Software Reliability

Software Reliability

What is software reliability?

the probability of failure-free software operation for a **specified period of time** in a **specified environment**

input
sw
output

We assume that programs will not be fault-free

One of the weakest links in *systems reliability* is **software reliability**. Even for control applications which usually have less complex software, it is well established that many failures are results of software bugs.

Software Reliability

Software is subject to

input
sw
output

1. design flaws

- mistakes in the interpretation of the specification that the software is supposed to satisfy (ambiguities)

- mistakes in the implementation of the specification: carelessness or incompetence in writing code, or inadequate testing

2. operational faults
   incorrect or unexpected usage faults
   - operational profile

operational profile: a set of alternatives of system operational scenarios and their associated probabilities of occurrence

Design Faults

- hard to visualize, classify, detect, and correct

- closely related to human factors and the design process

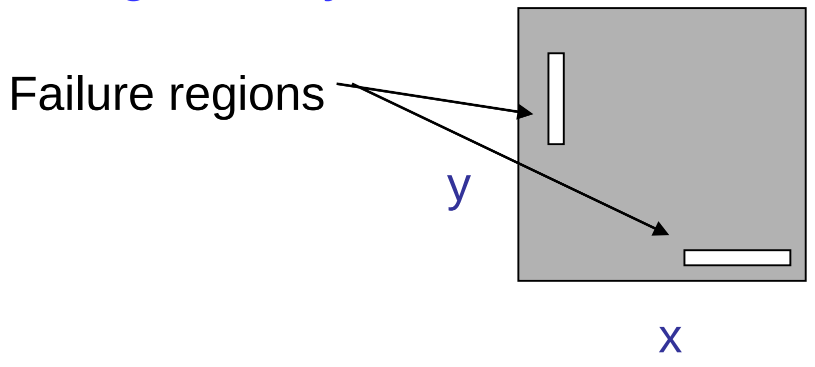
- a design flaw not discovered and corrected during testing, may possibly lead to a failure during system operation

Given a design flaw, only some type of inputs will exercise that fault to cause failures. Number of failures depend on how often these inputs exercise the sw flaw

Apparent reliability of a piece of software is correlated to how frequently design faults are exercised as opposed to number of design faults present!!!!!

Software faults and Failure regions

The input to the software is a set of variables, defining a Cartesian space, e.g. x and y

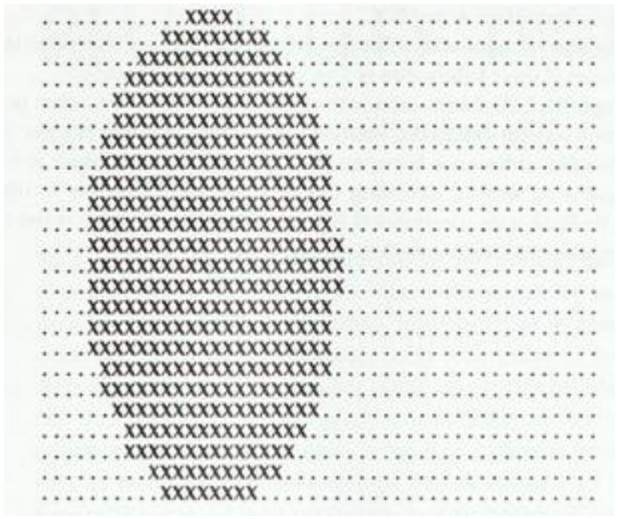


The software contains bugs if some inputs are processed erroneously

(efficacy of software fault tolerance techniques depends on how disjoint the failure regions of the versions are)

Faults tends to produce errors that are grouped together

Points in the input space that cause a *fault* to produce errors can tend to cluster and form regions called *error crystals* [Results of software error-data experiments, Finelli, NASA Langley Research Center, 1998]



x: an input value that caused a single fault to produce an erroneous output
. : inputs that produced correct outputs

These regions are a particular concern in real-time applications where the input variables may be slowly varying and thus triggering multiple failures because of a single fault

Error rates for faults in two programs

NASA studies observed widely varying error rates for the faults identified

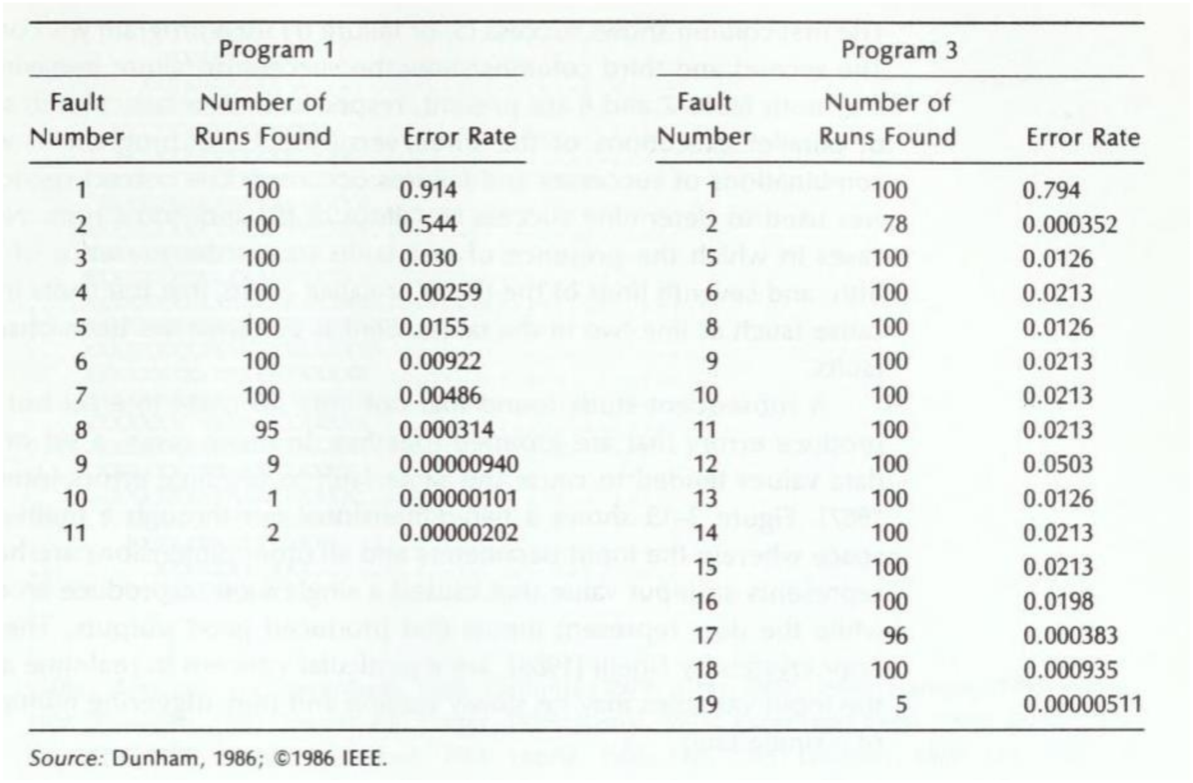
An experiment in software reliability:Life-critical applictions, Dunham J.R, Trans. On Soft. Engineering,1986

