Real-Time Systems & Fault Tolerance

Flávia Maristela

Instituto Federal da Bahia
Especialização em Computação Distribuída e Ubíqua (ECDU)

Salvador, Outubro de 2013

Schedule

Looking at the past...

Looking at the past...

1940 : ENIAC used to fail once every 2 days (average down time in **minutes**)

1966 : ARIANE 5 rocket was destroyed 37 seconds after its launch (overflow calculus error)

1985 : In Canada, a radiation therapy device malfunctioned and delivered lethal radiation doses for two years (Therac-25 medical accelerator)

1986 : Chernobyl - Operation errors and lack of security in the system

Looking at the past...

1993 : A silicon error causes Intel's highly promoted Pentium chip to make mistakes when dividing floating-point numbers that occur within a specific range

1998 : Mars Climate Orbiter was lost 9 months after its launch because there was a software fault (unit conversion - overflow)

1999 : Mars Polar Lander was missed 10 months after its launch due to an error in landing calculation (electromagnetic interference)

2000 National Cancer Institute (Panama) - therapy planning software miscalculates the proper dosage of radiation for patients undergoing radiation therapy.

... to understand the future

02/2012 : Programming error doomed Russian Mars probe - it fails to escape earth orbit due to simultaneous reboot of two subsystems

03/2012 : Eighteen companies sued over mobile apps - Facebook, Apple, Twitter, and Yelp are among the companies sued over gathering data from the address books of millions of smartphone users

05/2012 : Automatic Updates Considered Zombieware - Software updates take up much time/space; no one knows what's in them

07/2012 : A320 lost 2 of 3 hydraulic systems on takeoff. There were no loss of life; only passengers discomfort. Full account of incident not yet available, but it shows that redundancy alone is not a sufficient protection.

- Environment increased dramatically: scope and complexity (pervasive computing).
- Some systems have a high degree of complexity
  - Aircraft
  - Air traffic control
  - Medical devices
  - Nuclear reactors
  - High-speed trains
  - Banking
  - Nautical and military systems
- The consequences of failure of these systems can be soft catastrophic

Mobile, Pervasive and Ubiquitous

- Mobile Computing
  - The user can take the equipment anywhere without losing the computational power
- Pervasive computing
  - computer invisible to the user
  - it is necessary to obtain information about the environment in which user is inserted
  - Application is adjusted to better meet user needs (requirements)

Mobile, Pervasive and Ubiquitous

- Ubiquitous Computing
  - Mobility + Pervasive Omnipresence
  - The system can operate dynamically adjusting itself according to the context in which user is

- Software normally take most of the functionality of these systems pervasive and ubiquitous critical
- Errors in them can be catastrophic
- Despite advances, not all errors are identified
- IDEs help minimize these errors
- Errors can be corrected using artifacts

Why do we study fault-tolerance?

- Some systems requirements:
  - Functionality
  - Usability
  - Performance
  - **Dependability**
  - Cost
  - Manageability
  - Adaptability

Why do computer system fails?

1. Personal factors (35%): Lack of skill, lack of interest or motivation, fatigue, poor memory, age or disability
2. System design (20%): Insufficient time for reaction, tedium, lack of incentive for accuracy, inconsistent requirements or formats
3. Written instructions (10%): Hard to understand, incomplete or inaccurate, not up to date, poorly organized
4. Training (10%): Insufficient, not customized to needs, not up to date

Why do computer system fails?

1. Human-computer interface (10%): Poor display quality, fonts used, need to remember long codes, ergonomic factors
2. Accuracy requirements (10%): Too much expected of operator
3. Environment (5%): Lighting, temperature, humidity, noise

Most faults are related to software!

Why do computer system fails?

1. Human-computer interface (10%): Poor display quality, fonts used, need to remember long codes, ergonomic factors
2. Accuracy requirements (10%): Too much expected of operator
3. Environment (5%): Lighting, temperature, humidity, noise

