Fundamentals of Dependability

Jean-Claude Laprie
**Dependability**: ability to deliver service that can justifiably be trusted

Service delivered by a system: its behavior as it is perceived by its user(s)

User: another system that interacts with the former

Function of a system: what the system is intended to do

*(Functional) Specification*: description of the system function

Correct service: when the delivered service implements the system function

*(Service) Failure*: event that occurs when the delivered service deviates from correct service, either because the system does not comply with the specification, or because the specification did not adequately describe its function

Failure modes: the ways in which a system can fail, ranked according to failure severities

Part of system state that may cause a subsequent service failure: *error*

Adjudged or hypothesized cause of an error: *fault*

**Dependability**: ability to avoid failures that are unacceptably frequent or severe

Failures unacceptably frequent or severe: *dependability failure*
Dependability

- Readiness for usage
- Continuity of service
- Absence of catastrophic consequences on the user(s) and the environment
- Absence of unauthorized disclosure of information
- Absence of improper system alterations
- Ability to undergo repairs and evolutions

- Availability
- Reliability
- Safety
- Confidentiality
- Integrity
- Maintainability

Security

Absence of unauthorized access to, or handling of, system state
Dependability attributes

- Availability, Reliability, Safety, Confidentiality, Integrity, Maintainability: Primary attributes
- Secondary attributes
  - Specialization
    - Robustness: dependability with respect to external faults
    - Survivability: dependability in the presence of active fault(s)
    - Resilience: dependability when facing functional, environmental, or technological changes
  - Distinguishing among various types of (meta-)information
    - Accountability: availability and integrity of the person who performed an operation
    - Authenticity: integrity of a message content and origin, and possibly some other information, such as the time of emission
    - Non-repudiability: availability and integrity of the identity of the sender of a message (non-repudiation of the origin), or of the receiver (non-repudiation of reception)
Fault \rightarrow Error \rightarrow Failure \rightarrow Fault \rightarrow \ldots

- **Activation**: Part of system state that *may* cause a subsequent failure.
- **Propagation**: Deviation of the delivered service from correct service, i.e., implementing the system function.
- **Causation**: System does not comply with specification.
- **Error**: Adjudged or hypothesized cause of an error.

**Dependability Threats**

*ReSIST courseware — Jean-Claude Laprie — Fundamentals of Dependability*
Internal fault, dormant vulnerability

Activation

(computation process)

Error

propagation

Error

prop.

Error

Service Interface

Incorrect Service: Outage

Correct Service

Failure

Incorrect Service: Outage

Failure

Correct Service

External fault occurrence
System: Set of components interconnected in order to interact
Component: Another system
Means for Dependability

- Preventing occurrence or introduction of faults
- Delivering correct service in spite of faults
- Reducing the presence of faults
- Estimating the present number, the future incidence and the likely consequences of faults

Fault Prevention
Fault Tolerance
Fault Removal
Fault Forecasting

Dependability Provision
Dependability Assessment
Fault Avoidance
Fault Acceptance

ReSIST courseware — Jean-Claude Laprie — Fundamentals of Dependability
Dependability definitions

- Original definition: ability to deliver service that can justifiably be trusted
  - Enables to generalize availability, reliability, safety, confidentiality, integrity, maintainability, that are then attributes of dependability

- Alternate definition: ability to avoid service failures that are unacceptably frequent or severe
  - A system can, and usually does, fail. Is it however still dependable? When does it become undependable?

  criterion for deciding whether or not, in spite of service failures, a system is still to be regarded as dependable

- Dependence of system A on system B is the extent to which system A’s dependability is (or would be) affected by that of system B

- Trust: accepted dependence
  - Explicitly
  - Implicitly
## Dependability and similar notions

<table>
<thead>
<tr>
<th>Concept</th>
<th>Dependability</th>
<th>High Confidence</th>
<th>Survivability</th>
<th>Trustworthiness</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Goal</strong></td>
<td>1) ability to deliver service that can justifiably be trusted</td>
<td>consequences of the system behavior are well understood and predictable</td>
<td>capability of a system to fulfill its mission in a timely manner</td>
<td>assurance that a system will perform as expected</td>
</tr>
<tr>
<td></td>
<td>2) ability of a system to avoid service failures that are unacceptably frequent or severe</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Threats present</strong></td>
<td>1) development faults (e.g., software flaws, hardware errata, malicious logic)</td>
<td>• internal and external threats</td>
<td>1) attacks (e.g., intrusions, probes, denials of service)</td>
<td>1) hostile attacks (from hackers or insiders)</td>
</tr>
<tr>
<td></td>
<td>2) physical faults (e.g., production defects, physical deterioration)</td>
<td>• naturally occurring hazards and malicious attacks from a sophisticated and well-funded adversary</td>
<td>2) failures (internally generated events due to, e.g., software design errors, hardware degradation, human errors, corrupted data)</td>
<td>2) environmental disruptions (accidental disruptions, either man-made or natural)</td>
</tr>
<tr>
<td></td>
<td>3) interaction faults (e.g., physical interference, input mistakes, attacks, including viruses, worms, intrusions)</td>
<td></td>
<td>3) accidents (externally generated events such as natural disasters)</td>
<td>3) human and operator errors (e.g., software flaws, mistakes by human operators)</td>
</tr>
</tbody>
</table>
Dependability

Attributes
- Availability
- Reliability
- Safety
- Confidentiality
- Integrity
- Maintainability

Means
- Fault Prevention
- Fault Tolerance
- Fault Removal
- Fault Forecasting

Threats
- Faults
- Errors
- Failures

Security
# Outline

<table>
<thead>
<tr>
<th>Section</th>
<th>Slides</th>
</tr>
</thead>
<tbody>
<tr>
<td>Threats: faults, errors, failures</td>
<td>15</td>
</tr>
<tr>
<td>State of the art from statistics</td>
<td>28</td>
</tr>
<tr>
<td>Fault removal</td>
<td>45</td>
</tr>
<tr>
<td>Fault forecasting</td>
<td>57</td>
</tr>
<tr>
<td>Fault tolerance</td>
<td>66</td>
</tr>
<tr>
<td>Development of dependable systems</td>
<td>90</td>
</tr>
<tr>
<td>From dependability to resilience</td>
<td>102</td>
</tr>
<tr>
<td>Conclusion</td>
<td>109</td>
</tr>
<tr>
<td>References</td>
<td>111</td>
</tr>
</tbody>
</table>
Dependability threats: faults, errors, failures
Error of a programmer

Fault
Impaired instructions or data

Activation
Faulty component and inputs

Error

Propagation
When delivered service deviates (value, timing) from implementing function

Failure
Short-circuit in integrated circuit

**Failure**

**Causation**

**Fault**

Stuck-at connection, modification of circuit function

**Activation**

Faulty component and inputs

**Error**

**Propagation**

When delivered service deviates (value, timing) from implementing function

**Failure**
Operator Error
Inappropriate human-system interaction
Fault
Error
Propagation
When delivered service deviates (value, timing) from implementing function
Failure
Electromagnetic perturbation

Fault

Fault

Impaired memory data

Activation

Faulty component and inputs

Error

Propagation

When delivered service deviates (value, timing) from implementing function

Failure
ReSIST courseware — Jean-Claude Laprie — Fundamentals of Dependability
Failures
- Domain
- Détectability
- Consistency
- Consequences