Three versions of a lunch interceptor condition were generated: 11 faults discovered in the first version, 1 fault in the second version, and 19 faults in the third version

Error rate:
the frequency of erroneous outputs

15.250.000 program executions

The error rate for individual faults varies over several order of magnitude



Program 1			Program 3		
Fault Number	Number of Runs Found	Error Rate	Fault Number	Number of Runs Found	Error Rate
1	100	0.914	1	100	0.794
2	100	0.544	2	78	0.000352
3	100	0.030	5	100	0.0126
4	100	0.00259	6	100	0.0213
5	100	0.0155	8	100	0.0126
6	100	0.00922	9	100	0.0213
7	100	0.00486	10	100	0.0213
8	95	0.000314	11	100	0.0213
9	9	0.00000940	12	100	0.0503
10	1	0.00000101	13	100	0.0126
11	2	0.00000202	14	100	0.0213
			15	100	0.0213
			16	100	0.0198
			17	96	0.000383
			18	100	0.000935
			19	5	0.00000511

Source: Dunham, 1986; ©1986 IEEE.

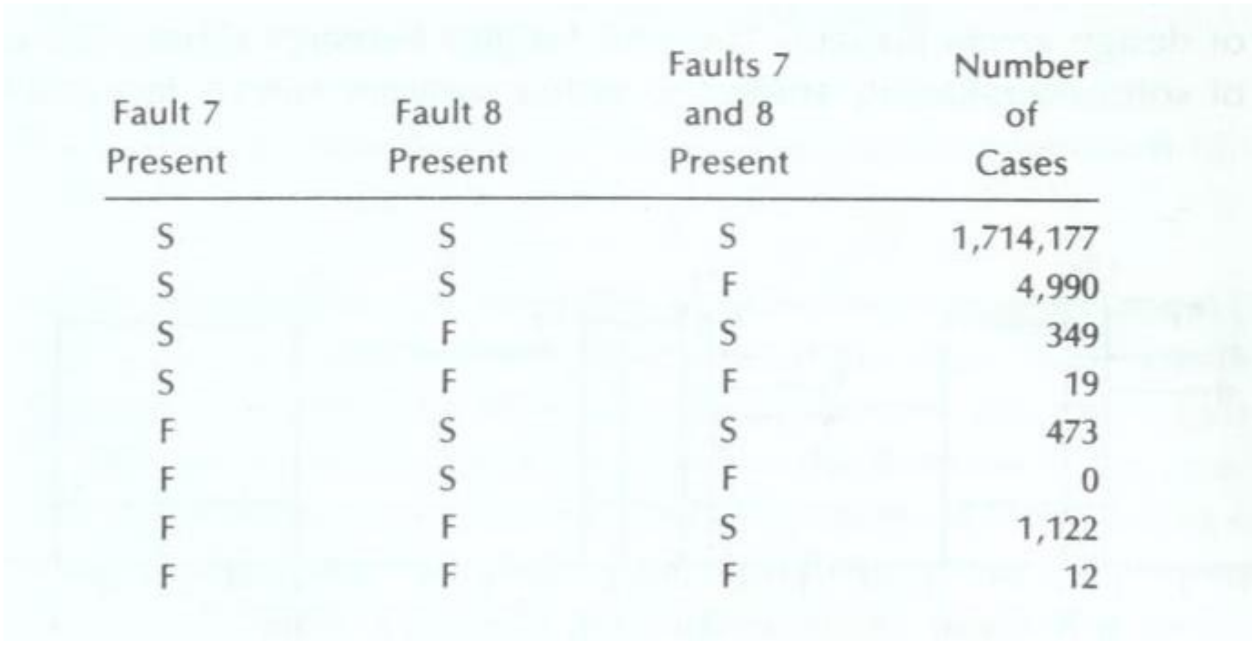
Many software reliability models assume that faults contribute equally to the rate at which a program generate erroneous outputs

Interaction between faults

Interaction between faults: sometimes the probability of failures is increased and at other time faults are masked

[An experiment in software reliability, Dunham, NASA Langley Research Center, 1986]

interaction between two different faults: 7 and 8



Fault 7 Present	Fault 8 Present	Faults 7 and 8 Present	Number of Cases
S	S	S	1,714,177
S	S	F	4,990
S	F	S	349
S	F	F	19
F	S	S	473
F	S	F	0
F	F	S	1,122
F	F	F	12