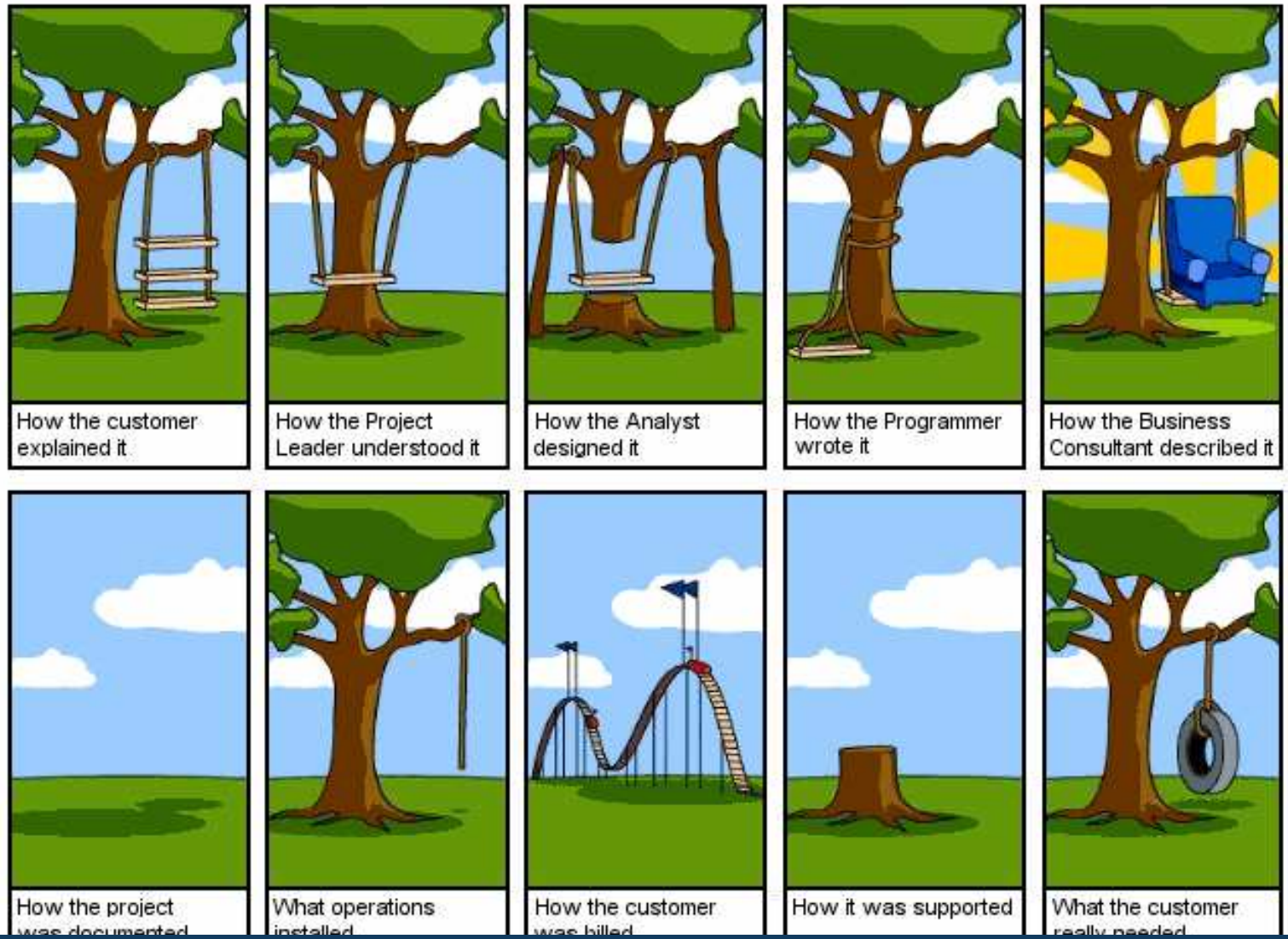
Most faults are related to software!

1. We must not forget faults that occur due to components obsolescence!!

"Computer engineering is the art and science of translating user requirements we do not fully understand; into hardware and software we cannot precisely analyze; to operate in environments we cannot accurately predict; all in such a way that the society at large is given no reason to suspect the extent of our ignorance."

Adapted from **Why There Are No Locks on the Bathroom Doors in the Hotel Louis XIV and Other Object Lessons**, Ralph Kaplan,, Fairchild Books, 2004, p. 229

Some design problems



Definition

- the ability of providing a trustful services

Definition

Dependability is an ability that is related to computing systems that continue to operate satisfactorily in the presence of faults

Basic Concepts

Failure

failure occurs when there is a transition from an expected behavior (correct) to a behavior that is not expected (incorrect)