**Value failures**
[value of delivered service deviates from implementing system function]

**Timing failures**
[timing of service delivery (instant or duration) deviates from implementing system function]

**Signalled failures**
[delivered service is detected as incorrect, and signalled as such]

**Unsignalled failures**
[incorrect service delivery is not detected]

**Consistent failures**
[incorrect service identically perceived by all users]

**Inconsistent, or Byzantine, failures**
[some, or all, users perceive differently incorrect service]

**Minor failures**
[harmful consequences are of similar cost to the benefits provided by correct service delivery]

**Catastrophic failures**
[cost of harmful consequences is orders of magnitude, or even incommensurably higher than the benefits provided by correct service delivery]
Failure domain

- Value (correct timing)
  - Service delivered too early
    - Early timing failures
  - Service delivered too late
    - Late timing failures
  - Suspended service
    - Halt failures
  - Erratic service
    - Erratic failures

- Timing (correct value)

- Value and timing

Failure of detecting mechanisms

- Non signalling of incorrect service: unsignalled failure
- Signalling incorrect service in absence of failure: false alarm

System whose all failures are, to an acceptable extent

- halt failures: fail-halt system
- minor failures: fail-safe system
Phase of creation or occurrence

Development faults
[occur during (a) system development, (b) maintenance during the use phase, and (c) generation of procedures to operate or to maintain the system]

Operational faults
[occur during service delivery of the use phase]

System boundaries

Internal faults
[originate inside the system boundary]

External faults
[originate outside the system boundary and propagate errors into the system by interaction or interference]

Phenomenological cause

Natural faults
[caused by physico-chemical natural phenomena without human participation]

Human-made faults
[result from human actions]

Intent

Malicious faults
[introduced by a human with the malicious objective of causing harm to the system]

Non-malicious faults
[introduced without a malicious objective]

Capability

Accidental faults
[introduced inadvertently]

Deliberate faults
[result of a decision]

Incompetence faults
[result from lack of professional competence by the authorized human(s), or from inadequacy of the development organization]

Persistence

Permanent faults
[presence is assumed to be continuous in time]

Transient faults
[presence is bounded in time]
Human-made Faults

Intent

Non-malicious
- Accidental (Mistakes)
- Deliberate (Bad decisions)
- Incompetence

Malicious
- Deliberate
- Intrusion attempts

Capability

Interaction (operators, maintainers) & Development (designers)
- Malicious logic faults: logic bombs, Trojan horses, trapdoors, viruses, worms, zombies
- Individuals & organizations

Physical faults    Development faults    Interaction faults

Hardware    Software    System
Failure pathology

- Active fault produces error(s)
  - Activation of dormant fault by computation process
  - Activation of vulnerability
  - External fault

- Error is latent or detected
  - Error propagation
  - Other errors
  - Error can disappear before detection

- Component failure
  - Fault as viewed by
    - Components interacting with failed component
    - System containing failed component

**Diagram:**
- Fault → Error → Failure → Fault
  - Activation or occurrence of fault
  - Propagation of error
  - Causation of failure (interaction, composition)
Fault → Error → Failure → Fault

Facility for stopping recursion

Context dependent

Interaction faults

Prior presence of a *vulnerability*: Internal fault that enables an external fault to harm the system

Activation or Occurrence

Propagation

Causation

Error alters service

Interaction or composition

Solid (hard) faults

Elusive (soft) faults

Activation reproducibility

Permanent faults (development, physical, interaction)

Transient faults (physical, interaction)

Elusive faults

Solid faults

Intermittent faults
State of the art from statistics
<table>
<thead>
<tr>
<th>Date</th>
<th>Event</th>
<th>Physical</th>
<th>Development</th>
<th>Interaction</th>
<th>Localized</th>
<th>Distributed</th>
<th>Availability</th>
<th>Reliability</th>
<th>Safety</th>
<th>Confidentiality</th>
</tr>
</thead>
<tbody>
<tr>
<td>June 1980</td>
<td>False alerts at the North American Air Defense (NORAD)</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>June 1985 - January 1987</td>
<td>Excessive radiotherapy doses (Therac-25)</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>August 1986 - 1987</td>
<td>The &quot;wily hacker&quot; penetrates several tens of sensitive computing facilities</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>November 1988</td>
<td>Internet worm</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>15 January 1990</td>
<td>9 hours outage of the long-distance phone in the USA</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>February 1991</td>
<td>Scud missed by a Patriot (Dhahran, Gulf War)</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>November 1992</td>
<td>Crash of the communication system of the London ambulance service</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>26 and 27 June 1993</td>
<td>Authorization denial of credit card operations in France</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>4 June 1996</td>
<td>Failure of Ariane 5 maiden flight</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>13 April 1998</td>
<td>Crash of the AT&amp;T data network</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>February 2000</td>
<td>Distributed denials of service on large Web sites</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>May 2000</td>
<td>Virus I love you</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>July 2001</td>
<td>Worm Code Red</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>August 2003</td>
<td>Propagation of the electricity blackout in the USA and Canada</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>October 2006</td>
<td>83,000 e-mail addresses, credit card info, banking transaction files stolen in UK</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>
### Average outage costs

<table>
<thead>
<tr>
<th>Industry sector</th>
<th>Energy</th>
<th>Manufacturing</th>
<th>Financial institutions</th>
<th>Insurance</th>
<th>Retail</th>
<th>Banking</th>
<th>Millions of $ revenue/hour lost</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2.8</td>
<td>1.6</td>
<td>1.4</td>
<td>1.2</td>
<td>1.1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

### Yearly cost of failures

**Estimates of insurance companies (2000)**

<table>
<thead>
<tr>
<th></th>
<th>France (private sector)</th>
<th>USA</th>
<th>UK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accidental faults</td>
<td>0.9 G€</td>
<td>4 G$</td>
<td></td>
</tr>
<tr>
<td>Malicious faults</td>
<td>1 G€</td>
<td></td>
<td>1.25 G£</td>
</tr>
<tr>
<td>Global estimate</td>
<td></td>
<td>USA : 80 G$</td>
<td>UE : 60 G€</td>
</tr>
</tbody>
</table>

### Maintenance costs

- Space shuttle on-board software: 100 M $ / an

### Cost of software project cancellation (failure of the development process)

<table>
<thead>
<tr>
<th></th>
<th>Success</th>
<th>Challenged</th>
<th>Cancelled</th>
</tr>
</thead>
<tbody>
<tr>
<td>USA [Standish Group, 2002, 13522 projects]</td>
<td>34%</td>
<td>51%</td>
<td>15%</td>
</tr>
<tr>
<td></td>
<td>1 G$</td>
<td>4 G$</td>
<td>7 G$</td>
</tr>
</tbody>
</table>

ReSIST courseware — Jean-Claude Laprie — Fundamentals of Dependability
### Accidental faults

<table>
<thead>
<tr>
<th>Faults</th>
<th>Rank</th>
<th>Proportion</th>
<th>Rank</th>
<th>Proportion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physical internal</td>
<td>3</td>
<td>~ 10%</td>
<td>2</td>
<td>15-20%</td>
</tr>
<tr>
<td>Physical external</td>
<td>3</td>
<td>~ 10%</td>
<td>2</td>
<td>15-20%</td>
</tr>
<tr>
<td>Human interactions</td>
<td>2</td>
<td>~ 20%</td>
<td>1</td>
<td>40-50%</td>
</tr>
<tr>
<td>Development</td>
<td>1</td>
<td>~ 60%</td>
<td>2</td>
<td>15-20%</td>
</tr>
</tbody>
</table>