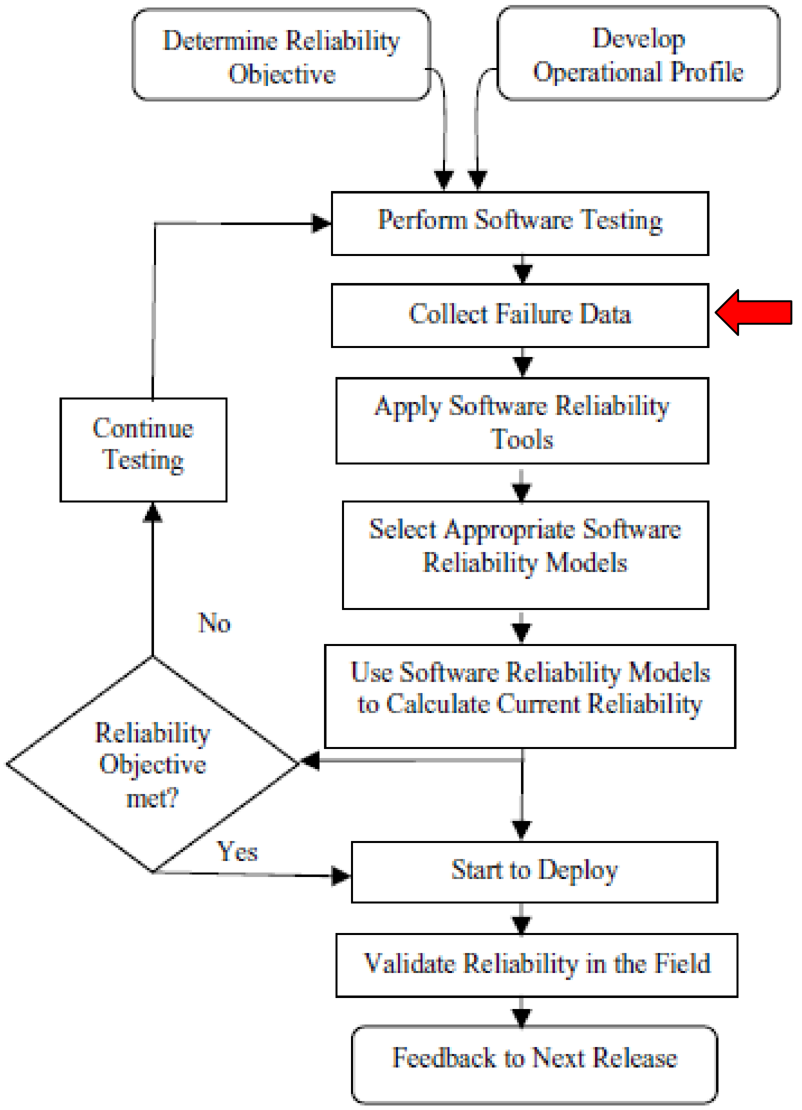
S: success of the program
F: failure of the program
Last column: number of parallel executions of the three versions of the program

Code coverage

- Code coverage, a metric used by code testers, indicating how completely a test set executes a software system, influences the reliability measure

- Several models have been proposed to determine the relationship between the number of faults/failures and the test coverage achieved with various distributions

Software reliability process overview


Determine Reliability Objective
Develop Operational Profile
Perform Software Testing
Collect Failure Data
Apply Software Reliability Tools
Select Appropriate Software Reliability Models
Use Software Reliability Models to Calculate Current Reliability
Continue Testing
No
Reliability Objective met?
Yes
Start to Deploy
Validate Reliability in the Field
Feedback to Next Release

Collection of data

Data to be collected

☞ Background information
  - Product itself: software size, language, functions, current version, workload
  - Usage environment: verification and validation methods, tools, etc.

☞ Data relative to failures and corrections
  - Date of occurrence, nature of failures, consequences
  - Type of faults, fault location

☞ Usually, recorded through
  - Failure Reports (FR)
  - Correction Reports (CR)

☞ Failure Report (FR)

Required Information

- Serial number (for identification)
- Report editor
- Product reference, version affected (or prototype)
- Date and time of failure occurrence

Desirable Information

- Failure occurrence condition
- Failure criticality or consequences
- Affected function or task
- Action proposed (if any)

☞ Correction Report (CR)

Required information

- Serial number (for identification)
- Report editor
- Date of correction
- Correction nature
- Product reference
- Reference to the FR

Desirable Information

- Identification of the modified components

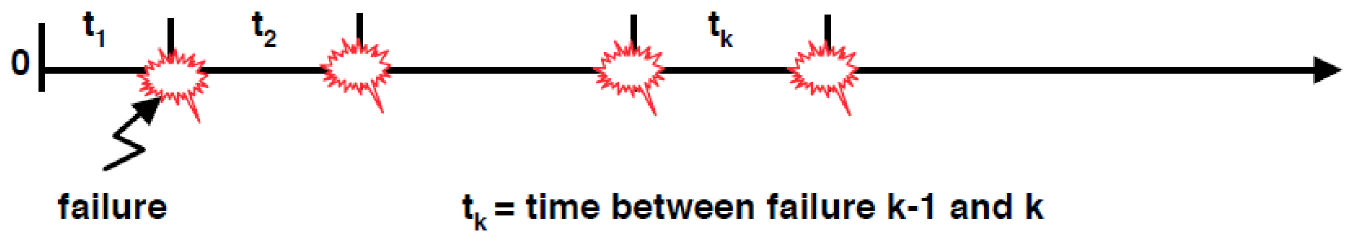
☞ Integration with already existing data collection programs

