Software Dependability:

Current Approaches and Research Gaps

Karama Kanoun

LAAS-CNRS

International DEOS Workshop, 16-17 December 2010 - Tokyo, Japan

Why Software Dependability Assessment?

☞ User / customer
- Confidence in the product
- Acceptable failure rate

☞ Developer / Supplier
- During production
  - ☞ Reduce # faults (zero defect)
  - ☞ Optimize development
  - ☞ Increase operational dependability
- During operation
  - ☞ Maintenance planning
- Long term
  - ☞ Improve software dependability of next generations

Approaches to Software Dependability Assessment

- ☞ Assessment based on software characteristics
  - Language, complexity metrics, application domain, …
- ☞ Assessment based on measurements
  - Observation of the software behavior
- ☞ Assessment based on controlled experiments
  - Ad hoc vs standardized → benchmarking
- ☞ Assessment of the production process
  - Maturity models

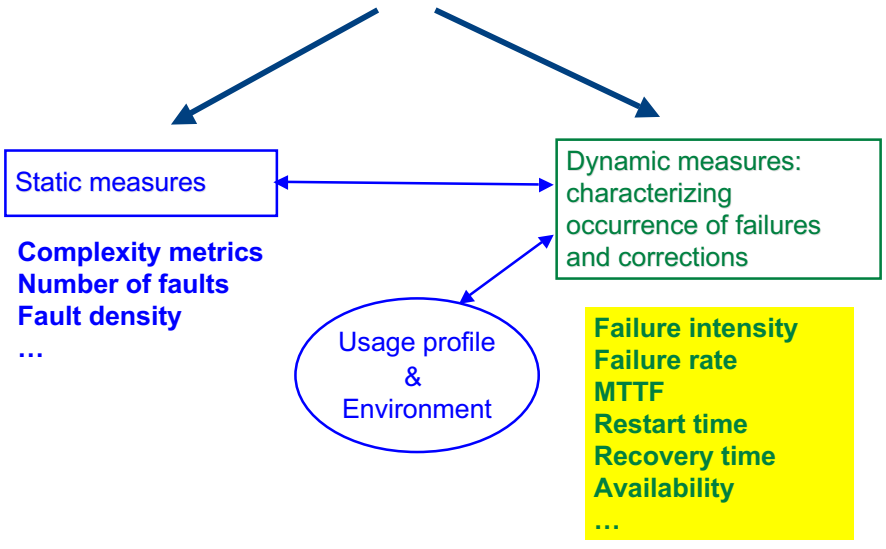
Outline of the Presentation

- ☞ Assessment based on software characteristics
  - Language, complexity metrics, application domain, …
- ☞ Assessment based on measurements
  - Observation of the software behavior
- ☞ Assessment based on controlled experiments
  - Ad hoc vs standardized → benchmarking
- ☞ Assessment of the production process
  - Maturity models

Software Dependability Assessment — Difficulties

- Non-repetitive process
- No relationship between failures and corrections
- Continuous evolution of usage profile
  - According to the development phase
  - Within a given phase
- Overselling of early reliability "growth" models
- Judgement on quality of the software developers
- What is software dependability?
  - Number of faults, fault density, complexity?
  - MTTF, failure intensity, failure rate?

Dependability Measures?