Number of failures  
[consequences and outage durations depend upon application]  

<table>
<thead>
<tr>
<th>Faults</th>
<th>Rank</th>
<th>Proportion</th>
<th>Rank</th>
<th>Proportion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Controlled systems</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(e.g., civil airplanes, phone network, Internet frontend servers)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dedicated computing systems</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(e.g., transaction processing, electronic switching, Internet backend servers)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Permanent faults and production yield for Intel microprocessors

1 FIT = 10^{-9}/h
DPM : defects per million

1st microprocessor: 4004, 1971, 2300 transistors
Transient operational physical faults: proportion increases with integration

Reduced geometric dimensions

Lower and lower energy

Environmental perturbations become faults more easily

Directly:
- $\alpha$ particles, outer space particles

Indirectly:
- Limitation of electrical fields
- Lower supply voltage
- Reduced noise immunity
Cosmic rays

Primaries disappear
~ 25 km

~ 1600/m²-s cascade to sea level

~ 100/cm²-s at 12000 m

~ 1/cm²-s sea-level flux

Low energy deflected

\( \mu^+ \)

\( \mu^- \)

\( n \)

\( p \)

\( \pi^+ \)

\( \pi^- \)

\( \pi^0 \)

\( e^+ \)

\( e^- \)

\( \gamma \)

\( p \)

\( n \)
Andrew network of CMU, 13 SUN stations, 21 station.yrs

<table>
<thead>
<tr>
<th></th>
<th>Number of manifestations</th>
<th>Mean time to manifestation (h)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Permanent faults</td>
<td>29</td>
<td>6552</td>
</tr>
<tr>
<td>Intermittent faults</td>
<td>1056</td>
<td>183</td>
</tr>
<tr>
<td>System failures</td>
<td>298</td>
<td>689</td>
</tr>
</tbody>
</table>

Tandem experimental data

Examination of anomaly log files (several tens of systems, 6 months): 132 recorded software faults

- 1 solid fault (« bohrbug »)
- 131 elusive faults (« heisenbugs »)
Complexity
Economic pressure

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

<table>
<thead>
<tr>
<th>Availability</th>
<th>Outage duration/yr</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.999999</td>
<td>32s</td>
</tr>
<tr>
<td>0.9999</td>
<td>5mn 15s</td>
</tr>
<tr>
<td>0.999</td>
<td>52mn 34s</td>
</tr>
<tr>
<td>0.99</td>
<td>8h 46mn</td>
</tr>
<tr>
<td>0.9</td>
<td>3j 16h</td>
</tr>
<tr>
<td>0.9</td>
<td>36j 12h</td>
</tr>
</tbody>
</table>
**Tandem**

<table>
<thead>
<tr>
<th>Number</th>
<th>Duration (yrs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clients</td>
<td>2000</td>
</tr>
<tr>
<td>Systems</td>
<td>9000</td>
</tr>
<tr>
<td>Processors</td>
<td>25500</td>
</tr>
<tr>
<td>Disks</td>
<td>74000</td>
</tr>
</tbody>
</table>

| Reported outages | 438 |
| System MTBF     | 21 years |

**Mean time to system crash, due to hardware failure**

- ECL-TCM
- CMOS
- 308X/3090
- G1-G5
- 9020


**High end IBM servers**

- MTBF (yrs)
  - Maintenance
  - Environment
  - Hardware
  - Operations
  - Software
  - Total

- 0
- 50
- 100
- 150
- 200
- 250
- 300
- 350
- 400
- 450

Website uptime statistics (Netcraft)

Top 50 most requested sites

<table>
<thead>
<tr>
<th>MTTR</th>
<th>Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 min</td>
<td>99.98</td>
</tr>
<tr>
<td>10 mins</td>
<td>99.83</td>
</tr>
<tr>
<td>1 hour</td>
<td>99.01</td>
</tr>
<tr>
<td>8 hours</td>
<td>98.59</td>
</tr>
</tbody>
</table>

![Graph showing website uptime statistics with availability percentages for different MTTR values.](image)
### Three large websites

[From D. Oppenheimer, A. Ganapathi, D.A. Patterson, ‘Why do Internet services fail, and what can be done about it?’, USISTS ‘03]

<table>
<thead>
<tr>
<th>Website</th>
<th><strong>Online</strong> (mature)</th>
<th><strong>Readmostly</strong> (mature)</th>
<th><strong>Content</strong> (bleeding edge)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service characteristic</td>
<td>Hits per day</td>
<td>~100 million</td>
<td>~100 million</td>
</tr>
<tr>
<td></td>
<td># of machines</td>
<td>~500, 2 sites</td>
<td>&gt;2000, 4 sites</td>
</tr>
<tr>
<td></td>
<td>Front-end node architecture</td>
<td>Solaris on SPARC and x86</td>
<td>Open-source OS on x86</td>
</tr>
<tr>
<td></td>
<td>Period of data stud.</td>
<td>7 months</td>
<td>6 months</td>
</tr>
<tr>
<td></td>
<td>Component failures</td>
<td>296</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>Service failures</td>
<td>40</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>MTTF</td>
<td>126 hours</td>
<td>206 hours</td>
</tr>
</tbody>
</table>

| Service failure cause by location | | | |
| Front-end | 77% | 0% | 66% |
| Back-end | 3% | 10% | 11% |
| Network | 18% | 81% | 18% |
| Unknown | 2% | 9% | 4% |

| Average TTR by part of service (hrs) | | | |
| Front-end | 9.4 (16 serv. fai.) | N/A | 2.5 (10 serv. fai.) |
| Back-end | 7.3 (5 serv. fai.) | 0.2 (1 serv. fai.) | 14 (3 serv. fai.) |
| Network | 7.8 (4 serv. fai.) | 1.2 (16 serv. fai.) | 1.2 (2 serv. fai.) |

| Average availability | 93.5% | 97.2% | 97.8% |
Component failure to service failure

Online

Content

ReSIST courseware — Jean-Claude Laprie — Fundamentals of Dependability
Malicious faults

SEI/CERT Statistics: vulnerabilities reported

Malicious faults
Slammer/Sapphire worm

The fastest computer worm in history. As it began spreading throughout the Internet, it doubled in size every 8.5 seconds. It infected more than 90 percent of vulnerable hosts within 10 minutes. The worm began to infect hosts slightly before 05:30 UTC on Saturday, January 25, 2003. Sapphire exploited a buffer overflow vulnerability in computers on the Internet running Microsoft's SQL Server or MSDE 2000 (Microsoft SQL Server Desktop Engine). This weakness in an underlying indexing service was discovered in July 2002; Microsoft released a patch for the vulnerability before it was announced. The worm infected at least 75,000 hosts, perhaps considerably more, and caused network outages and such unforeseen consequences as canceled airline flights, interference with elections, and ATM failures.