☞ Importance of training

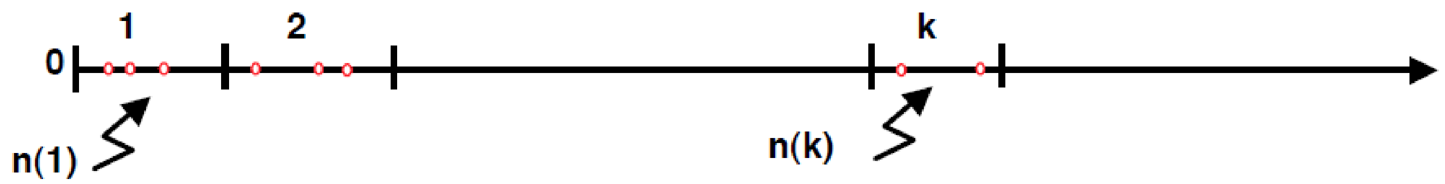
Data pre-processing for reliability analysis

☞ **Two kinds of data sets can be extracted from FRs and CRs**

- Time to failures (or between failures)

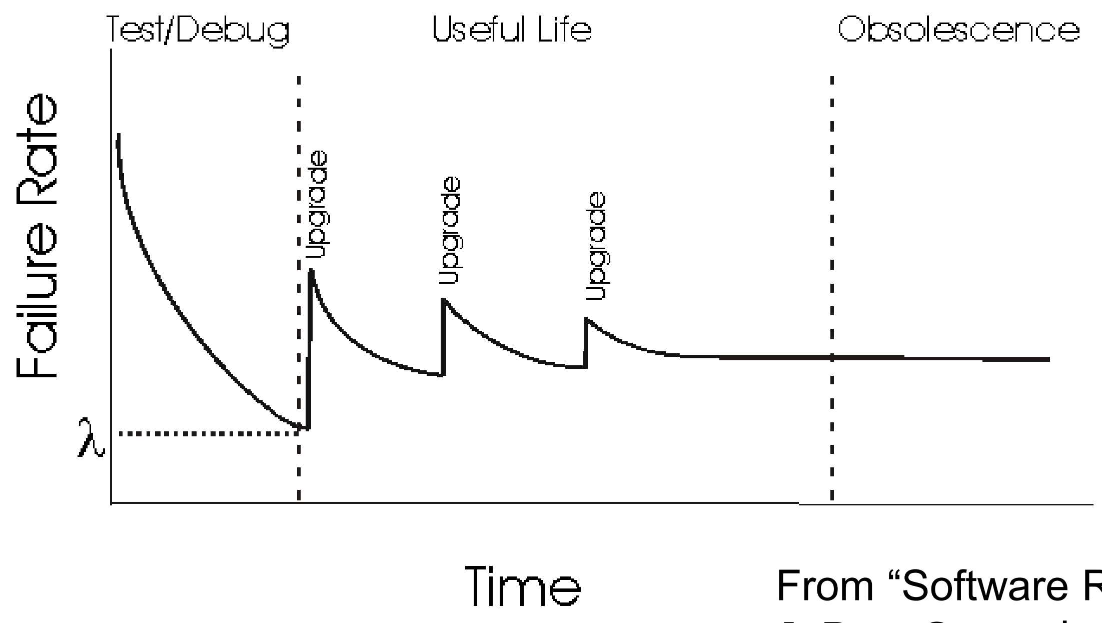


- Grouped data
  - ☞ Number of failures per unit of time, n(k)
  - ☞ Cumulative number of failures N(k)



SOFTWARE RELIABILITY EVOLUTION

As a software is used, design faults are discovered and corrected. Consequently, the reliability should improve, and the failure rate should decrease BUT corrections could cause new faults



upgrades imply feature upgrades, not upgrades for reliability.

