UNIVERSITÀ
DEGLI STUDI
FIRENZE

Metrics in software verification and validation: some research challenges

A. Fantechi - DINFO
Università di Firenze

Software Dependability

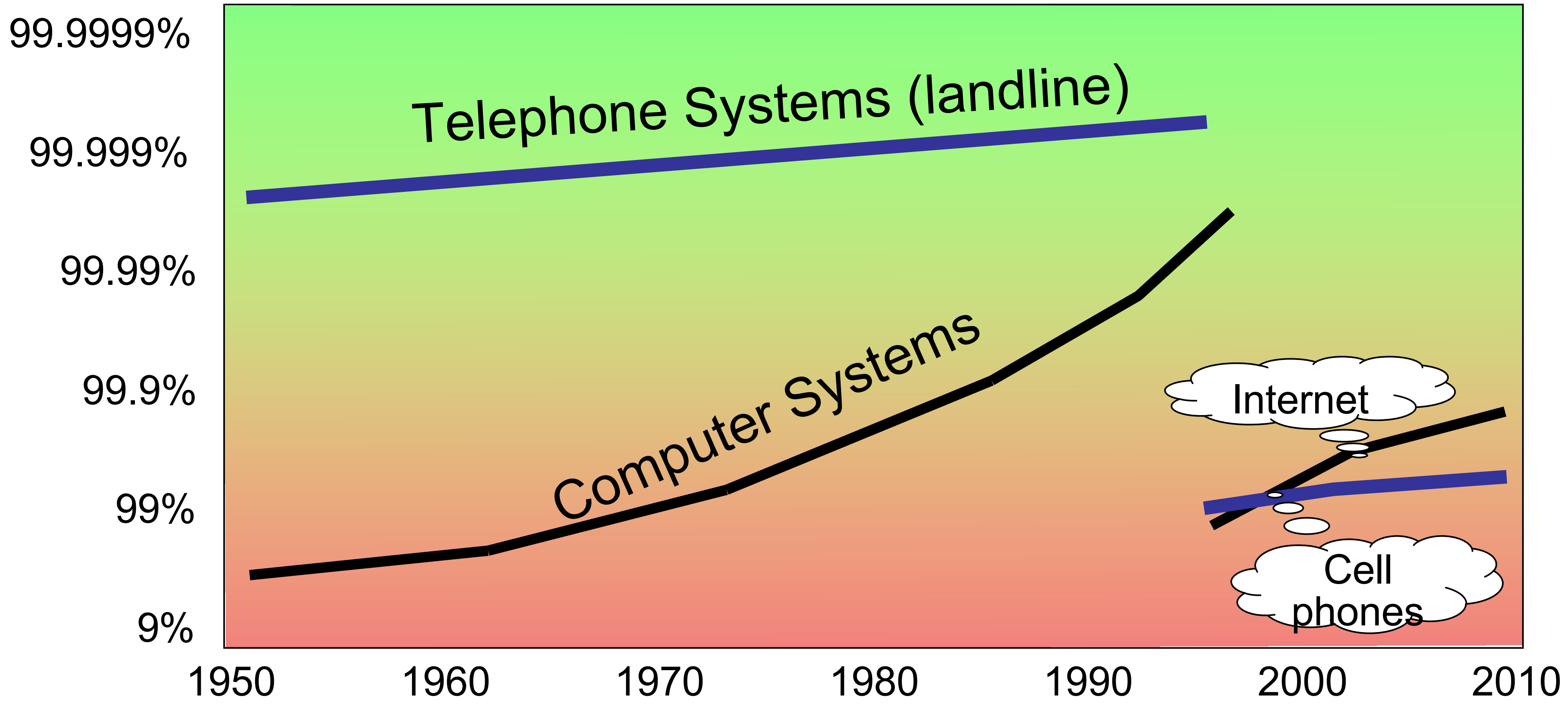
- The complex software applications that pervade nowadays safety critical domains, such as transportation, raise increasing concerns about the ability of software development, verification and validation processes to avoid the presence of software faults in such applications.
- How to measure such ability is a matter of debate, especially since theory and practice tell that perfect complex software does not exists.
- An account of currently applied techniques and of open research challenges will be given.