Number of Faults vs MTTF

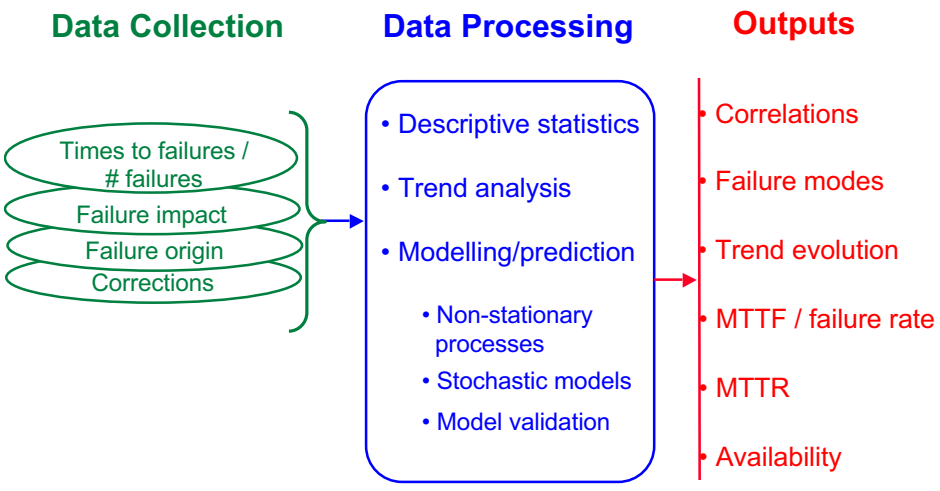
Percentage of faults and corresponding MTTF (published by IBM)

MTTF≤1.58 y

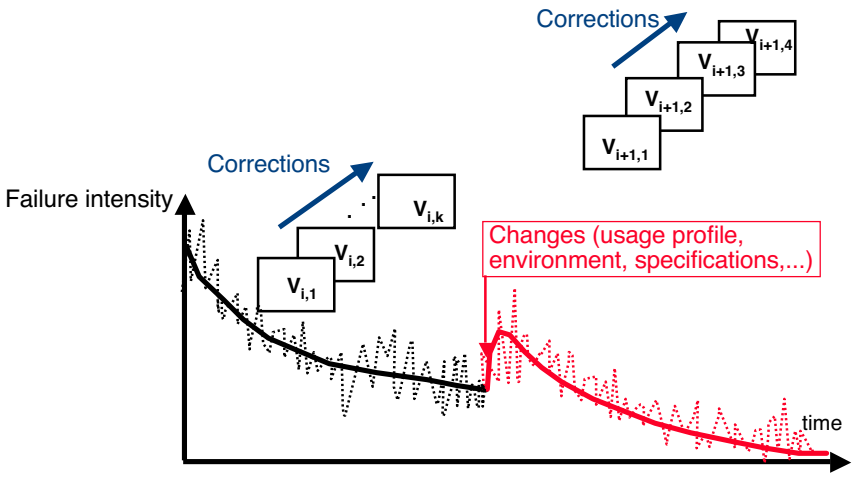
Product \ MTTF (years)	5000	1580	500	158	50	15.8	5	1.58
1	34,2	28,8	17,8	10,3	5,0	2,1	1,2	0,7
2	34,3	28,0	18,2	9,7	4,5	3,2	1,5	0,7
3	33,7	28,5	18,0	8,7	6,5	2,8	1,4	0,4
4	34,2	28,5	18,7	11,9	4,4	2,0	0,3	0,1
5	34,2	28,5	18,4	9,4	4,4	2,9	1,4	0,7
6	32,0	28,2	20,1	11,5	5,0	2,1	0,8	0,3
7	34,0	28,5	18,5	9,9	4,5	2,7	1,4	0,6
8	31,9	27,1	18,4	11,1	6,5	2,7	1,4	1,1
9	31,2	27,6	20,4	12,8	5,6	1,9	0,5	0,0