From “Software Reliability”,
J. Pan, Carnegie Mellon University, 1999

identify periods of reliability growth and decrease

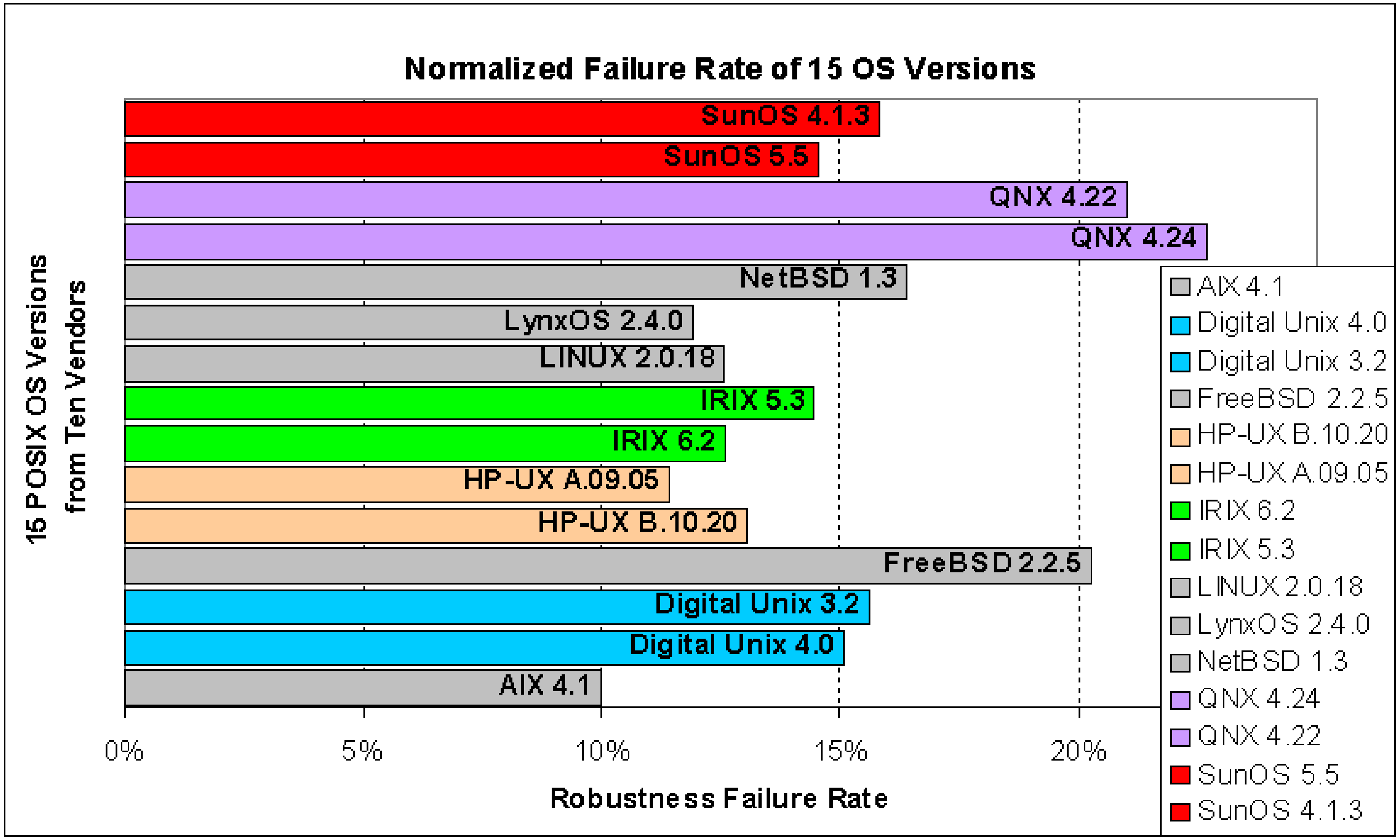
SOFTWARE RELIABILTY EVOLUTION

- in the useful-life phase, software will experience a drastic increase in failure rate each time an upgrade is made. The failure rate levels off gradually, partly because of the defects found and fixed after the upgrades.

- Even bug fixes may be a reason for more software failures, if the bug fix induces other defects into software

- in the last phase, software does not have an increasing failure rate as hardware does. In this phase, software is approaching obsolescence; there are no motivations for any upgrades or changes to the software. Therefore, the failure rate will not change.

Sometimes redesign or reimplementation of some modules with better engineering approaches in a new version of the product



From “Software Reliability”, J. Pan, Carnegie Mellon University, 1999

Software Reliability models

There are basically two types of software reliability models:

1) "defect density" models

attempt to predict software reliability from **design parameters**
use code characteristics such as code complexity in terms of lines of code, number of operators, nesting loops,
number of input/output, the software development process, etc

2) "software reliability growth" models

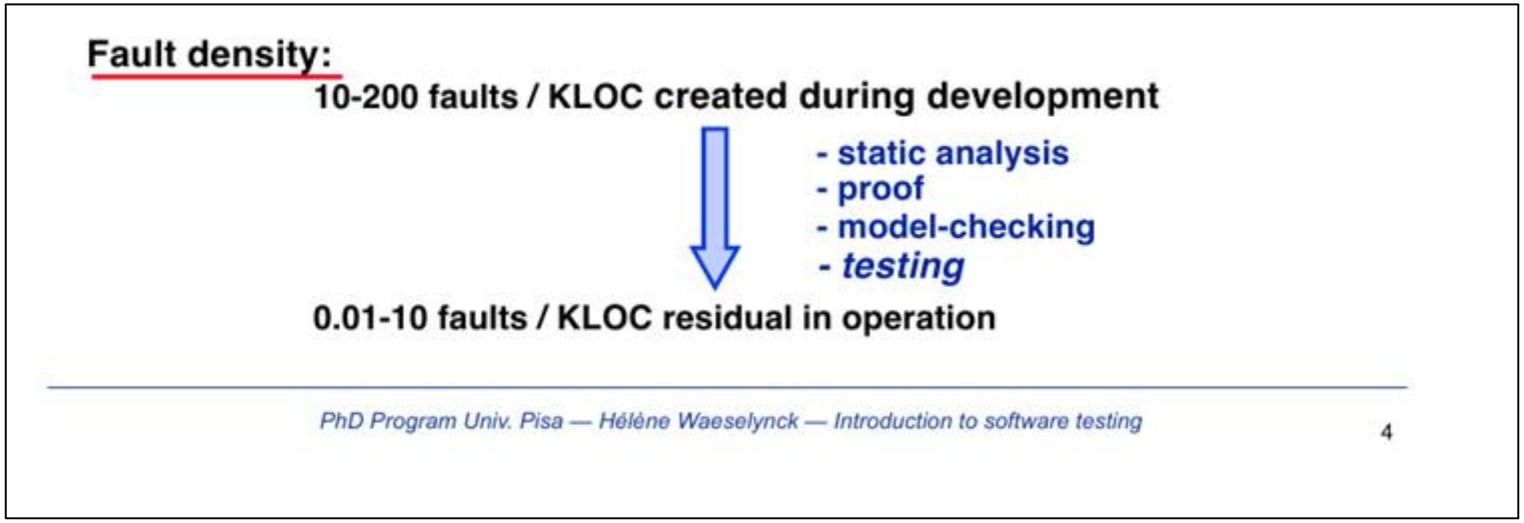
- attempts to predict software reliability from **test data**
- statistically correlates failure detection data with known probability distributions

Defect density models

Fault density: number of faults for KLOC (thousands of lines of code)

Fault density ranges from 10 to 50 for "good" software and from 1 to 5 after intensive testing using automated tools

[Miller 1981]
Miller E.F, et al. "Application of structural quality standards to Software", Softw. Eng. Standard Appl. Workshop, IEEE, 1981



Fault density:

10-200 faults / KLOC created during development

- static analysis
- proof
- model-checking
- *testing*

0.01-10 faults / KLOC residual in operation

PhD Program Univ. Pisa — Hélène Waeselynck — Introduction to software testing 4

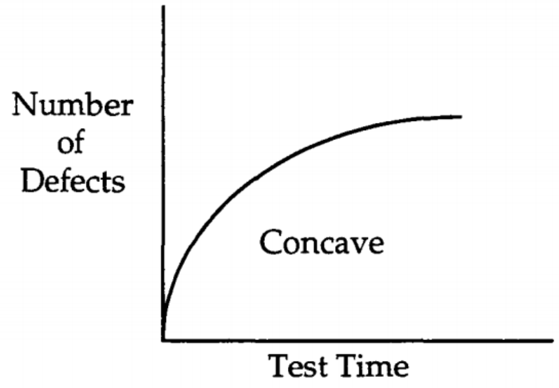
Software Reliability Growth Models

Based on the idea of an iterative improvement process of software. Software is tested, the times between successive failures are recorded, and faults are removed.

testing -> correction ->testing

Based on the assumption that the failure rate is proportional to the number of bugs in the code.

Each time a bug is repaired, there are fewer total bugs in the code, the failure rate decreases as the number of faults detected (and removed) increases, and the total number of faults detected asymptotically approaches a finite value.



Software Reliability Growth Models

Software failures are **random events**, because they are result of two processes:

- the introduction of faults
- and then activation through selection of input values

These processes are random in nature:

- we do not know which bugs are in the software

- we do not know when inputs will activate those bugs

Software reliability growth models are developed in general by probability distribution of failure times

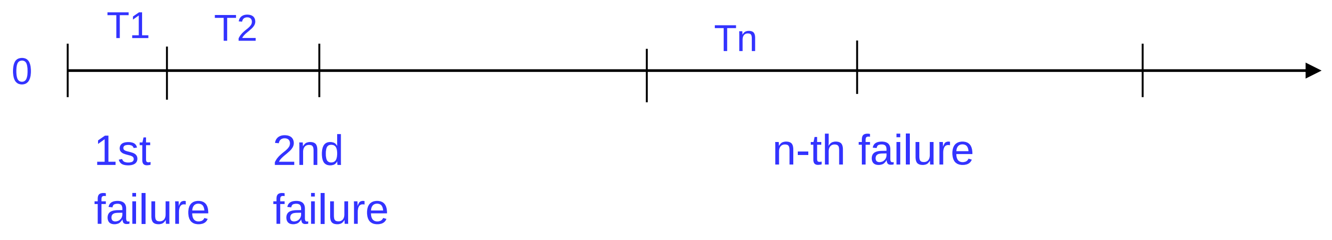
Reliability growth characterization

continuous time reliability growth

Assume times between successive failures are modeled by random variables T1, ..., Tn

T1, time to the first failure

Ti, i>1, time between failure i-1 and failure i



Reliability growth: Ti $<=_{st}$ Tk for all i < k

Prob {Ti < x} >= Prob {Tk <= x} forall i < k and for all x

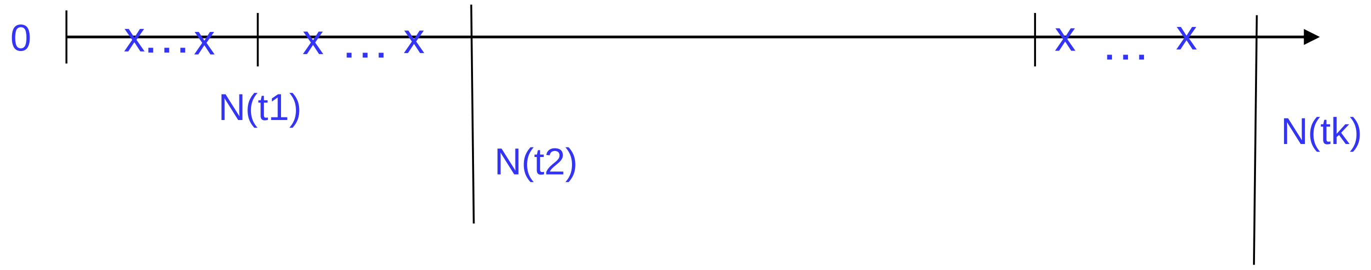
Reliability growth characterization

Number of failures: the number of failures is decreasing

Cumulative number of failure law:

the number of failure events in an interval of the form [0, tk]
is larger than the number of failure events taking place in an interval
of the same length beginning later

Random Variables N(t1), ..., N(tn)
N(ti) = cumulative number of failures between 0 and ti



RELIABILITY GROWTH MODELS

☞ Failure rate models
(Failure rate equations & relationship between successive failure rates)

- Deterministic, piecewise Poisson Process models: Jelinski Moranda, Musa
- Stochastic, doubly stochastic process model: Littlewood-Verrall

☞ Failure intensity models: succession of failures
(based on Non-Homogeneous Poisson Process (NHPP))

- Exponential model (Goel Okumoto)
- Hyperexponential model (Kanoun-Laprie)
- S-Shaped model (Yamada et al)

Jelinski Moranda model: equations

- Parameters

  N_0 = total number of faults

  Φ = fault manifestation rate

  $\lambda(i)$ = failure rate of the i-th failure

  Ti = random variable: time between failures i-1 and i (observation = ti)