Software Dependability

- Dependability attributes (quantitative):
  - **Reliability** *(probability of correct behaviour at time t)*
  - ***Availability*** (probability of avaialble service at time t)
  - **Safety** (probability of safe behaviour - or fail safe status - at time t)
  - **Confidentiality/Security** (less common quantitative metrics)

- Usually ***system attributes***
- Established methods for quantifying and predicting dependability attributes at Hardware level

System Availability measures



[From J. Gray, 'Dependability in the Internet era']

Availability	Outage duration/yr
0,999999	32s
0,99999	5min 15s
0,9999	52min 34s
0,999	8h 46min
0,99	3d 16h

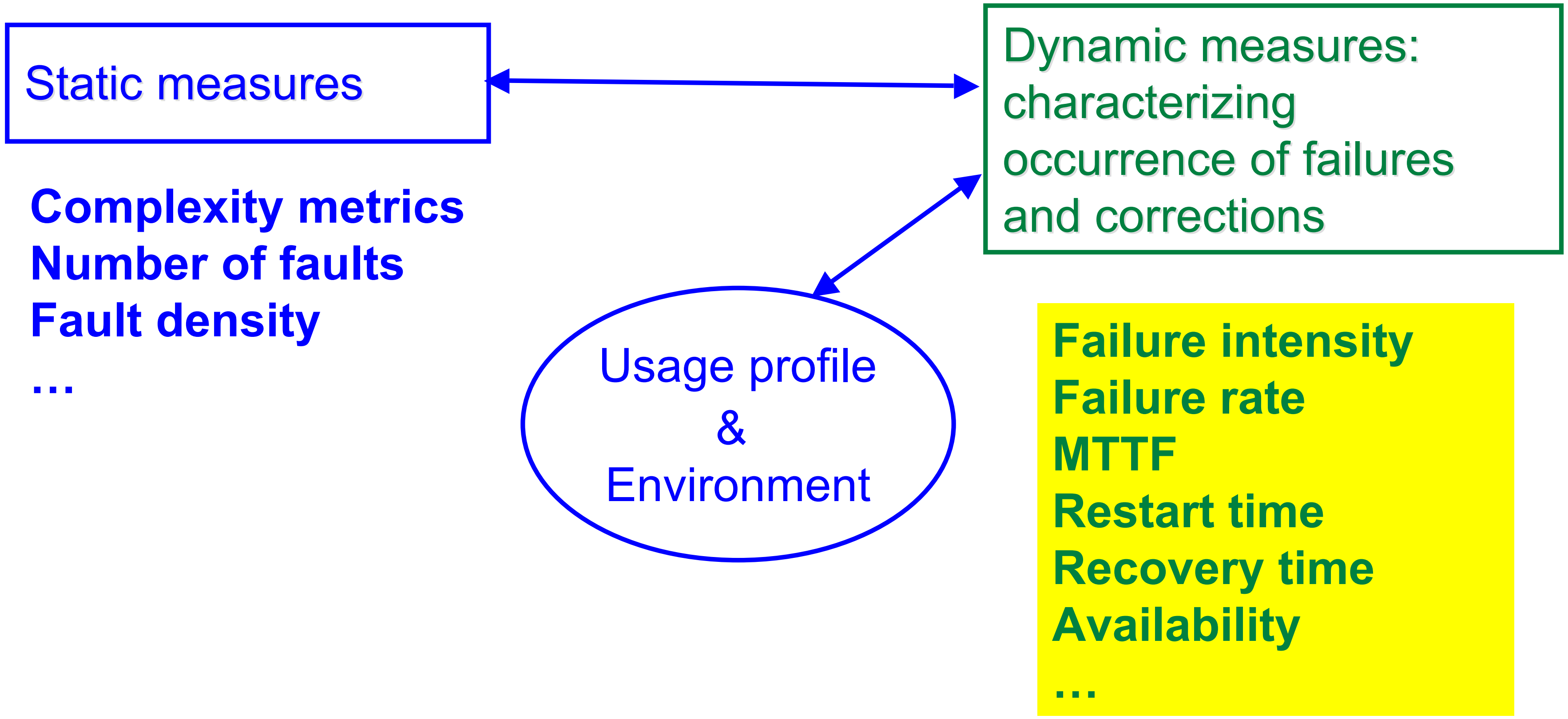
(by Karama Kanoun)