Global Information Security Survey 2004 — Ernst & Young

Loss of availability: Top ten incidents

Percentage of respondents that indicated the following incidents resulted in an unexpected or unscheduled outage of their critical business

- Hardware failures
- Major virus, Trojan horse, or Internet worms
- Telecommunications failure
- Software failure
- Third party failure, e.g., service provider
- System capacity issues
- Operational errors, e.g., wrong software loaded
- Infrastructure failure, e.g., fire, blackout
- Former or current employee misconduct
- Distributed Denial of Service (DDoS) attacks

Non malicious: 76%
Malicious: 24%

Occurrences

Risk perception

Occurrence impact

3 year trends
- stable
- increase
- decrease
Fault Removal
Reducing the presence of faults

- **Verification**
  - Checking whether the system satisfies verification conditions
    - general
    - specific

- **Diagnosis**

- **Correction**

- **Non-regression verification**
Verification

Static

Static analysis

System

Reviews and inspections

Static analysis

Theorem proving

Model checking

Dynamic

Effective execution

Symbolic inputs

Valued inputs

System model

System

Symbolic execution

Test

Specification

Design

Implementation
Reviews and inspections ↔ Testing

HP data

<table>
<thead>
<tr>
<th>Uncovered faults/hour</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operational testing</td>
</tr>
<tr>
<td>Functional testing</td>
</tr>
<tr>
<td>Structural testing</td>
</tr>
<tr>
<td>Inspections</td>
</tr>
</tbody>
</table>

Schneider Electric data: nuclear control software

![Chart showing verification effort in man.day and number of uncovered faults for different types of tests](chart.png)
Space shuttle on-board software (500 000 lines de code)
Average figures: for critical systems, ≥ 70%
Cost of fault removal all the more high as it is late

Relative cost of fault removal

<table>
<thead>
<tr>
<th>Phase</th>
<th>1976 (Boehm)</th>
<th>1995 (Baziuk)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements &amp; specification</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Design</td>
<td>5 - 7.5</td>
<td></td>
</tr>
<tr>
<td>Implementation</td>
<td>8 - 15</td>
<td>90 - 440</td>
</tr>
<tr>
<td>Integration</td>
<td>10 - 25</td>
<td>90 - 440</td>
</tr>
<tr>
<td>Qualification</td>
<td>25 - 75</td>
<td>440</td>
</tr>
<tr>
<td>Operation/Maintenance</td>
<td>50 - 500</td>
<td>470 - 880</td>
</tr>
</tbody>
</table>

IBM data

Faults introduced by phase

Faults corrected by phase

Cost of fault correction by phase

Fault percentage

$25
$130
$250
$1000
$14,000

Implen testing
Unit testing
Integr. testing
Qualif. testing
Operat. life
### Percentage of erroneous corrections wrt number of removed faults

<table>
<thead>
<tr>
<th>Activity</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functional specs review</td>
<td>1%</td>
</tr>
<tr>
<td>Detailed specs review</td>
<td>2%</td>
</tr>
<tr>
<td>Inspection of component logic</td>
<td>2%</td>
</tr>
<tr>
<td>Inspection of component code</td>
<td>3%</td>
</tr>
<tr>
<td>Unit testing</td>
<td>4%</td>
</tr>
<tr>
<td>Functional testing</td>
<td>6%</td>
</tr>
<tr>
<td>System testing</td>
<td>10%</td>
</tr>
</tbody>
</table>
Typical distribution of software faults uncovered during development

- Incorrect or poorly expressed requirements
- Incorrect or poorly expressed specification
- Design-implementation faults affecting several components
- Design-implementation faults affecting one component
- Typographical fault
- Regression fault
- Others

Percentage of uncovered faults

0 10 20 30 40
Persistent software faults classified by manifestation frequency
(200 faults examined)

<table>
<thead>
<tr>
<th>Category</th>
<th>Projet A (500 K ins.)</th>
<th>Projet B (100 k ins.)</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Omitted logic (existing code too simple)</td>
<td>36</td>
<td>24</td>
<td>60</td>
</tr>
<tr>
<td>2. Non re-initialization of data</td>
<td>17</td>
<td>6</td>
<td>23</td>
</tr>
<tr>
<td>3. Regression fault</td>
<td>5</td>
<td>12</td>
<td>17</td>
</tr>
<tr>
<td>4. Documentation fault (correct software)</td>
<td>10</td>
<td>6</td>
<td>16</td>
</tr>
<tr>
<td>5. Inadequate specification</td>
<td>10</td>
<td>1</td>
<td>11</td>
</tr>
<tr>
<td>6. Faulty binary correction</td>
<td>0</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>7. Erroneous comment</td>
<td>0</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>8. IF instruction too simple</td>
<td>9</td>
<td>2</td>
<td>11</td>
</tr>
<tr>
<td>9. Faulty data reference</td>
<td>6</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>10. Data alignment fault (left bits wrt right bits, etc.)</td>
<td>4</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>11. Timing fault causing data loss</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12. Non-initialization of data</td>
<td>3</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>13. Other categories of lesser importance (total less than or equal to 4)</td>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
</tbody>
</table>
## Fault density

- **Created faults:** 10 à 300 / KLOC
- **Residual faults:** 0.01 à 10 / KLOC

### Example: increments of AT&T ESS-5

<table>
<thead>
<tr>
<th>Size of increment</th>
<th>Density of faults uncovered during development (x)</th>
<th>Density of faults uncovered in the field (y)</th>
<th>Learning factor (y/x)</th>
<th>Type of software</th>
</tr>
</thead>
<tbody>
<tr>
<td>42069</td>
<td>28,5</td>
<td>7,2</td>
<td>0,25</td>
<td>Preventive maintenance</td>
</tr>
<tr>
<td>5422</td>
<td>67,3</td>
<td>21,0</td>
<td>0,31</td>
<td>Billing</td>
</tr>
<tr>
<td>9313</td>
<td>79,3</td>
<td>27,7</td>
<td>0,35</td>
<td>On-line upgrading</td>
</tr>
<tr>
<td>14467</td>
<td>26,5</td>
<td>7,2</td>
<td>0,27</td>
<td>System growth</td>
</tr>
<tr>
<td>165042</td>
<td>101,6</td>
<td>5,3</td>
<td>0,05</td>
<td>Hardware fault recovery</td>
</tr>
<tr>
<td>16504</td>
<td>84,1</td>
<td>2</td>
<td>0,02</td>
<td>Software fault recovery</td>
</tr>
<tr>
<td>38737</td>
<td>149,4</td>
<td>5,8</td>
<td>0,04</td>
<td>System integrity</td>
</tr>
</tbody>
</table>
Design faults (‘errata’) of Intel processors

- Pentium [March 95]
- Pentium Pro [Nov 95]
- Pentium II [May 97]
- Mobile P. II [Aug 98]
- Celeron [Apr 98]
- Xeon [May 98]

Design faults

Processors [date of first update]

Total
Present Jan 99
Fault Forecasting
Estimation of presence, creation and consequences of faults