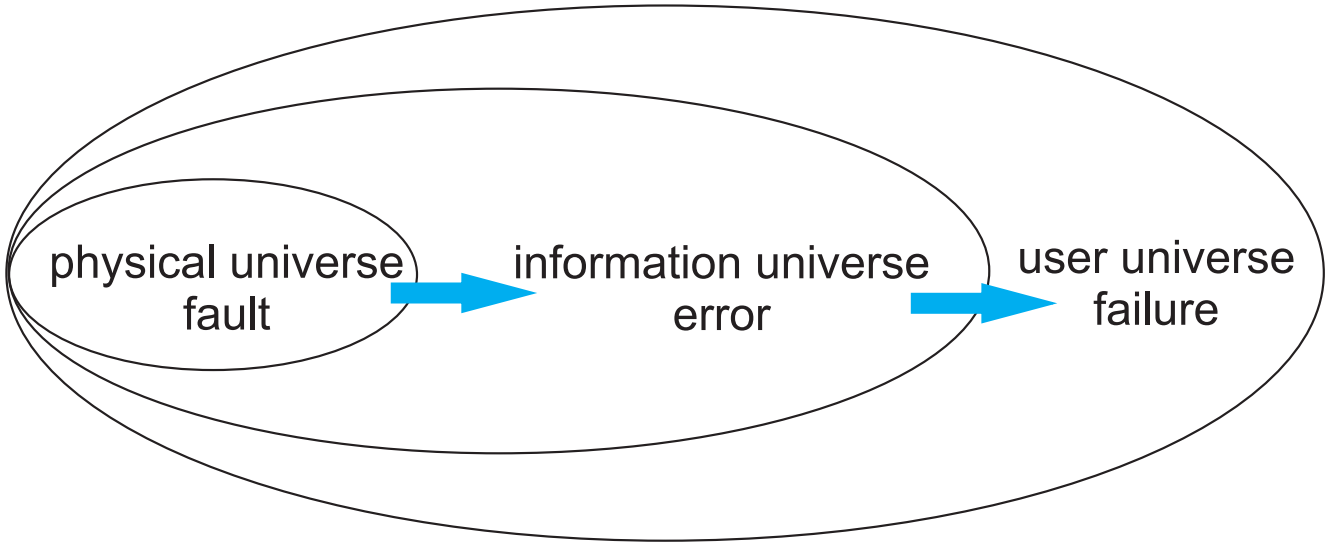
Error

The system state whose processing could lead to a failure is called *error state*

Fault

physical or algorithmic causes of errors

Relation between Faults, Errors and Failures



- Kinds of faults
  1. Permanent: caused by a permanent defect in a computing unit
  2. Transient: occur only during a given time, then disappear (Ex.: faults caused by electromagnetic interference)
  3. Intermittent: transient faults occur repeatedly

Kinds of faults	Description
Server Failure	The server does not work in the specified manner
Omission Failure [1]	There is an omission in responding to an input
Receiving	The server cannot receive incoming messages
Sending	The server cannot send messages
Timing Failure	Server's response is functionally correct, but untimely,
Early	too early
Late	too late
Response Failure	Server responds incorrect
Value	Response value is not correct
State Transition	Server deviates from its correct state

[1] Crash Faults: successive omission faults

- How to solve fault tolerance issues??

- How to solve fault tolerance issues??
- REDUNDANCY!!

- How to solve fault tolerance issues??
- REDUNDANCY!!
  - Hardware (Spatial)
  - Information
  - Temporal
  - Software

- Hardware (Spatial) Redundancy: consists of replicating physical components in the system
  - Static: redundant hardware is always active
  - Dynamic: redundant hardware is activated only in case of faults

- Information Redundancy: add extra bits to information
  - Detect errors
  - Correct errors
  - Very used in files storage and networks
  - Eg.: *Checksum*

- Temporal Redundancy: consists of repeating computation in time
  - Suitable for transient faults
  - Does not need extra hardware or software
  - Increases performance requirements

- Software Redundancy: suitable for dealing with software faults
  - Software components can be executed in parallel or sequentially
  - Depends on available hardware

- In another point of view:
  - Benign: fail-stop failures. Easy to detect and easy to correct
  - Malicious (Byzantine): a component that apparently works correctly

- availability: readiness for correct service.
- reliability: continuity of correct service.
- 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.
- confidentiality: the absence of unauthorized disclosure of information.