Software Dependability Assessment Difficulties

- Software failures are not random, are deterministic (that is, two identical software components running in the same environment fail at the same time - *see Ariane 5 case*)
- Software failures are not due to consumption phenomena, are design errors
- Software failures are sensitive to actual usage profile (which is an external, not an internal characteristic of software)

- What is software dependability
  - → dependability measures?
    - Number of faults, fault density, complexity?
    - MTTF, failure intensity, failure rate?

How to measure software dependability?



Static measures
Dynamic measures: characterizing occurrence of failures and corrections
Usage profile & Environment

Complexity metrics
Number of faults
Fault density
…

Failure intensity
Failure rate
MTTF
Restart time
Recovery time
Availability
…

(by Karama Kanoun)

Static measures
Software faults metrics

- Number of faults
- **Fault density**

    = number of faults for KLOC

    = number of faults for FP

→ a posteriori measures

→ Prediction possible only under the assumption that *similar* systems show the same fault density.

→ *similar = produced with the same development process ??*

Software faults metrics

Some figures from safety critical domains:

Automotive domain

- 2 .. 3 bugs per KLOC

- Typical car: 100MLOC --> 200Kbugs

Avionics

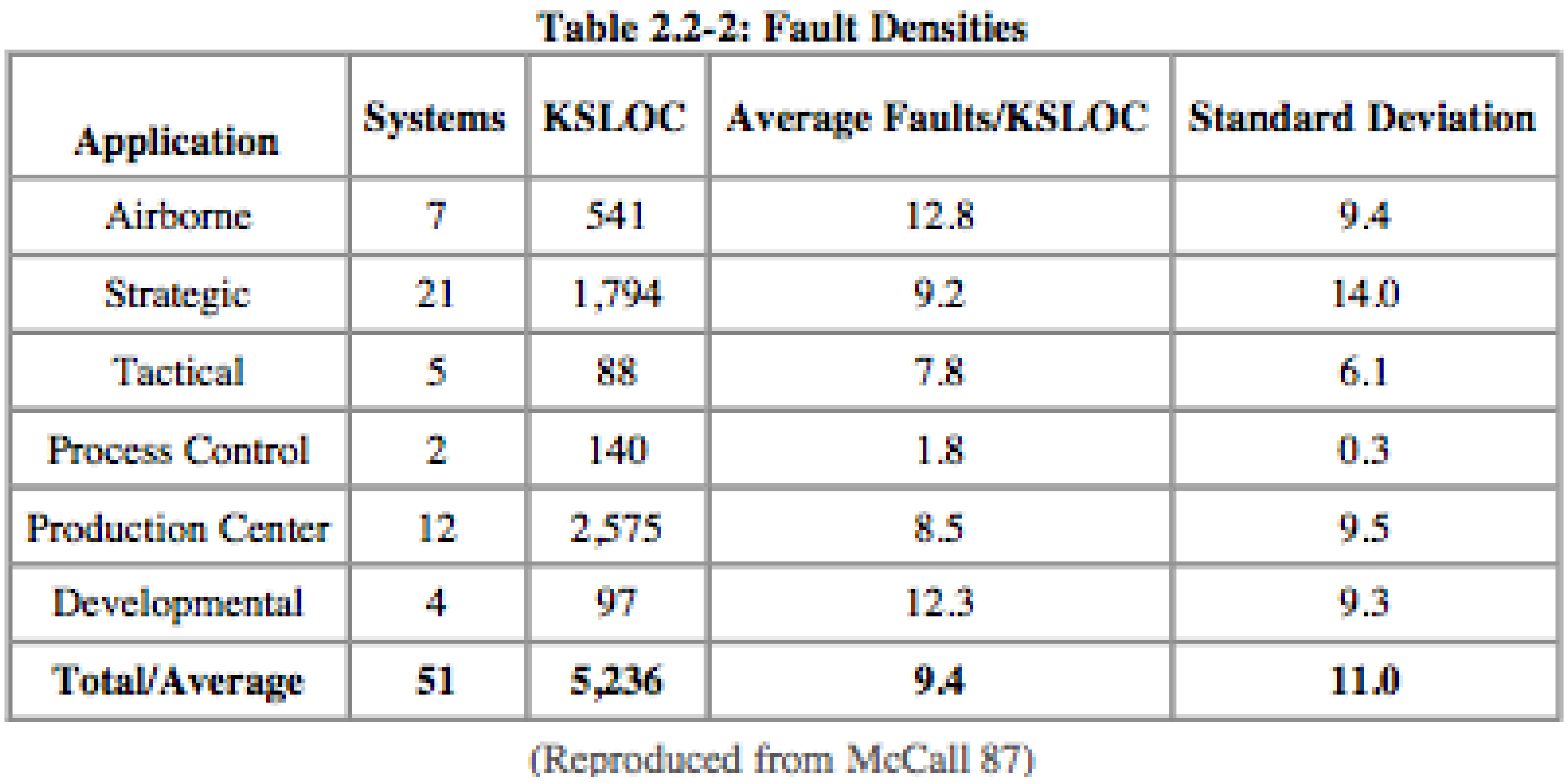


Table 2.2-2: Fault Densities