- Identification, analysis of consequences, and classification of failures
  - Ordinal or qualitative evaluation
- Probabilistic evaluation of the extent to which some dependability attributes are satisfied
  - Probabilistic or quantitative evaluation
    - Modelling
      - Behavior model of system / failure, maintenance actions, solicitations
    - Operational testing
      - Evaluation test according to operational input profile
<table>
<thead>
<tr>
<th>Ordinal evaluation</th>
<th>Probabilistic evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Failure mode and effect (and criticality) analysis [FME(C)A]</td>
<td></td>
</tr>
<tr>
<td>Reliability diagrams</td>
<td></td>
</tr>
<tr>
<td>Fault trees</td>
<td></td>
</tr>
<tr>
<td>State diagrams</td>
<td></td>
</tr>
<tr>
<td>Markov chains</td>
<td></td>
</tr>
<tr>
<td>Petri nets</td>
<td></td>
</tr>
</tbody>
</table>
Failure intensity (average number of failures per unit of time)

- Constant failure intensity
  - Stable reliability
    - Preserved ability to deliver correct service [stochastic equality of times to failure]
    - Stationary stochastic processes
  - Non-stationary stochastic processes

- Decreasing failure intensity
  - Reliability growth
    - Improved ability to deliver correct service [stochastic increase of times to failure]

- Increasing failure intensity
  - Reliability decrease
    - Degraded ability to deliver correct service [stochastic decrease of times to failure]
IBM Data

- Average number of unique faults / 3 months
- Relative month where failure first reported

Graph showing the average number of unique faults over 3 months for different versions of software:
- Version X
- Version X+1
- Version X+2
- Version X+3
- Version X+4
- Before version X

The x-axis represents the relative month where the failure first occurred, ranging from 1 to 35.

Legend:
- Solid grey: Before version X
- Light grey: Version X
- Medium grey: Version X+1
- Dark grey: Version X+2
- Dark grey with lines: Version X+3
- Light grey with lines: Version X+4
AT&T Data

Hardware replacement rate trend

Field data

Packs per 1k lines per month (/50)

Yr 1  Yr 2  Yr 3
Operational testing

Number of executions for

\[ P\{ p \leq p_0 \} \geq 1 - \alpha \]

\[ p : \text{ failure probability per execution} \]
\[ p_0 : \text{ reliability objective} \]
\[ 1 - \alpha : \text{ confidence level} \]

\[ \sum_{j=0}^{f} \binom{j}{N} p_0^j (1 - p_0)^{N-j} \geq \alpha \]

\( f \): number of successive failures

Required effort

- Increasing function of the number of observed failures
- Zero-failure:

  \[
  N \geq \frac{\ln(\alpha)}{\ln(1-p_0)} \]
  \[
  T \geq \frac{\ln(\alpha)}{\lambda_0}
  \]

executions

hours

ReSIST courseware — Jean-Claude Laprie — Fundamentals of Dependability
## Zero failure

### DISCRETE TIME

Number of program executions

<table>
<thead>
<tr>
<th>$p_0$</th>
<th>10-1</th>
<th>10-2</th>
<th>10-3</th>
<th>10-4</th>
</tr>
</thead>
<tbody>
<tr>
<td>10-1</td>
<td>23</td>
<td>46</td>
<td>69</td>
<td>92</td>
</tr>
<tr>
<td>10-2</td>
<td>230</td>
<td>461</td>
<td>691</td>
<td>921</td>
</tr>
<tr>
<td>10-3</td>
<td>2303</td>
<td>4605</td>
<td>6908</td>
<td>9210</td>
</tr>
<tr>
<td>10-4</td>
<td>23026</td>
<td>46052</td>
<td>69078</td>
<td>92103</td>
</tr>
<tr>
<td>10-5</td>
<td>230259</td>
<td>460517</td>
<td>690776</td>
<td>921034</td>
</tr>
<tr>
<td>10-6</td>
<td>2302585</td>
<td>4605170</td>
<td>6907755</td>
<td>9210340</td>
</tr>
</tbody>
</table>

### CONTINUOUS TIME

Program execution duration

<table>
<thead>
<tr>
<th>$\lambda_0$</th>
<th>10-1</th>
<th>10-2</th>
<th>10-3</th>
<th>10-4</th>
</tr>
</thead>
<tbody>
<tr>
<td>10-1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>10-2</td>
<td>10</td>
<td>20</td>
<td>1</td>
<td>1.3</td>
</tr>
<tr>
<td>10-3</td>
<td>3.2</td>
<td>6.4</td>
<td>9.6</td>
<td>1</td>
</tr>
<tr>
<td>10-4</td>
<td>2.6</td>
<td>5.3</td>
<td>7.9</td>
<td>10.5</td>
</tr>
<tr>
<td>10-5</td>
<td>26.2</td>
<td>52.3</td>
<td>78.9</td>
<td>105.1</td>
</tr>
<tr>
<td>10-6</td>
<td>262.8</td>
<td>525.7</td>
<td>788.6</td>
<td>1051.4</td>
</tr>
</tbody>
</table>

### $f$ failures, $p_0 = 10^{-3}$, $\alpha = 10^{-2}$

<table>
<thead>
<tr>
<th>$f$</th>
<th>$N$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6636</td>
</tr>
<tr>
<td>2</td>
<td>8403</td>
</tr>
<tr>
<td>3</td>
<td>10042</td>
</tr>
<tr>
<td>4</td>
<td>11601</td>
</tr>
<tr>
<td>5</td>
<td>13105</td>
</tr>
<tr>
<td>6</td>
<td>14567</td>
</tr>
<tr>
<td>7</td>
<td>15996</td>
</tr>
<tr>
<td>8</td>
<td>17398</td>
</tr>
<tr>
<td>9</td>
<td>18779</td>
</tr>
</tbody>
</table>
Study of the FAA Airworthiness Directives (AD)


Confirmed avionics ADs: 33
   Hardware: 20   Software: 13

Equipments
   Rockwell/Collins
   Bendix/King
   Honeywell/Sperry
   Tracor
   Aerospace

Estimation of software reliability

Failure rate (h⁻¹)

10⁻¹⁰  10⁻⁹  10⁻⁸  10⁻⁷  10⁻⁶

- No software problem reported
- DME (88)
- DME (94)
- TCAS II (91)
- TCAS (94)
- Omega
- ATC Transp.
- Average
Fault Tolerance
Delivering service implementing system function in spite of faults

- Error detection: identification of error presence

- System recovery: transformation of erroneous state in a state free from detected error and from fault that can be activated again

  - Error handling: error removal from system state, if possible before failure occurrence

  - Fault handling: avoiding fault(s) to be activated again
Error detection

Concurrent detection, during service delivery
  Addition of error detection mechanisms in component
  Self-checking component

Preemptive detection: service delivery suspended, search for latent errors and dormant faults

Error handling

Rollback: brings the system back into a state saved prior to error occurrence
  Saved state: recovery point

Rollforward: new state (free from detected error) found

Compensation: erroneous state contains enough redundancy for enabling error masking
Fault handling

- **Diagnostics**: identifies and records the error cause(s), in terms of localisation and category
- **Isolation**: performs physical or logical exclusion of the faulty component(s) from further contribution to service delivery, i.e., makes the fault(s) dormant
- **Reconfiguration**: either switches in spare components or reassigns tasks among non-failed components
- **Reinitialization**: checks, updates and records the new configuration, and updates system tables and records

- **Intermittent faults**
  - Isolation and reconfiguration not necessary
  - **Identification**
    - Error handling
    - Fault diagnosis
  - Non recurrence of error
  - Absence of fault
  - Intermittent fault
Error detection and system recovery or Detection - recovery

