Software Dependability: How Far are We?

Karama Kanoun

Dependability of Computing Systems: Memories and Future
15-16 April 2010 - Toulouse - France
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
  • Assessment of the product
  • Assessment of the production process

☞ Assessment based on controlled experiments
  • Ad hoc vs standardised → benchmarking
Outline of the Presentation

☞ Assessment based on software characteristics
  • Language, complexity metrics, application domain, …

☞ Assessment based on measurements
  • Assessment of the product
  • Assessment of the production process

☞ Assessment based on controlled experiments
  • Ad hoc vs standardized → benchmarking
Dependability Measures?

Static measures

- Complexity metrics
  - Number of faults
  - Fault density
  ...

Dynamic measures:
- characterizing occurrence of failures and corrections

Usage profile & Environment
- Failure intensity
- Failure rate
- MTTF
- Restart time
- Recovery time
- Availability
  ...

Assessment Based on Measurements

Software Process Improvement (SPI)

Data Collection

- Times to failures / # failures
- Failure impact
- Failure origin
- Corrections

Objectives of the analysis

Data related to similar projects

Feedback to software development process

Data Processing

- Data validation
- Descriptive statistics
- Trend analysis
- Modelling/prediction
  - Non-stationary processes
  - Stochastic models
  - Model validation

Measures

Capitalize experience

Assessment Based on Measurements

Software Process Improvement (SPI)

Data Collection

- Times to failures / # failures
- Failure impact
- Failure origin
- Corrections

Objectives of the analysis

Data related to similar projects

Feedback to software development process

Data Processing

- Data validation
- Descriptive statistics
- Trend analysis
- Modelling/prediction
  - Non-stationary processes
  - Stochastic models
  - Model validation

Measures

Capitalize experience
Benefits from SPI Programmes

☞ **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
Assessment Based on Measurements

Data Collection
- Times to failures / # failures
- Failure impact
- Failure origin
- Corrections

Data Processing
- Data validation
- Descriptive statistics
- Trend analysis
- Modelling/prediction
  - Non-stationary processes
  - Stochastic models
  - Model validation
- Trend evolution
- Failure modes
- Measures
  - MTTF
  - Failure rate
  - Failure Intensity
  - Availability
Why Trend Analysis?

Failure intensity

Corrections

Changes (usage profile, environment, specifications, ...)

time
Example: Electronic Switching System

Failure intensity

# systems

Validations

Operation

Example: Electronic Switching System

Examp

Example

Electronic Switching System
Cumulative number of failures

Electronic Switching System (Cont.)
Electronic Switching System (Cont.)

Cumulative number of failures → Hyperexponential model application
⇒ maintenance planning

Observed # failures [20-32] = 33
Predicted # failures [21-32] = 37
Failure intensity and failure rate in operation (for an average system)

<table>
<thead>
<tr>
<th>Component</th>
<th>Residual failure rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Telephony</td>
<td>$1.2 \times 10^{-6} /h$</td>
</tr>
<tr>
<td>Defense</td>
<td>$1.4 \times 10^{-5} /h$</td>
</tr>
<tr>
<td>Interface</td>
<td>$2.9 \times 10^{-5} /h$</td>
</tr>
<tr>
<td>Management</td>
<td>$8.5 \times 10^{-6} /h$</td>
</tr>
<tr>
<td>Sum</td>
<td>$5.3 \times 10^{-5} /h$</td>
</tr>
</tbody>
</table>

Residual failure rate: $5.7 \times 10^{-5} /h$
Other Example: Operating System

Observed Time to Failure during operation

Mean Time to Failure

Trend evolution = stable dependability
## Validity of Results

<table>
<thead>
<tr>
<th>Early Validation</th>
<th>End of Validation</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>☜ Trend analysis</td>
<td>☜ Trend analysis +</td>
<td>☜ Trend analysis +</td>
</tr>
<tr>
<td>→ development</td>
<td>Assessment</td>
<td>Assessment</td>
</tr>
<tr>
<td>follow-up</td>
<td>• operational profile</td>
<td>High relevance</td>
</tr>
<tr>
<td></td>
<td>• enough data</td>
<td>Examples:</td>
</tr>
<tr>
<td></td>
<td>☜ Limits: $10^{-3}/h$ - $10^{-4}/h$</td>
<td>E10-B (Alcatel ESS):</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1400 systems, 3 years</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\lambda = 5 \times 10^{-6}/h$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\lambda_c = 10^{-7}/h$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Nuclear I&amp;C systems:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8000 systems, 4 years</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\lambda: 3 \times 10^{-7}/h \rightarrow 10^{-7}/h$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\lambda_c = 4 \times 10^{-8}/h$</td>
</tr>
</tbody>
</table>
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)

☞ Accumulation of experience ⇒ software process improvement
  ⇒ assessment of the software process

☞ Case of Off-The-Shelf software?
Dependability Benchmarking
Off-The-Shelf software

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

Ad hoc Standard

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

Properties
Reproducibility, repeatability, portability, representativeness, acceptable cost

Results: available & reusable
Benchmarks of Operating Systems

Which OS for my computer system?

- Limited knowledge: functional description
- Limited accessibility and observability
  ⇒ **Black-box approach** ⇒ robustness benchmark
Faults = corrupted system calls
OS Response Time to Faults in the Application

Windows

Linux

Without corruption

In the presence of corrupted system calls
Mean Restart Time

**Windows**

- NT 4: Without corruption - 80 seconds
- 2000: Without corruption - 80 seconds
- XP: Without corruption - 80 seconds
- NT4 Server: In the presence of corrupted system calls - 120 seconds
- 2000 Server: In the presence of corrupted system calls - 120 seconds
- 2003 Server: In the presence of corrupted system calls - 120 seconds

**Linux**

- 2.2.26: Without corruption - 80 seconds
- 2.4.5: Without corruption - 80 seconds
- 2.4.26: Without corruption - 80 seconds
- 2.6.6: Without corruption - 80 seconds
- 2.6.6: In the presence of corrupted system calls - 120 seconds
Detailed Restart Time

Windows XP

Linux 2.2.26

check disk
More on Windows family

Impact of application state after failure

Restart time (seconds)
Benchmark Characteristics

☞ A benchmark should not replace software test and validation

☞ Non-intrusiveness $\Rightarrow$ robustness benchmarks

(faults injected outside the benchmark target)

☞ Make use of available inputs and outputs $\Rightarrow$ impact on measures

☞ Balance between cost and degree of confidence

☞ # dependability benchmark measures >

# performance benchmark measures

$\Rightarrow$ Lack of maturity
Maturity

♦ 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

“Ad hoc” benchmarks

“Competition” benchmarks

??
Ultimate objective: more reliable software, faster, and cheaper!
Software Dependability:
How Far are We?

Karama Kanoun

Dependability of Computing Systems: Memories and Future
15-16 April 2010 - Toulouse - France