Application	**Systems**	**KSLOC**	**Average Faults/KSLOC**	**Standard Deviation**
Airborne	7	541	12.8	9.4
Strategic	21	1,794	9.2	14.0
Tactical	5	88	7.8	6.1
Process Control	2	140	1.8	0.3
Production Center	12	2,575	8.5	9.5
Developmental	4	97	12.3	9.3
Total/Average	**51**	**5,236**	**9.4**	**11.0**

(Reproduced from McCall 87)

Railway

How many „Bugs“ do we have to expect?

- Typical production type SW has 1 … 10 bugs per 1.000 lines of code (LOC).
- Very mature, long-term, well proven software: 0,5 bugs per 1.000 LOC
- Highest software quality ever reported :
  - *Less than 1 bug per 10.000 LOC*
  - At cost of more than 1.000 US$ per LoC *(1977)*
  - *US Space Shuttle with 3 m LOC* costing 3b US$ (out of 12b$ total R&D)

  ➔ Cost level not typical for the railway sector (< 100€/LoC)

- Typical ETCS OBU kernel software size is about 100.000 LOC or more
  - That means: 100 … 1.000 undisclosed defects per ETCS OBU
  - Disclosure time of defects can vary between a few days …. thousands of years

Question: What happens to bug counts over the SW life cycle ?

Business vs. safety-critical

	fault density	(at 100 LOC per FP)	
	(Fault per KLOC)	(Fault per FP)	
Automotive	**3**	0,03	safety-critical
Aviation, av.	**9,7**	0,097	
Typical low	**1**	0,01	
Typical high	**10**	0,1	
Mature	**0,5**	0,005	
Shuttle	**0,1**	0,001	
Agile 1	1,28	**0,0128**	Business
Agile 2	4,8	**0,048**	
Agile 3	22	**0,22**	
Traditional, best	200	**2**	
Traditional, average	450	**4,5**	