1.58 y ≤ MTTF≤ 5 y

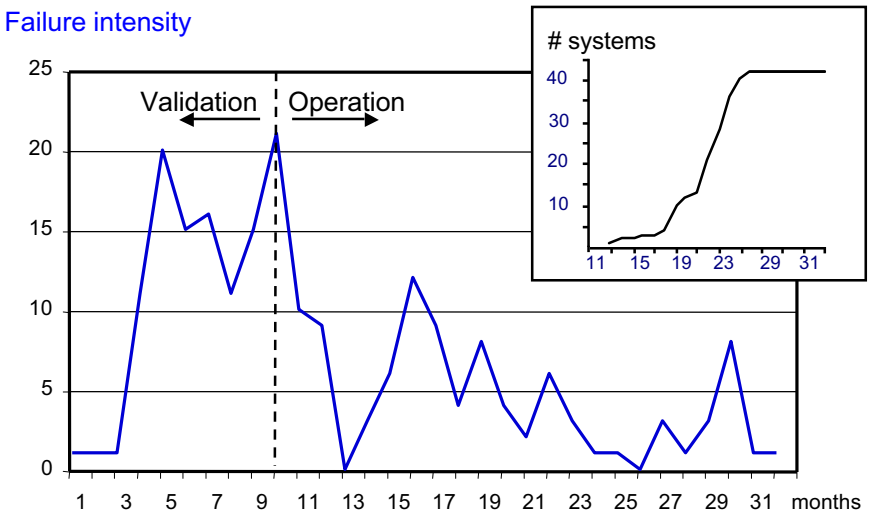
Assessment Based on Measurements



Why Trend Analysis?

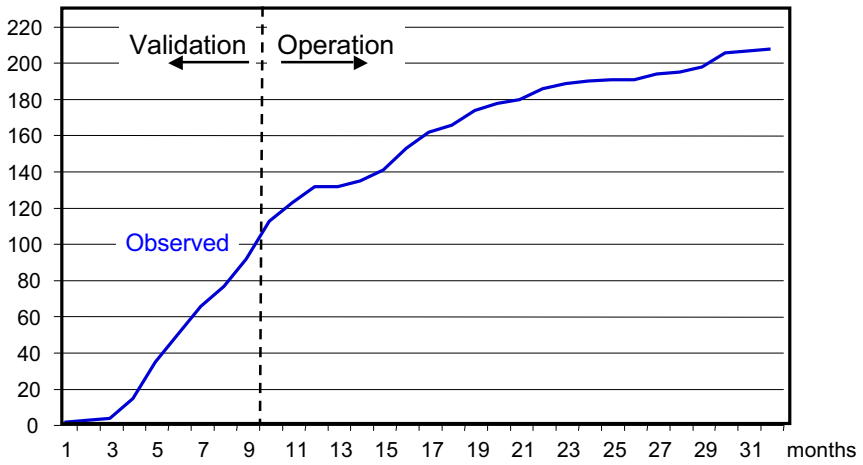


Example: Electronic Switching System



Electronic Switching System (Cont.)

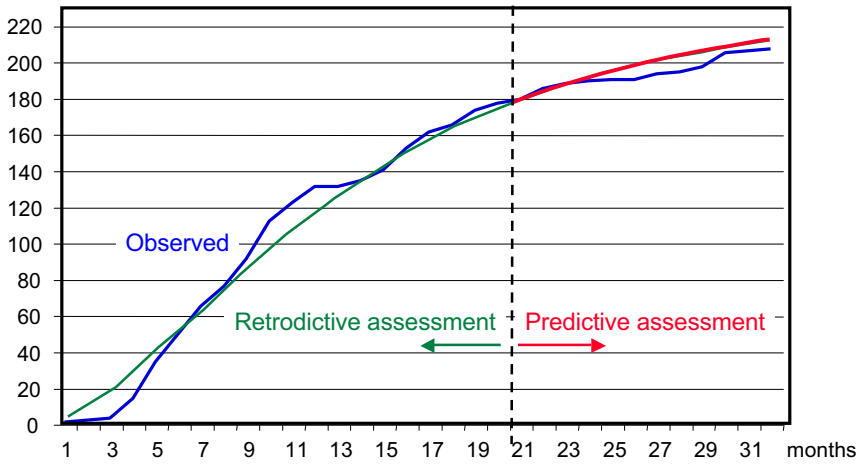
Cumulative number of failures



11

Electronic Switching System (Cont.)