Availability

- Property of a system to be immediately ready for use
- Probability that the system is functioning properly at any given time and be available to perform their duties on behalf of their users
- Usually defined in terms of a given time instant

Reliability

- Property of a system able to operate continuously without failure
- It is defined in terms of a time interval rather than a time instant
- Probability that the system is functioning properly at any given time and be available to perform their duties on behalf of their users

Safety

- If a system fails to function properly for a while, nothing catastrophic will happen

Maintainability

- Ease with which a failed system can be repaired
- High capacity systems maintenance can also show a high degree of availability, particularly if the fault can be detected and repaired automatically

- Security is a composite of the attributes of:
  - confidentiality
  - integrity
  - availability, requiring the concurrent existence of
    1. availability for authorized actions only
    2. confidentiality
    3. integrity with "improper"meaning "unauthorized".

Dependability Attributes

- Fault prevention: prevent the occurrence or introduction of faults.
- Fault tolerance: avoid service failures in the presence of faults.
- Fault removal: reduce the number and severity of faults.
- Fault forecasting: estimate the present number, the future incidence, and the likely consequences of faults.

Dependability Techniques

Fault Prevention & Fault Tolerance

Aim at providing the ability to deliver a system function that can be trusted

Fault Removal & Fault Forecasting

Aim at reaching confidence in the ability to deliver a system function that can be trusted. To do so, these techniques justify that both functional and dependability and security specifications are adequate and that the system is likely to meet them.

Why do we study fault-tolerance?

Dependability
- ✓Reliability
- ✓Safety
- ✓Availability
- ✓...

Threats
- ✓Fault
- ✓Error
- ✓Failure

How to achieve?
- ✓ Fault Tolerance
- ✓ ...

Fault Tolerance: why?

1. Critical applications: nuclear plants, medicine, aerospace. A malfunctioning can lead to a catastrophic result. Fault probability must be extremely low
2. Critical Environments: nuclear plants, aerospace, military. Electromagnetic disturbances, particles collision
3. Complex systems: consists of millions of equipments, each one with a give fail probability

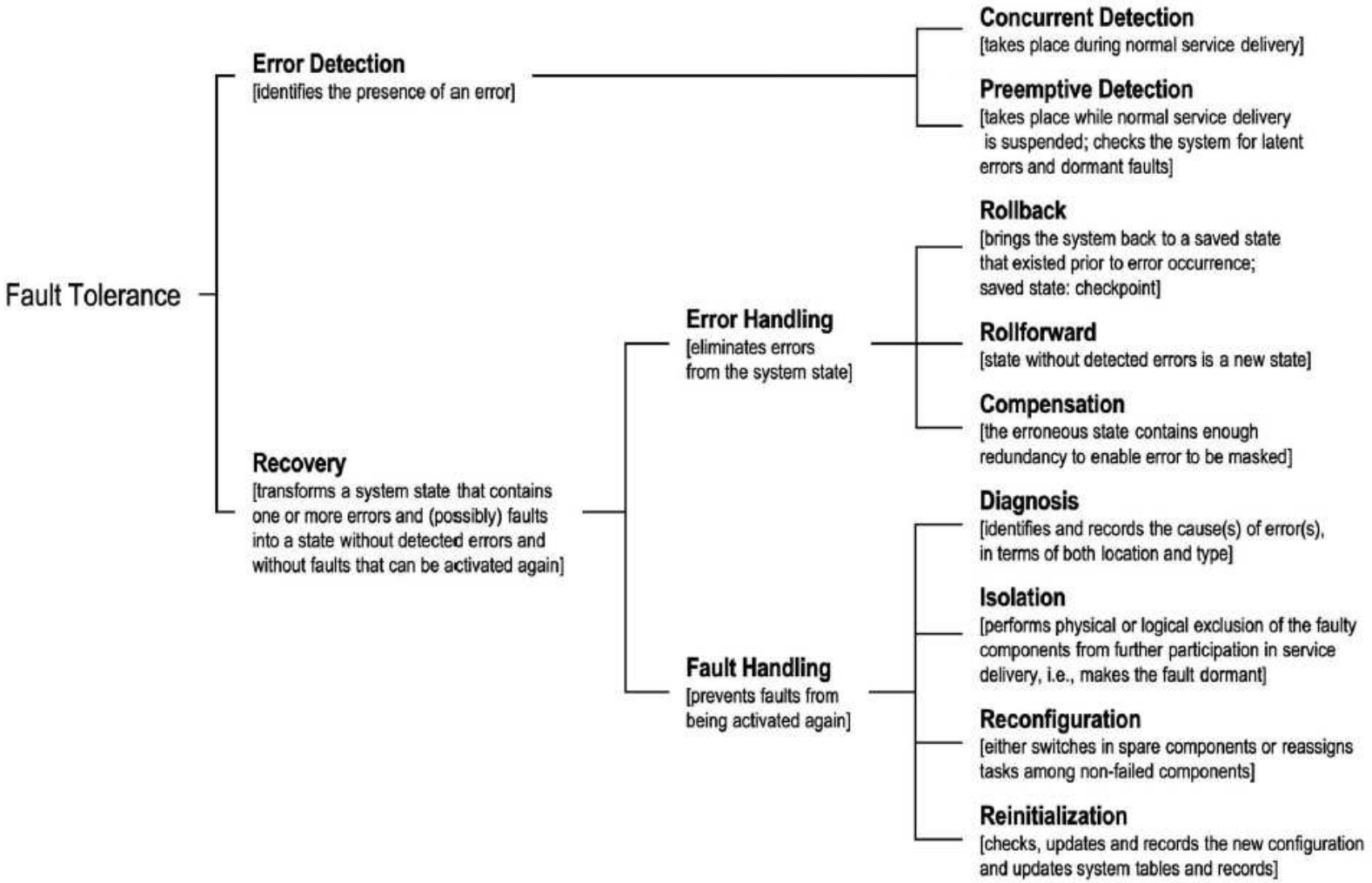
"When a complex system succeeds, that success masks its proximity to failure... Thus, the failure of the Titanic contributed much more to the design of safe ocean liners than would have her success. That is the paradox of engineering and design."