Business ref.: www.leanagilepartners.com/publications.html

Lowering fault density

- Improving the development process helps to lower fault density
- Software Process Improvements
  - **Motorola** (Arlington Heights), mix of methods:

    Fault density reduction = 50% within 3.5 years
  - **Raytheon** (Electronic Systems), CMM:

    Rework cost divided by 2 after two years of experience

    Productivity increase = 190%

    Product quality: multiplied by 4
- Tradeoff between development (and verification) costs w.r.t. fault density reduction

Lowering fault density *safety-critical domain*

- Responsibility given to developers, verifiers, assessors,…
- No one certifies that a given product has a fault density lower than any given figure
- Certification that safety guidelines have been accurately followed
- Safety guidelines demand for highy structured development and verification processes
- Typically, verification costs higher than development costs
- Advanced techniques for development and specification may lower the costs of following guidelines (Model based development, Formal Methods, etc…)
- In particular, formal verification is in principle in many cases, and in practice in some cases, able to produce zero-defects software

Dynamic measures
Software reliability

- Indeed, safety certification of hardware admits probabilistic quantification of reliability/safety

$$\boxed{R(t) = e^{-\lambda t}}$$

- MTTF – Mean Time To Failure

  $MTTF = 1/ \lambda$

Established methods to estimate reliability of hardware components and assemblies allow to compute reliability figures that are used for safety certification.

Increasing interest in safety-critical systems community to have a similar probabilistic quantification of software reliability as well, for a uniform reasoning about hardware-software systems (and systems of systems)

Dynamic measures
Software reliability

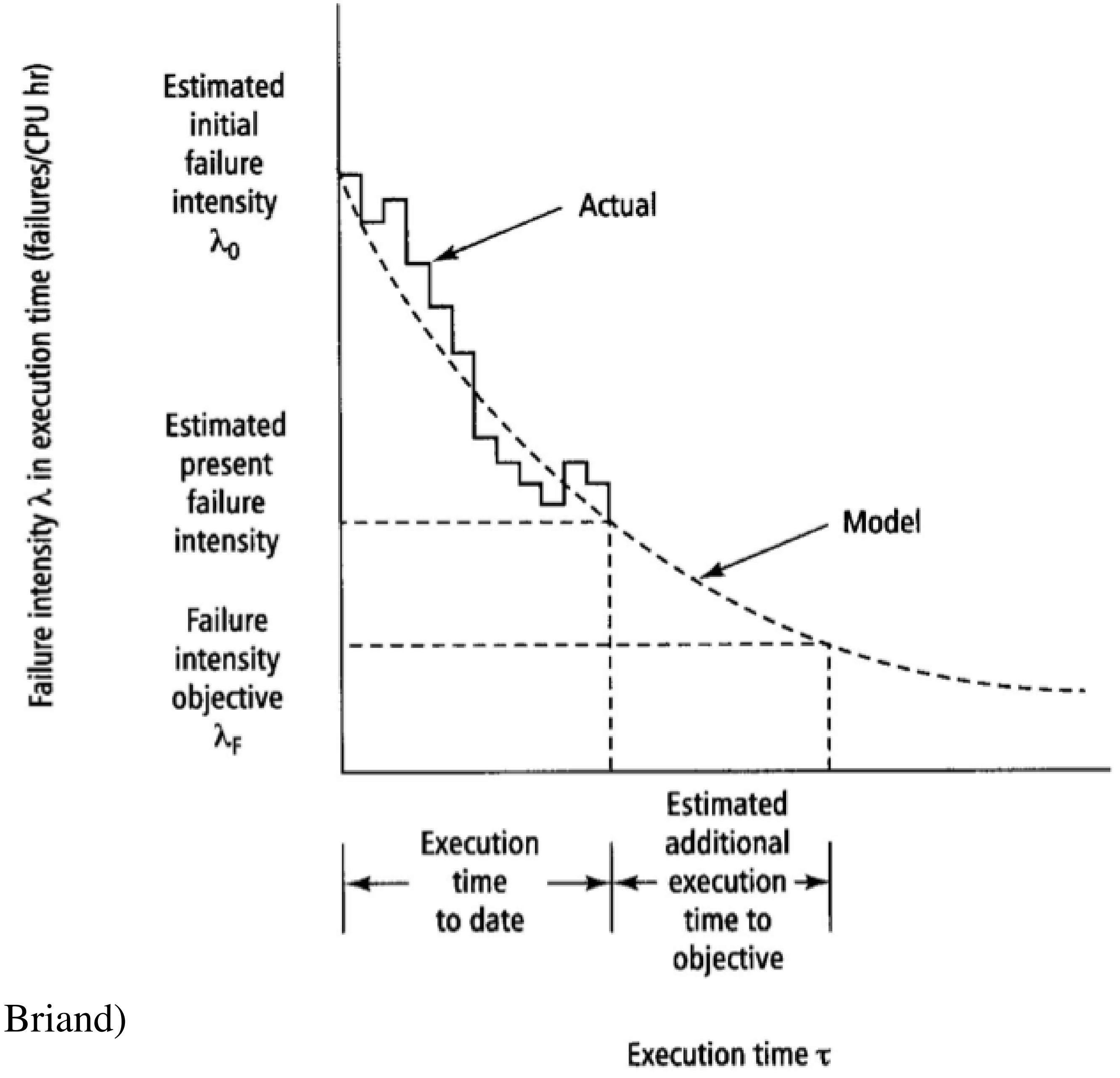
- **Software Reliability:** The probability of failure-free software operation for a time-period
- **Software reliability prediction:** Determines future reliability based upon software metrics data collected (available).
- **Cumulative Failure Function:** The mean cumulative failures at each point in time;
- **Failure Intensity Function:** The rate of change of the cumulative failure function (aka *failure rate function*)
- **Mean Time to Failure (MTTF):** The expected mean time that the next failure will occur;