- Relations

  $\lambda(i) = \Phi\,[N_0 - (i - 1)] = di / dt \qquad i = 1, 2, \ldots, N_0$

  Prob. $(Ti < ti) = \Phi\,(N_0 - i + 1].\ \exp\{\,\Phi\,(N_0 - i + 1).ti\}$

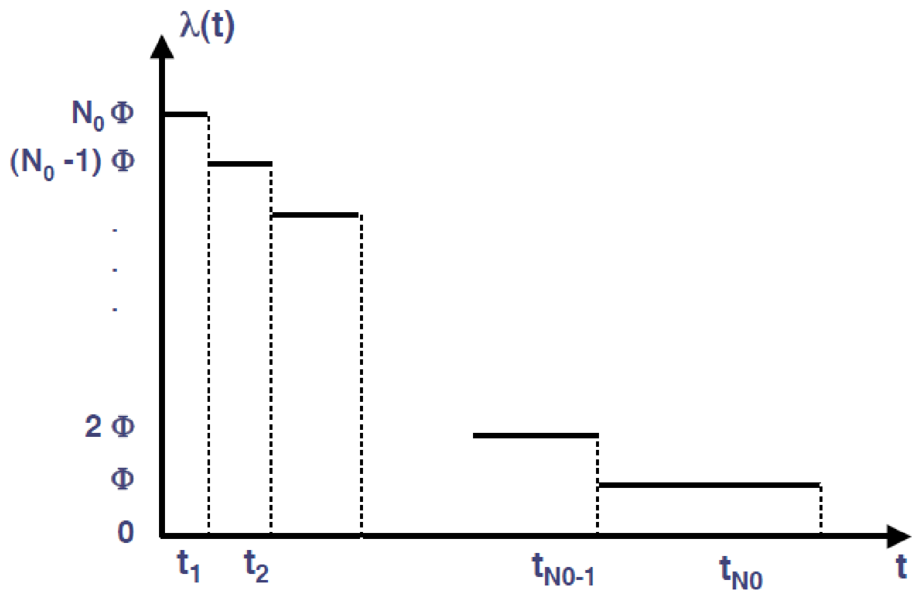
  $$MTTF_i = \frac{1}{\lambda(i)} = \frac{1}{\Phi\,[N_0 - (i - 1)]} \longleftarrow$$

  $N(t) = N_0\,[\,1 - \exp(-\Phi\, t)\,]$ = number of faults detected at t

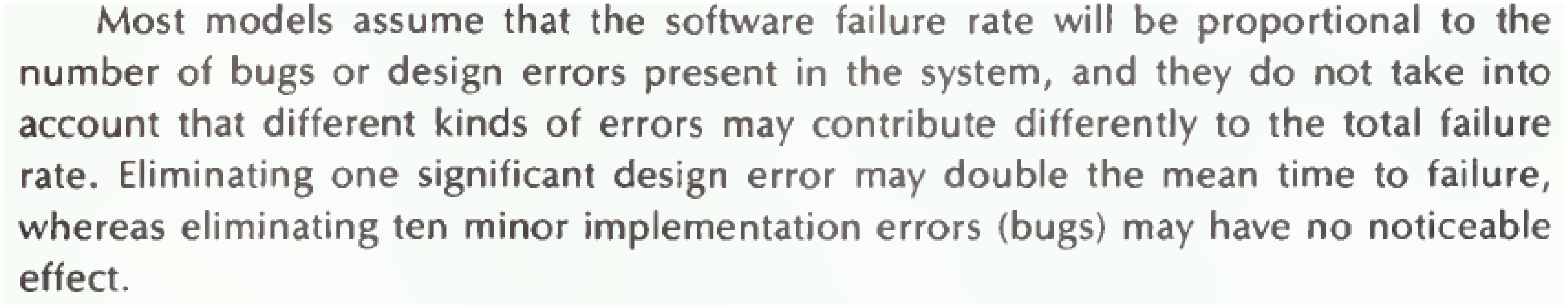
- Parameters to be estimated: N_0 , Φ

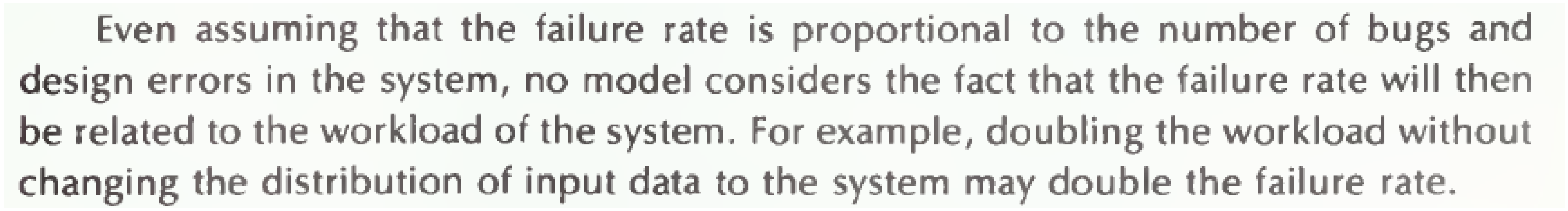
Jelinski-Moranda model: $\lambda(t)$

☞ the failure rate is constant and tends to 0 when t tends to ∞



Measures of reliability name	**Measures of reliability formula**
The probability density function	$f(t_i) = \varphi[N - (i - 1)]e^{-\varphi[N-(i-1)]t_i}$
The software reliability function	$R(t_i) = e^{-\varphi[N-(i-1)]t_i}$
The failure rate function	$\lambda(t_i) = \varphi[N - (i - 1)]$
The mean time to failure function	$\mathrm{MTTF}(t_i) = \dfrac{1}{\varphi[N - (i - 1)]}$


Most models assume that the software failure rate will be proportional to the number of bugs or design errors present in the system, and they do not take into account that different kinds of errors may contribute differently to the total failure rate. Eliminating one significant design error may double the mean time to failure, whereas eliminating ten minor implementation errors (bugs) may have no noticeable effect.