Cumulative number of failures → Hyperexponential model application
⇒ maintenance planning



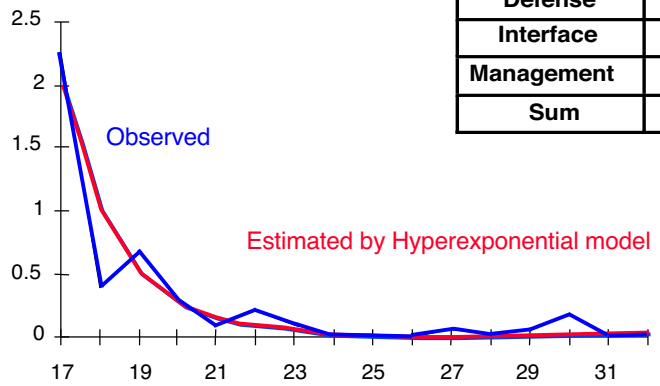
Observed # failures [20-32] = 33 Predicted # failures [21-32] = 37

12

Electronic Switching System (Cont.)

Failure intensity and failure rate in operation
(for an average system)

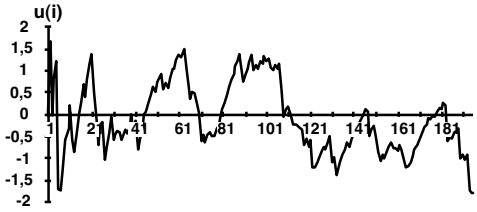
Component	Residual failure rate
Telephony	1.2 10^{-6} /h
Defense	1.4 10^{-5} /h
Interface	2.9 10^{-5} /h
Management	8.5 10^{-6} /h
Sum	5.3 10^{-5}/h



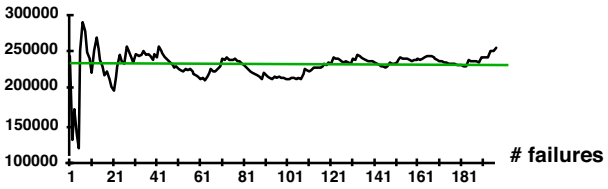
Other Example: Operating System in Operation

Data = Time to Failure during operation

Trend evolution
= stable dependability



Mean
Time to Failure



Validity of Results

Early Validation	End of Validation	Operation
☞ Trend analysis → development follow-up ~~Assessment~~	☞ Trend analysis + ☞ Assessment • operational profile • enough data? ☞ Limits: 10^{-3}/h -10^{-4}/h	☞ Trend analysis + ☞ Assessment High relevance Examples: E10-B (Alcatel ESS): 1400 systems, 3 years $\lambda = 5\ 10^{-6}$/h $\lambda_c = 10^{-7}$/h Nuclear I&C systems: 8000 systems, 4 years λ: 3 10^{-7}/h → 10^{-7}/h $\lambda_c = 4\ 10^{-8}$/h

15

Research Gaps

- ☞ Applicability to safety critical systems
  - During development
- ☞ Applicability to new classes of systems
  - Service oriented systems
  - Adaptive and dynamic software systems ⇒ on-line assessment
- ☞ Industry implication
  - Confidentiality ⇒ real-life data
  - Cost (perceptible overhead, invisible immediate benefits)
- ☞ Case of Off-The-Shelf software components?
- ☞ Accumulation of experience ⇒ software process improvement

  ⇒ assessment of the software process

16

Off-The-Shelf software components — Dependability Benchmarking

- No information available from software development
- Evaluation based on controlled experimentation

Ad hoc → Internal purpose

Standard → Dependability benchmarking

Evaluation of dependability measures / features in a non-ambiguous way → comparison