Henry Petroski, **Success through Failure: The Paradox of Design**, Princeton U. Press, 2006, p. 95

Fault Tolerance Techniques

- Fault Tolerance techniques can be of two disjoint classes:
  1. Fault Masking: faults are masked at the origin and does not appear as an error
  2. Error Detection and System Recovery: detects the error, identify the component in which it occurs and recovers the system through time redundancy

Fault Tolerance Techniques



Single Version Software Environment

- Aims at monitoring faults ⇒ limited fault tolerance
- Checks application decision flows to see if it is in accordance with specification
- May be used as software reexecution [2]

[2] Software or temporal redundancy

Multiple Version Software Environment

- Use the same versions of the same software (different implementations)
- Software versions are developed independently
- Several techniques:
  1. Recovery blocks
  2. N-version programming

Multiple Data Representation Environment

- Uses different representations of input data to provide fault tolerance
- May use "*N-copy programming*", which is an implementation of N-version programming focusing on input data diversity

Phase 1 **Error Detection**: Before an error, there is only a latent fault, which cannot be detected

Phase 2 **Confinement**: Due to fault latency, some incorrect data may be propagated in the system. This phase aims at defining the limit in error propagation (depends on project decisions)

Phase 3 **Recovery**: Aims at putting the system into a safe state

Phase 4 **Treatment**: Consists pf identifying the error origin, the fault and to recover the whole system

- Recovery is part of fault tolerance process, which includes:
  1. Error Detection: identify the state in which error occurred
  2. Error Diagnosis: identify error causes and evaluates eventual damages caused by errors
  3. Error Isolation: avoids errors propagation
  4. Error Recovery: changes error state: from **erroneous** to **error free**

Error Recovery Techniques

- Backward Error Recovery
  - System states are stored in specific points (checkpoints)
  - Eventually, rollback to a safe state → assuming that error occurred after last safe state
- Forward Error Recovery
  - Detects the error
  - Evaluates the damages
  - Rebuilds erroneous state

Backward Error Recovery

- Tries to rollback to a safe state
- Based on checkpoints
- Checkpoints must not be affected by errors (low probability)
- Advantages:
  1. Generic approach: does not depend on application
  2. Does not assume previous knowledge about errors
  3. Can deal with faults not detected in implementation or tests
  4. Suitable for transitory errors

Backward Error Recovery

- Disadvantages:
  1. Requires extra resources: storage, time
  2. System must be able to slow down → during recovery
  3. Special attention to domino effects

Forward Error Recovery

- After an error, the system tries to change to a future safe state
- Suitable to be used with redundancy: several redundant units are executed to reach the result
- An specific module judges suitable response
- Advantages:
  1. Does not need extra resources
  2. If faults are known this mechanism is more efficient
  3. If there is a deadline miss, this technique is more suitable than Backward Recovery. Why?

Forward Error Recovery

- Disadvantages:
  1. Application Specific
  2. May only be suitable for expected errors
  3. May no be helpful if error does not behave as expected
  4. May not be very suitable for distributed systems