Fault masking and system recovery or Masking

Systematic application even in error absence
Effectiveness of error processing: *coverage factor*

\[ c = P\{ \text{service is delivered} \mid \text{failed component} \} \]

Component failure is covered if system error successfully processed

Duplex system, active redundancy

Component failure rate: \( \lambda \)

Repair rate: \( \mu \)

\[ \bar{c} = 1 - c \]

Reliability

\[ R(t) \approx \exp\{ -2 (\bar{c} + \frac{\lambda}{\mu} ) \lambda \ t \} \]

\[ \text{MTTF} \approx \frac{1}{2 (\bar{c} + \frac{\lambda}{\mu} ) \lambda} \]

Unavailability

\[ \bar{A} = 1 - A \approx 2 (\bar{c} + \frac{\lambda}{\mu} ) (\lambda / \mu) \]

\[ \bar{A}_{NR} = \lambda / \mu \]

Equivalent failure rate

\[ \lambda_{eq} = 2 (\bar{c} + \frac{\lambda}{\mu} ) \lambda \]

\[
\begin{align*}
2 \bar{c} \lambda & \quad \rightarrow \quad \text{non-covered failure} \\
2 (\lambda / \mu) \lambda & \quad \rightarrow \quad \text{two successive failures, first failure covered} \\
\Rightarrow & \quad \text{coverage factor influential when } 1-c \sim \lambda / \mu
\end{align*}
\]
\( \frac{\lambda}{\mu} \)

\( R(t) \)

\( c = 0.95 \)
\( c = 0.99 \)
\( c = 0.995 \)
\( c = 0.999 \)
\( c = 1.000 \)

\( \lambda_{\text{eq}} \)

\( \lambda \)

\( \frac{\lambda}{\mu} = 10^{-3} \)

\( 3.6 \text{ j/yr} \)
\( 8.8 \text{ h/yr} \)
\( 53 \text{ min/yr} \)
\( 5 \text{ min/yr} \)
\( 30 \text{ sec/yr} \)
\( 3 \text{ sec/yr} \)

\( \lambda \)
Experimental data

Systems
Processors 9000
Disks 25500
MTBF System 21 years

MTBF (yrs)

450
400
350
300
250
200
150
100
50
0

Maintenance
Environment
Hardware
Operations
Software
Total

Number of reported failures

0 50 100 150 200

1 2 3 4 5 6 7 8

Length of fault chains

Length of fault chains having led to failure

ReSIST courseware — Jean-Claude Laprie — Fundamentals of Dependability
**Distributed system:** set of processors (processing and storage) communicating by messages via communication network

Hierarchy of fault tolerant protocols according to relation "uses service"

<table>
<thead>
<tr>
<th>Group membership</th>
<th>Common mechanism: consensus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atomic multicast</td>
<td></td>
</tr>
<tr>
<td>Clock synchronization</td>
<td></td>
</tr>
</tbody>
</table>
Fail-silent processors

- Implies self-checking processors
- Consequences:
  1) Only correct value messages are sent
    - minimum number of processors for consensus in presence of t faults: $n \geq t+2$
  2) Error detection: halting detection by interrogating and waiting
  3) Resource replication for tolerating t faults ("halts"): $\geq t+1$
  4) Communication network saturation by “babbling” is impossible
  5) Network architecture:
Arbitrary failure processors

Induced complications:

1) Possibility of non consistent ("byzantine") behavior
   - minimum number of processors for consensus in presence of t faults: \( n \geq 3 \cdot t + 1 \)
   - \( t + 1 \) exchanges of messages
2) Error detection ≠ halt detection
3) Resource replication for tolerating t faults ≥ 2 \( \cdot t + 1 \)
4) Communication network saturation by “babbling” possible
5) Possibility of "lying" on source address
6) Network architecture:
### Private values

<table>
<thead>
<tr>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

### After 1st exchange

<table>
<thead>
<tr>
<th>Origin</th>
<th>P1P2 P3 P4</th>
<th>P1P2 P3 P4</th>
<th>P1P2 P3 P4</th>
<th>P1P2 P3 P4</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) 2 3 9</td>
<td>1 (2) 3 4</td>
<td>1 2 (3) 9</td>
<td>X X X (4)</td>
<td></td>
</tr>
</tbody>
</table>

**Broadcasting private values**

### After 2nd exchange

<table>
<thead>
<tr>
<th>From P1</th>
<th>From P2</th>
<th>From P3</th>
<th>From P4</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1 2 3 9)</td>
<td>— — 3 4</td>
<td>— 2 — 9</td>
<td>— 5 6 —</td>
</tr>
</tbody>
</table>

**Broadcasting values received from their processors**

<table>
<thead>
<tr>
<th>From P1</th>
<th>From P2</th>
<th>From P3</th>
<th>From P4</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1 2 3 9)</td>
<td>(1 2 3 4)</td>
<td>(1 2 3 9)</td>
<td>(X X X 4)</td>
</tr>
</tbody>
</table>

### After majority vote

<table>
<thead>
<tr>
<th>P1P2 P3 P4</th>
<th>P1P2 P3 P4</th>
<th>P1P2 P3 P4</th>
<th>P1P2 P3 P4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 9</td>
<td>1 2 3 9</td>
<td>1 2 3 9</td>
<td>X X X X</td>
</tr>
</tbody>
</table>

---

*ReSIST courseware — Jean-Claude Laprie — Fundamentals of Dependability*
Practical aspects

- Performance overhead due to volume of exchanged data
  - $t=1 \rightarrow 9$ values received by each processor
  - $t=2 \rightarrow 156$ values received by each processor

- Use of signatures
  Signatures attached to exchanged data such that corruption probability without detection is negligible
  - $2t+1$ processors and communication channels
  - $t+1$ exchanges
  - Reduction of volume of data exchanged
    - $t=1 \rightarrow 4$ values received by each processor
    - $t=2 \rightarrow 40$ values received by each processor
Other utilization: data with likely variations (e.g., from sensors), median vote
Development fault tolerance