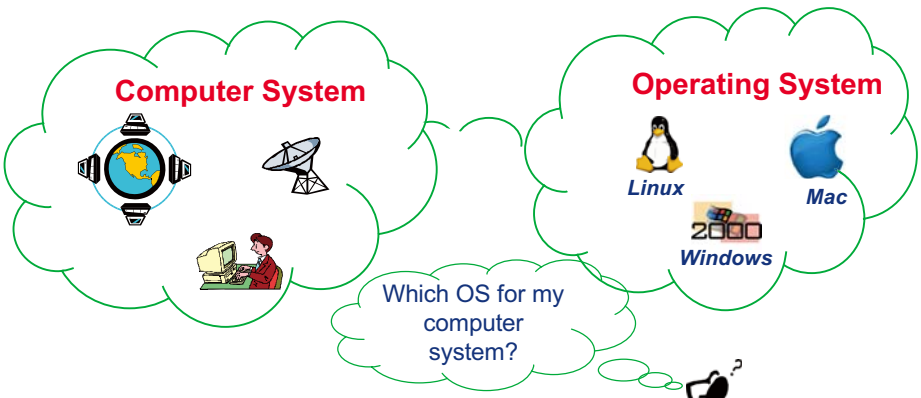
⇓

Properties

Results: available & reusable

Reproducibility, repeatability, portability, representativeness, acceptable cost

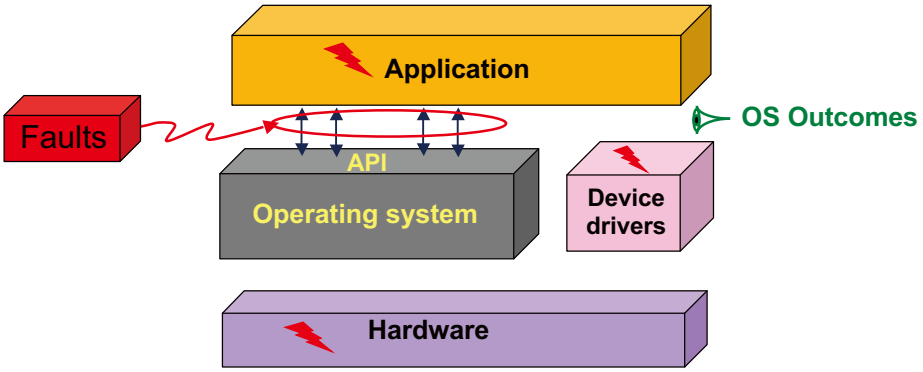
Benchmarks of Operating Systems



- Limited knowledge: functional description
- Limited accessibility and observability

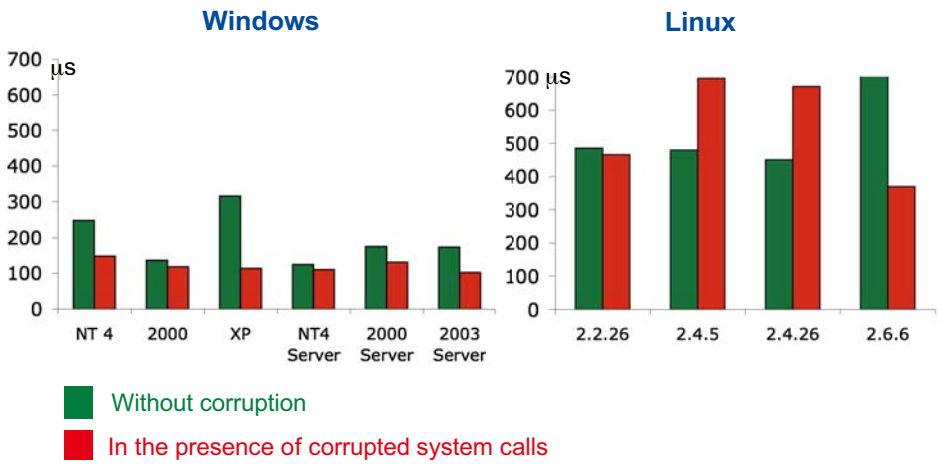
⇒ **Black-box approach ⇒ robustness benchmark**

