.

Chain of threats: Faults-Errors-Failures



From A. Avizienis, J.C. Laprie, B. Randell, C. Landwehr. Basic Concepts and Taxonomy of Dependable and Secure Computing, IEEE Transactions on Dependable and Secure Computing, Vol. 1, N. 1, 2004

Chain of threats

A fault is active when it produces an error; otherwise, it is dormant.

An active fault is either

- an internal fault that was previously dormant and that has been activated by the computation process or environmental conditions, or
- an external fault.

Fault activation:
is the application of an input (the activation pattern) to a component that causes a dormant fault to become active.

Most internal faults cycle between their dormant and active states.

Code is full of unactivated faults (faults are named bugs)

Testing can check the presence of faults, not the absence of faults.

We are interested in faults that may be activated.

We use the control flow and test coverage.

Chain of threats

Error propagation within a given component (i.e., internal propagation) is caused by the computation process.

An error is successively transformed into other errors.

Error propagation from component A to component B that receives service from A (i.e., external propagation) occurs when, through internal propagation, an error reaches the service interface of component A.

At this time, service delivered by A to B becomes incorrect, and the ensuing service failure of A appears as an external fault to B and propagates the error into B via its use interface.

Chain of threats

A service failure occurs when an error is propagated to the service interface and causes the service delivered by the system to deviate from correct service.

The failure of a component causes a permanent or transient fault in the system that contains the component.

Service failure of a system causes a permanent or transient external fault for the other system(s) that receive service from the given system.

Chain of threats

Given a system with defined boundaries, a single fault is a fault caused by one adverse physical event or one harmful human action.

Multiple faults are two or more concurrent, overlapping, or sequential single faults whose errors overlap in time, that is, the errors due to these faults are concurrently present in the system.

Consideration of multiple faults leads one to distinguish independent faults, that are attributed to different causes, and related faults, that are attributed to a common cause.

Related faults generally cause similar errors, i.e., errors that cannot be distinguished by whatever detection mechanisms are being employed

Independent faults usually cause distinct errors.

However, it may happen that independent faults (especially omissions) lead to similar errors, or that related faults lead to distinct errors.

Dependable system

Point 1)

Assumptions on how the system is used: very important

External faults during normal operation, caused by wrong assumptions on the operational conditions

Sometimes operational conditions are underspecified.

Point 2)

Faults are unexpected events. Something is happening in the system that we did not plan before.

How can we build a system that tolerates faults if we do not know faults?

«information from literature (knowledge of fault classes) and from experience allows the user to decide which faults should be included in the dependability specification»

Specification of the fault free system + fault assumption

Point 3)
Fault classes are relevant to choose the dependability mean.

Example: temporary/permanent faults

How can you deal wih temporary faults?

In the case of short power outages: extra battery can be used as fault tolerance mechanism

In case of network connections problems (network and connectivity are assumed temporary problem): retry can be used as fault tolerant mechanism.

Additional problem: if the net is partitioned, retry does not help.

How can you deal wih permanent faults?
Redundancy (you need to have a spare sw or hw component)

Exceptions in programs

How can you deal wih exceptions?
Catch is used as fault tolerant mechanism.