Error detection (defensive programming) and exception handling

Error detection and recovery points

Design diversity

Double programming

Recovery blocks

N-Version programming

N-self-checking programming

Two-fold diversity + comparison

Two-fold diversity + acceptance test

Three-fold diversity + vote

Four-fold diversity + switching

Soft faults

Solid faults

Service continuity

Prevention of error propagation

Fail-fast

Solid faults

Soft faults

Design diversity

Error detection (defensive programming) and exception handling

Fail-fast

Error detection

Error detection and recovery points

Recovery blocks

N-Version programming

N-self-checking programming

Design diversity

Recovery blocks

N-Version programming

N-self-checking programming

Double programming

Recovery blocks

N-Version programming

N-self-checking programming

Design diversity

Error detection (defensive programming) and exception handling

Fail-fast

Error detection

Error detection and recovery points

Recovery blocks

N-Version programming

N-self-checking programming

Design diversity

Two-fold diversity + comparison

Two-fold diversity + acceptance test

Three-fold diversity + vote

Four-fold diversity + switching

Soft faults

Solid faults

Service continuity

Development fault tolerance

Error detection (defensive programming) and exception handling

Error detection and recovery points

Design diversity

Double programming

Recovery blocks

N-Version programming

N-self-checking programming

Two-fold diversity + comparison

Two-fold diversity + acceptance test

Three-fold diversity + vote

Four-fold diversity + switching

Soft faults

Solid faults

Service continuity

Prevention of error propagation

Fail-fast

Solid faults

Soft faults

Design diversity

Error detection (defensive programming) and exception handling

Fail-fast

Error detection

Error detection and recovery points

Recovery blocks

N-Version programming

N-self-checking programming

Design diversity

Two-fold diversity + comparison

Two-fold diversity + acceptance test

Three-fold diversity + vote

Four-fold diversity + switching

Soft faults

Solid faults

Service continuity

Development fault tolerance

Error detection (defensive programming) and exception handling

Error detection and recovery points

Design diversity

Double programming

Recovery blocks

N-Version programming

N-self-checking programming

Two-fold diversity + comparison

Two-fold diversity + acceptance test

Three-fold diversity + vote

Four-fold diversity + switching

Soft faults

Solid faults

Service continuity

Prevention of error propagation

Fail-fast

Solid faults

Soft faults

Design diversity

Error detection (defensive programming) and exception handling

Fail-fast

Error detection

Error detection and recovery points

Recovery blocks

N-Version programming

N-self-checking programming

Design diversity

Two-fold diversity + comparison

Two-fold diversity + acceptance test

Three-fold diversity + vote

Four-fold diversity + switching

Soft faults

Solid faults

Service continuity

Development fault tolerance

Error detection (defensive programming) and exception handling

Error detection and recovery points

Design diversity

Double programming

Recovery blocks

N-Version programming

N-self-checking programming

Two-fold diversity + comparison

Two-fold diversity + acceptance test

Three-fold diversity + vote

Four-fold diversity + switching

Soft faults

Solid faults

Service continuity

Prevention of error propagation

Fail-fast

Solid faults

Soft faults

Design diversity

Error detection (defensive programming) and exception handling

Fail-fast

Error detection

Error detection and recovery points

Recovery blocks

N-Version programming

N-self-checking programming

Design diversity

Two-fold diversity + comparison

Two-fold diversity + acceptance test

Three-fold diversity + vote

Four-fold diversity + switching

Soft faults

Solid faults

Service continuity

Development fault tolerance

Error detection (defensive programming) and exception handling

Error detection and recovery points

Design diversity

Double programming

Recovery blocks

N-Version programming

N-self-checking programming

Two-fold diversity + comparison

Two-fold diversity + acceptance test

Three-fold diversity + vote

Four-fold diversity + switching

Soft faults

Solid faults

Service continuity

Prevention of error propagation

Fail-fast

Solid faults

Soft faults

Design diversity

Error detection (defensive programming) and exception handling

Fail-fast

Error detection

Error detection and recovery points

Recovery blocks

N-Version programming

N-self-checking programming

Design diversity

Two-fold diversity + comparison

Two-fold diversity + acceptance test

Three-fold diversity + vote

Four-fold diversity + switching

Soft faults

Solid faults

Service continuity

Development fault tolerance

Error detection (defensive programming) and exception handling

Error detection and recovery points

Design diversity

Double programming

Recovery blocks

N-Version programming

N-self-checking programming

Two-fold diversity + comparison

Two-fold diversity + acceptance test

Three-fold diversity + vote

Four-fold diversity + switching

Soft faults

Solid faults

Service continuity

Prevention of error propagation

Fail-fast

Solid faults

Soft faults

Design diversity

Error detection (defensive programming) and exception handling

Fail-fast

Error detection

Error detection and recovery points

Recovery blocks

N-Version programming

N-self-checking programming

Design diversity

Two-fold diversity + comparison

Two-fold diversity + acceptance test

Three-fold diversity + vote

Four-fold diversity + switching

Soft faults

Solid faults

Service continuity

Development fault tolerance

Error detection (defensive programming) and exception handling

Error detection and recovery points

Design diversity

Double programming

Recovery blocks

N-Version programming

N-self-checking programming

Two-fold diversity + comparison

Two-fold diversity + acceptance test

Three-fold diversity + vote

Four-fold diversity + switching

Soft faults

Solid faults

Service continuity

Prevention of error propagation

Fail-fast

Solid faults

Soft faults

Design diversity

Error detection (defensive programming) and exception handling

Fail-fast

Error detection

Error detection and recovery points

Recovery blocks

N-Version programming

N-self-checking programming

Design diversity

Two-fold diversity + comparison

Two-fold diversity + acceptance test

Three-fold diversity + vote

Four-fold diversity + switching

Soft faults

Solid faults

Service continuity

Development fault tolerance

Error detection (defensive programming) and exception handling

Error detection and recovery points

Design diversity

Double programming

Recovery blocks

N-Version programming

N-self-checking programming

Two-fold diversity + comparison

Two-fold diversity + acceptance test

Three-fold diversity + vote

Four-fold diversity + switching

Soft faults

Solid faults

Service continuity

Prevention of error propagation

Fail-fast

Solid faults

Soft faults

Design diversity

Error detection (defensive programming) and exception handling

Fail-fast

Error detection

Error detection and recovery points

Recovery blocks

N-Version programming

N-self-checking programming

Design diversity

Two-fold diversity + comparison

Two-fold diversity + acceptance test

Three-fold diversity + vote

Four-fold diversity + switching

Soft faults

Solid faults

Service continuity

Development fault tolerance

Error detection (defensive programming) and exception handling

Error detection and recovery points

Design diversity

Double programming

Recovery blocks

N-Version programming

N-self-checking programming

Two-fold diversity + comparison

Two-fold diversity + acceptance test

Three-fold diversity + vote

Four-fold diversity + switching

Soft faults

Solid faults

Service continuity

Prevention of error propagation

Fail-fast

Solid faults

Soft faults

Design diversity

Error detection (defensive programming) and exception handling

Fail-fast

Error detection

Error detection and recovery points

Recovery blocks

N-Version programming

N-self-checking programming

Design diversity

Two-fold diversity + comparison

Two-fold diversity + acceptance test

Three-fold diversity + vote

Four-fold diversity + switching

Soft faults

Solid faults

Service continuity

Development fault tolerance

Error detection (defensive programming) and exception handling

Error detection and recovery points

Design diversity

Double programming

Recovery blocks

N-Version programming

N-self-checking programming

Two-fold diversity + comparison

Two-fold diversity + acceptance test

Three-fold diversity + vote

Four-fold diversity + switching

Soft faults

Solid faults

Service continuity

Prevention of error propagation

Fail-fast

Solid faults

Soft faults

Design diversity

Error detection (defensive programming) and exception handling

Fail-fast

Error detection

Error detection and recovery points

Recovery blocks

N-Version programming

N-self-checking programming

Design diversity

Two-fold diversity + comparison

Two-fold diversity + acceptance test

Three-fold diversity + vote

Four-fold diversity + switching

Soft faults

Solid faults

Service continuity

Development fault tolerance

Error detection (defensive programming) and exception handling

Error detection and recovery points

Design diversity

Double programming

Recovery blocks

N-Version programming

N-self-checking programming

Two-fold diversity + comparison

Two-fold diversity + acceptance test

Three-fold diversity + vote

Four-fold diversity + switching

Soft faults

Solid faults

Service continuity

Prevention of error propagation

Fail-fast

Solid faults

Soft faults

Design diversity

Error detection (defensive programming) and exception handling

Fail-fast

Error detection

Error detection and recovery points

Recovery blocks

N-Version programming

N-self-checking programming

Design diversity

Two-fold diversity + comparison

Two-fold diversity + accepta...
Examples of recovery points

Tandem process pairs

Statistics on 169 software failures

1 processor halted $\rightarrow$ 138
Several processors halted $\rightarrow$ 31

efficiency: 82% (138/169)
* does not account for under-reporting

Causes of multiple processor halts

2nd halt: processor executing the backup of failed primary
same fault primary-backup
different fault
unclear
Not process-pair related
Unclear
Design diversity