Robustness Benchmarks

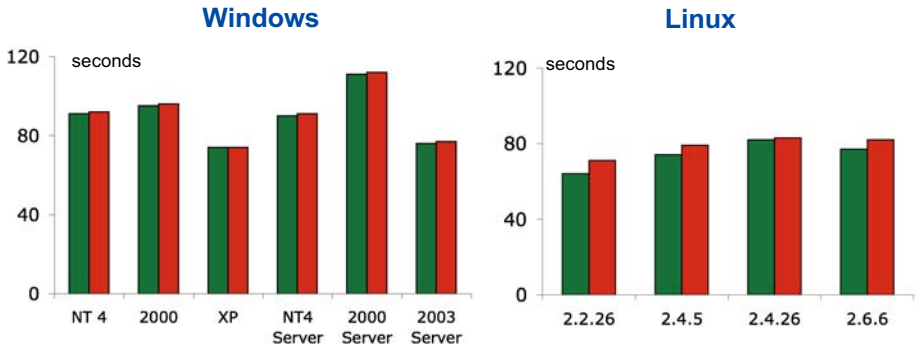


Faults = corrupted system calls

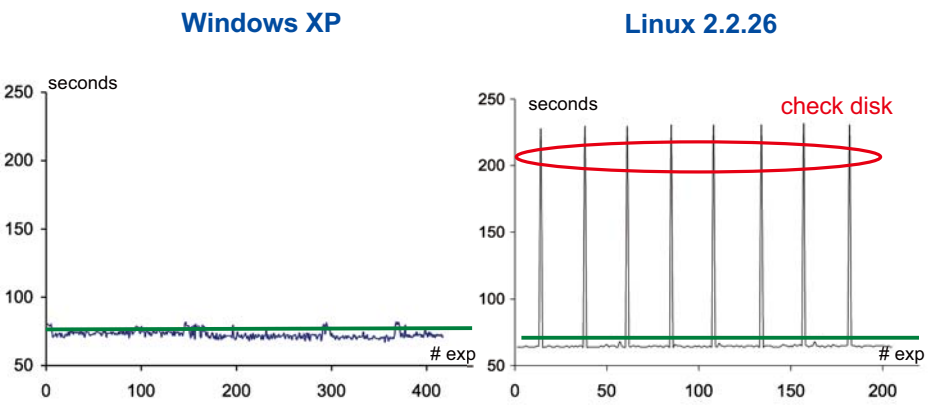
OS Response Time to Faults in the Application



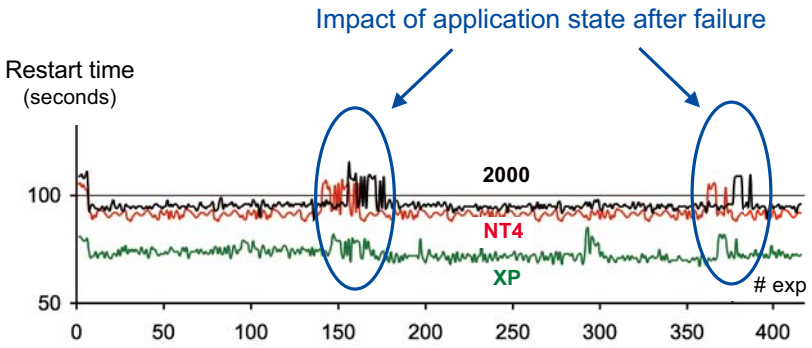
Mean Restart Time



Detailed Restart Time



More on Windows family



Benchmark Characteristics and Limitations

- A benchmark should not replace software test and validation
- Non-intrusiveness ⇒ robustness benchmarks
  (faults injected outside the benchmark target)
- Make use of available inputs and outputs → impact on measures
- Balance between cost and degree of confidence
- # dependability benchmark measures >
  # performance benchmark measures

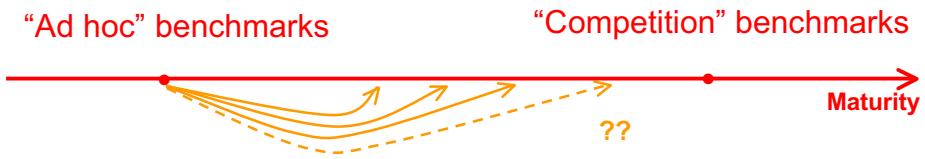
Maturity of Dependability Benchmarks

☞ Dependability benchmarks

- Infancy
- Isolated work
- Not explicitly addressed
- Acceptability?