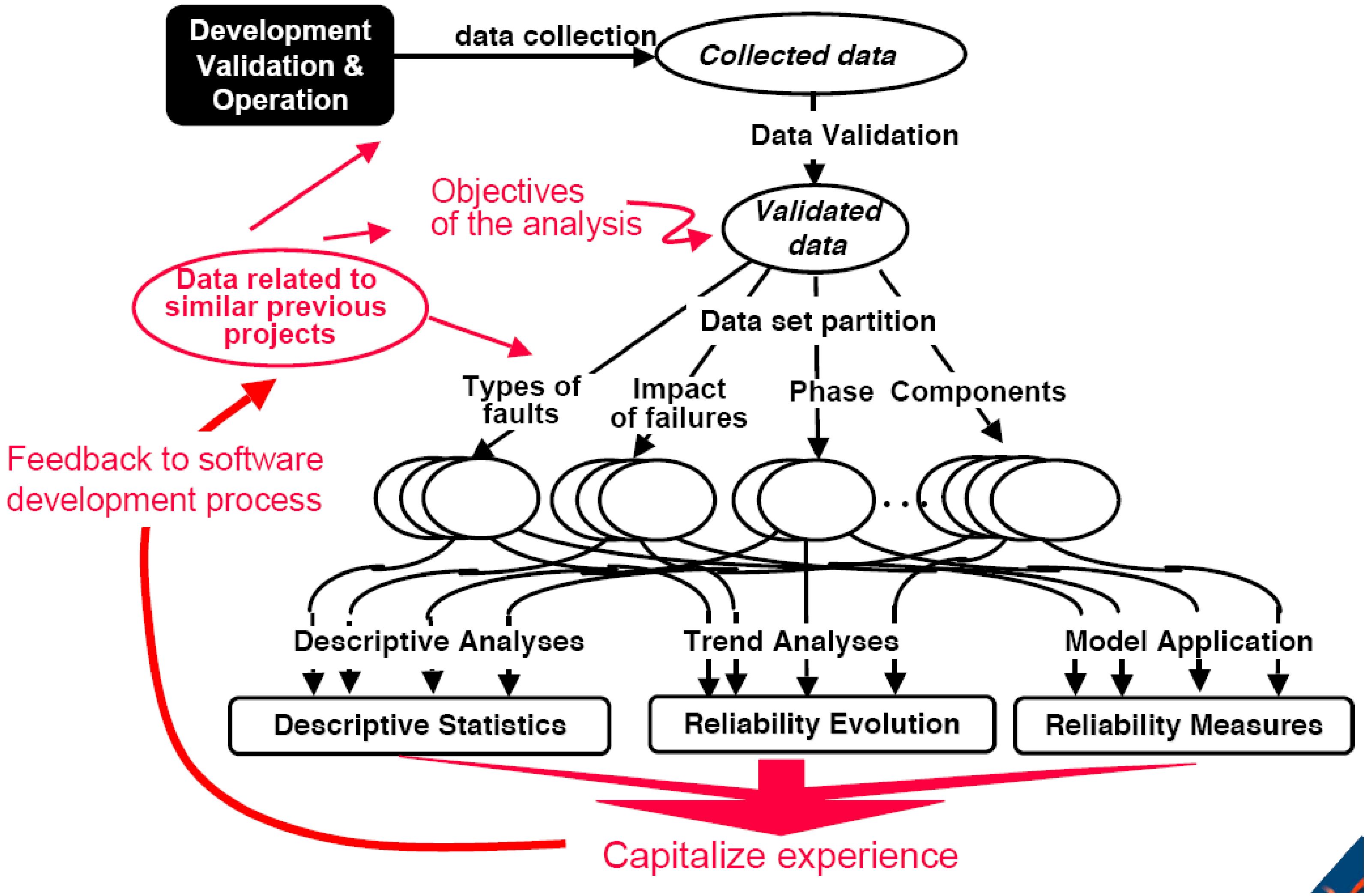

Even assuming that the failure rate is proportional to the number of bugs and design errors in the system, no model considers the fact that the failure rate will then be related to the workload of the system. For example, doubling the workload without changing the distribution of input data to the system may double the failure rate.

Siewiorek, et al
Reliable Computer Systems, Prentice Hall,1992

Software Reliability Engineering

Software Reliability Engineering (SRE) is the quantitative study of the operational behavior of software-based systems with respect to user requirements concerning reliability.

A global software reliability analysis method



(*In* Karama Kanoun, *ReSIST network of Excellence Courseware "Software Reliability Engineering", 2008* *http://www.resist-noe.org/*)

- Data collection process

  - includes data relative to product itself (software size, language, workload, ...), usage environment: verification & validation methods and failures

  - Failure reports (FR) and correction reports (CR) are generated

- Data validation process

  data elaborated to eliminate FR reporting of the same failure, FR proposing a correction related to an already existing FR, FR signalling a false or non identified problem, incomplete FRs or FRs containing inconsistent data (Unusable) …

  - Data extracted from FRs and CRs

    Time to failures (or between failures)

    Number of failures per unit of time

    Cumulative number of failures

- Descriptive statistics
  - make syntheses of the observed phenomena
  - Analyses Fault typology, Fault density of components, Failure / fault distribution among software components (new, modified, reused)
  - Analyses Relationships Fault density / size / complexity; Nature of faults / components; Number of components affected by changes made to resolve an FR .

    …….

- Trend tests
  - Control the efficiency of test activities
    - Reliability decrease at the beginning of a new activity: OK
    - Reliability row after reliability decrese: OK
    - Sudden reliability grow CAUTION!
    - …….

- Model application
  - Trend in accordance with model assumptions

Software Reliability

Due to the nature of software, no general accepted mechanisms exist to predict software reliability

Important empirical observation and experience

Good engineering methods can largely improve software reliability

Software testing serves as a way to measure and improve software reliability

Unfeasibility of completely testing a software module:
  defect-free software products cannot be assured

Databases with software failure rates are available but numbers should be used with caution and adjusted based on observation and experience