Software reliability models

- Collection of data about past software failures (both in the testing phase and in the operational life)

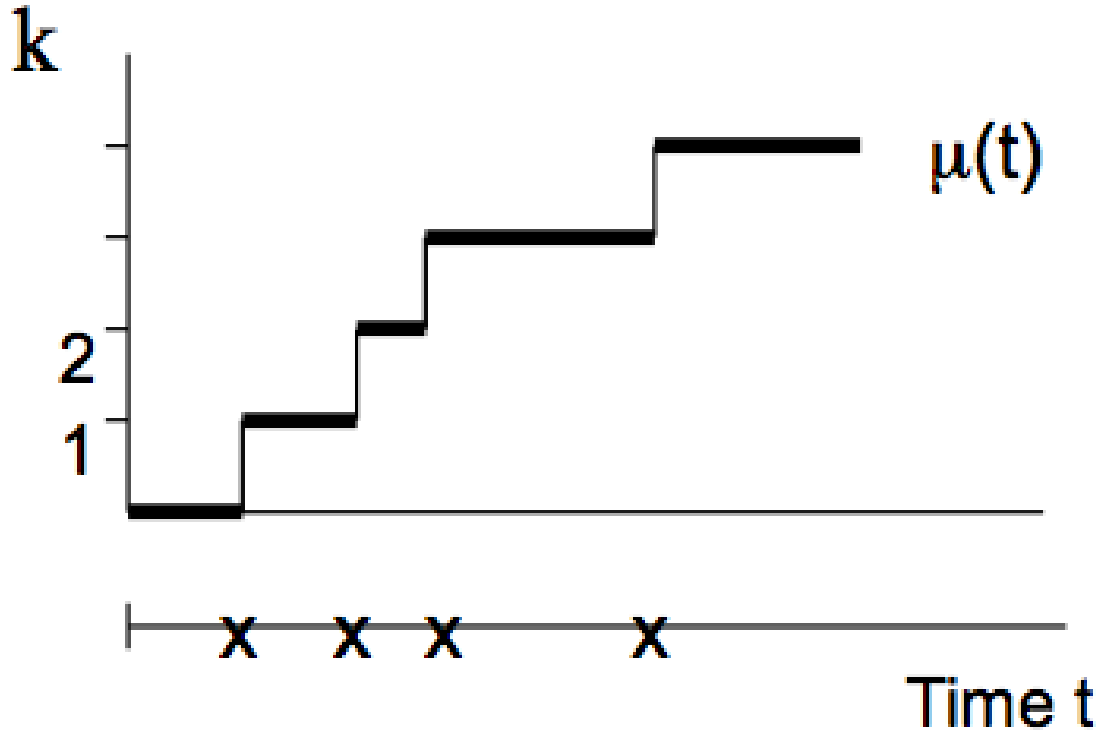
- Using a model to predict failure intensity, MTTF, or residual failure number on the basis of collected data