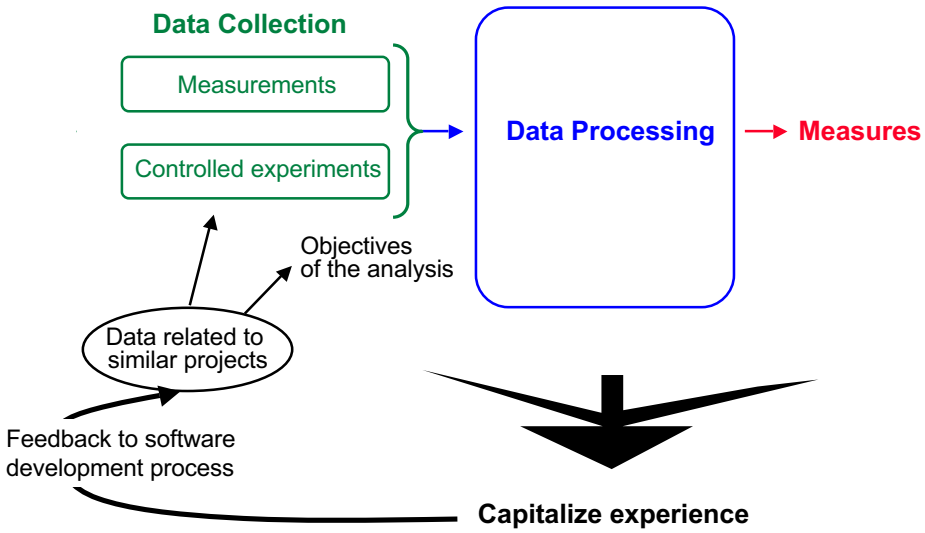
☞ Performance benchmarks

- Mature domain
- Cooperative work
- Integrated to system development
- Accepted by all actors for competitive system comparison



25

Software Process Improvement (SPI)



26

Examples of Benefits from SPI Programs

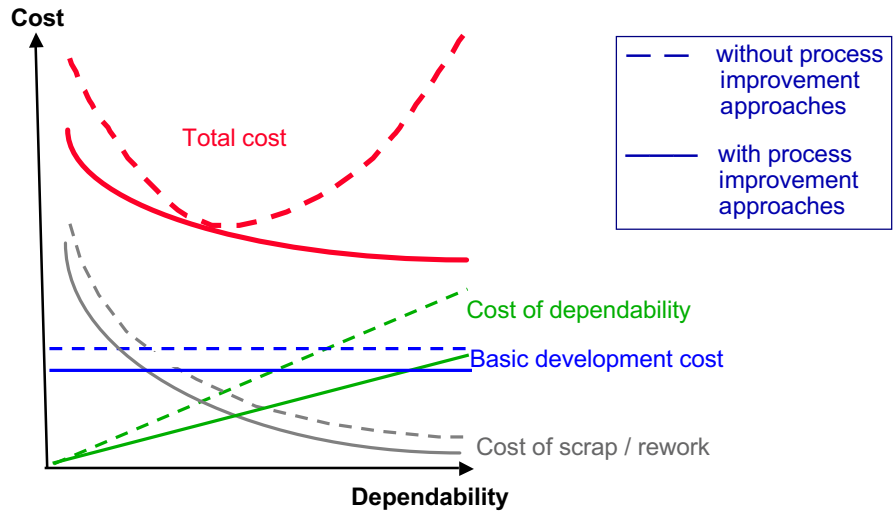
- **AT&T**(quality program):
  Customer reported problems divided by 10
  Maintenance program divided by 10
  System test interval divided by 2
  New product introduction interval divided by 3

- **IBM** (defect prevention approach):
  Fault density divided by 2 with an increase of 0.5 % of the product resources

- **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

27



Process improvement ⇒ Dependability improvement & cost reduction!

28

Ultimate objective:
more reliable software, faster, and cheaper!

29

Software Dependability:

Current Approaches and Research Gaps

Karama Kanoun

LAAS-CNRS

International DEOS Workshop, 16-17 December 2010 - Tokyo, Japan

30