Conceptual View of Prediction



(by Lionel Briand)

Cumulative failures



Increasing TTF --> Reliability growth

(by Lionel Briand)

Software reliability models

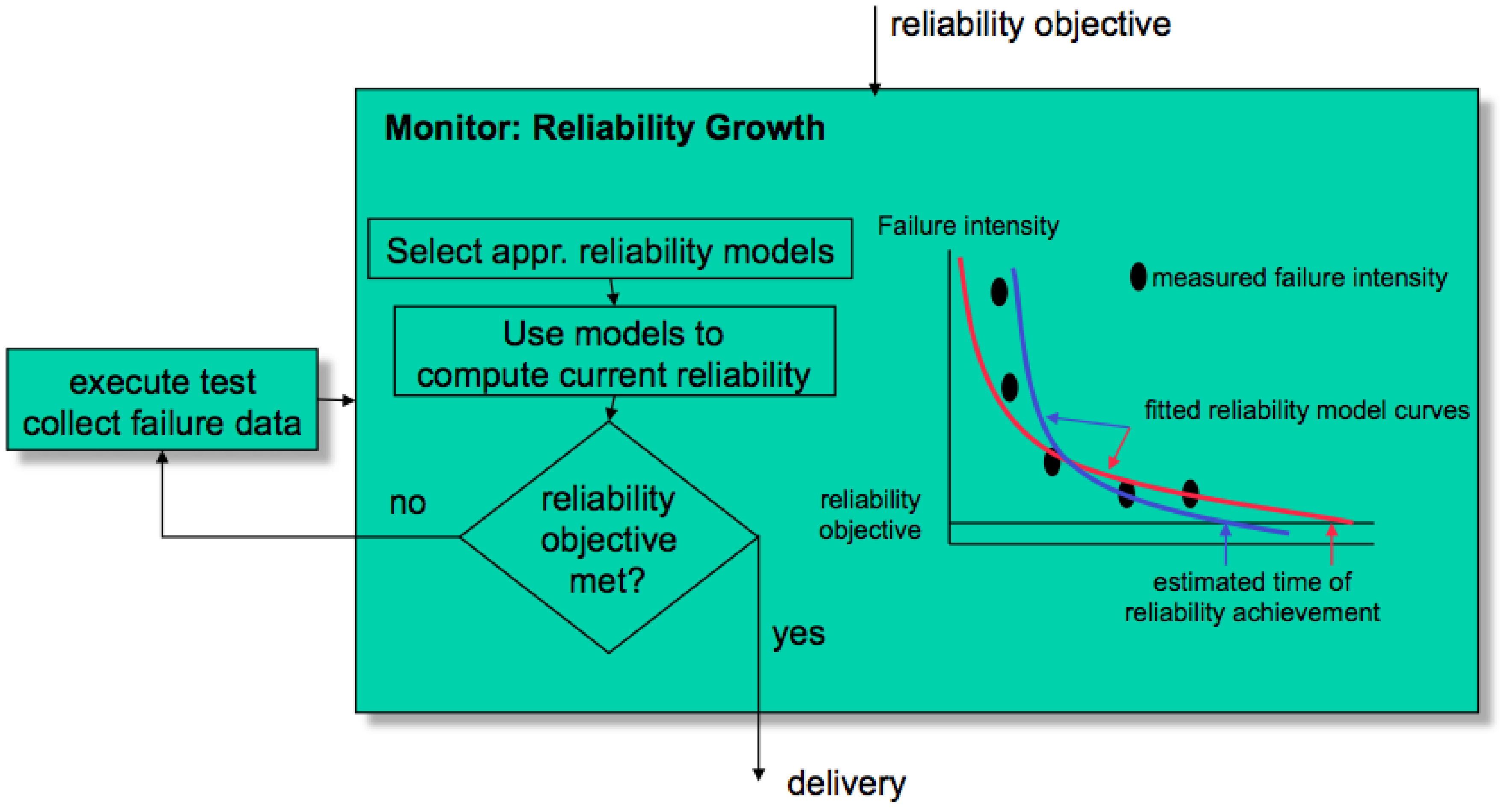
- Assumptions
  - At time 0 there are N faults
  - Every fault is independent from the others and hass the same probability of causing a failure during the testing phase
  - Every fault detected duriing testing is removed in null time
- Jelinski-Moranda model
  - Failure intensity at a given time is proportional to the number of residual faults at that time:

$$Z(t_i) = \theta * [N - (i - 1)]$$

- Musa model
  - Failure intensity depending on execution time:

$$Z(t) = \phi * f * (N - n_t)$$

… criteria for stopping testing


reliability objective
Monitor: Reliability Growth
Select appr. reliability models
Use models to compute current reliability
execute test collect failure data
no
reliability objective met?
yes
delivery
Failure intensity
measured failure intensity
fitted reliability model curves
reliability objective
estimated time of reliability achievement

(by Lionel Briand)

Software Reliability Models

- Many Models exist (ca. 40 published)
- No model can be considered the best.
- Different Assumptions of Models:
  - Definition of testing process
  - Finite or infinite number of failures?
  - No faults introduced during debugging?
  - Distribution of data (Poisson, Binomial)?
  - Data requirements? (inter-failure data, failure count data)
- Assessing the goodness-of-fit
  - Kolmogorow-Smirnov, Chi-Square

- Usually, Reliability Models assume *reliability growth*.

Pro's and Con's

Pro's

- Reliability can be specified
- Objective and direct assessment of reliability
- Prediction of time to delivery

Con's

- Usage model may be difficult to devise
- Selection of reliability growth model difficult
- Measurement of time crucial but may be difficult in some contexts
- Not easily applicable to safety-critical systems as very high reliability requirements would take years of testing to verify.

Ultra High Reliability / Safety Prediction

It is essential to consider actual achievability and testability when predicting reliability for software systems that must be relatively high.

Ultra high reliability demands can be not testable or demonstrable.

For example, if the demand for the failure rate is 10^{-4} FPH (~1 year MTTF) then there must be sufficient resources for extensive validation and verification to demonstrate this level.

The current state of the art is limited in providing any help in assessing the software reliability at this level.

Safety critical Systems often require reliability figures of 10^{-7} to 10^{-9} FPH

Software reliability - DO178C

Software Reliability Models:

Many methods for predicting software reliability based on developmental metrics have been published … This document does not provide guidance for those types of methods, because at the time of writing, currently available methods did not provide results in which confidence can be placed.

Conclusions

- Although used to measure process improvements and productivity, software dependability metrics do not appear yet suitable for safety critical systems certification (while hardware dependability measures are)
- "Absolute" safety (zero-defects) sought with advanced development and verification technologies
- Though, accurate quantification of probability of software failures is more and more required, to match the estimation techniques avialable for hardware.
- Still an open research challenge…