➤ Aim: failure independency
  ➤ Obstacles: common specification, common difficulties, inter-variant synchronizations and decisions

➤ Operational use
  ➤ Civil avionics: generalized, at differing levels
  ➤ Railway signalling: partial
  ➤ Nuclear control: partial

➤ Dependability improvement
  ➤ Gain, although less than for physical fault tolerance
  ➤ Contribution to verification of specification
Sensor management in a redundant inertia platform

20 programmes (1600 to 5000 lines)

920746 tests on flight simulator

Average failure probab 1 version $5.3 \times 10^{-3}$

Average failure probab 3 -version systems $4.2 \times 10^{-4}$

Reliability improvement : 13
Architecture of Airbus computers

- 28V
- Power
- Memory
- Processor
- I/O
- Control channel
- Monitoring channel
- Protection: lightning, EMI, over/under voltage
Airbus A-320

ReSIST courseware — Jean-Claude Laprie — Fundamentals of Dependability
Verification and evaluation of fault tolerance

Faults (deficiencies) in algorithms and mechanisms of fault tolerance

Fault tolerance Coverage

Evaluation / Influence

Fault forecasting

Modelling

Test

Fault injection

Improvement

Fault removal

Dynamic verification

Static verification

Test

Model checking
## Fault injection

A fault injection process involves injecting faults into a system to test its error detection and handling mechanisms. The diagram illustrates the flow of inputs, activity, and outputs in relation to the target system. The process is categorized into two main types: physical and informational.

### Physical Injection

- **By hardware**
  - Radiations
  - Interferences
  - Pins

### Informational Injection

- **In simulation**
  - By software
    - Memory
    - Executive
    - Processor

### Target system

<table>
<thead>
<tr>
<th>Injection</th>
<th>Simulation model</th>
<th>Prototype or actual system</th>
</tr>
</thead>
</table>
| Informational | In simulation | By software
  - Memory
  - Executive
  - Processor |
| Physical | By hardware | By hardware
  - Radiations
  - Interferences
  - Pins |

- **Representativity of faults**

---

ReSIST courseware — Jean-Claude Laprie — Fundamentals of Dependability
Delta-4 system (LAAS)

Faults --> 94% --> Errors --> 85% --> Detected errors --> 99% --> Tolerated errors

3% --> 1%

Robustness testing of POSIX calls (CMU)

Test of Chorus Classix R3 (LAAS)

<table>
<thead>
<tr>
<th>System</th>
<th>Failure probability</th>
<th>% responses</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIX 4.1</td>
<td>0.1</td>
<td>20</td>
</tr>
<tr>
<td>Digital Unix 4.0</td>
<td>0.15</td>
<td>30</td>
</tr>
<tr>
<td>Free BSD 2.2.5</td>
<td>0.2</td>
<td>40</td>
</tr>
<tr>
<td>Irix 6.2</td>
<td>0.25</td>
<td>50</td>
</tr>
<tr>
<td>HP UX B.10.20</td>
<td>0.15</td>
<td>30</td>
</tr>
<tr>
<td>Linux 2.0.18</td>
<td>0.1</td>
<td>20</td>
</tr>
<tr>
<td>LynxOS 2.4.0</td>
<td>0.15</td>
<td>30</td>
</tr>
<tr>
<td>NetBSD 1.3</td>
<td>0.2</td>
<td>40</td>
</tr>
<tr>
<td>QNX 4.2.4</td>
<td>0.25</td>
<td>50</td>
</tr>
<tr>
<td>SunOS 5.5</td>
<td>0.15</td>
<td>30</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Error status</th>
<th>Internal faults</th>
<th>External faults</th>
</tr>
</thead>
<tbody>
<tr>
<td>No obs</td>
<td>Average</td>
<td>Min-Max</td>
</tr>
<tr>
<td>Error</td>
<td>Excep.</td>
<td></td>
</tr>
<tr>
<td>status</td>
<td>Kernel</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Syst. hang</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Appl. hang</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Incor. res.</td>
<td></td>
</tr>
</tbody>
</table>

ReSIST courseware — Jean-Claude Laprie — Fundamentals of Dependability
Development of Dependable Systems
Development steps

Requirements and specification

Design

Realisation (including selection and adaptation of pre-existing components)

Integration

Qualification

Means for dependability

Fault Prevention

Fault Tolerance

Fault Removal

Fault Forecasting
V development model

Specification → Qualification

Architectural design → Integration

Detailed design → Component realisation

Creation

Verification
Effort distribution

<table>
<thead>
<tr>
<th>Years</th>
<th>Requirements and specification</th>
<th>Architectural design</th>
<th>Detailed design</th>
<th>Component realisation</th>
<th>Integration</th>
<th>Qualification</th>
</tr>
</thead>
<tbody>
<tr>
<td>1960-1970</td>
<td>10%</td>
<td></td>
<td>80%</td>
<td>10%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1980</td>
<td>20%</td>
<td>60%</td>
<td></td>
<td>20%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1990</td>
<td>40%</td>
<td>30%</td>
<td></td>
<td>30%</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Techniques and approaches ↔ System criticitality

Risk

Failure likelihood

Possible consequences of failures

Standards

- CEI 61508 (1998) – Combination of a damage probability and of its severity
- ISO/CEI 15026 (1998) – A function of the probability of occurrence of a given threat and the potential adverse consequences of that threat’s occurrence
- MIL-STD-882D (February 2000) – An expression of the impact and possibility of a mishap in terms of potential mishap severity and probability of occurrence
<table>
<thead>
<tr>
<th>Probability of occurrence</th>
<th>Minor</th>
<th>Significant</th>
<th>Major</th>
<th>Catastrophic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequent</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Probable</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Occasional</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Infrequent</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Improbable</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Probabilities of occurrence and severities of consequences

Application domains (transportation, energy production, telecommunications, etc.)
Examples of occurrence probabilities

<table>
<thead>
<tr>
<th>Probability of occurrence</th>
<th>Continuous time (h⁻¹)</th>
<th>Discrete time (by solicitation)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequent</td>
<td>&gt; 10⁻⁴</td>
<td>&gt; 10⁻¹</td>
</tr>
<tr>
<td>Probable</td>
<td>10⁻⁴ - 10⁻⁵</td>
<td>10⁻¹ - 10⁻²</td>
</tr>
<tr>
<td>Occasional</td>
<td>10⁻⁵ - 10⁻⁶</td>
<td>10⁻² - 10⁻³</td>
</tr>
<tr>
<td>Infrequent</td>
<td>10⁻⁶ - 10⁻⁸</td>
<td>10⁻³ - 10⁻⁴</td>
</tr>
<tr>
<td>Improbable</td>
<td>10⁻⁸ - 10⁻¹⁰</td>
<td>10⁻⁴ - 10⁻⁶</td>
</tr>
</tbody>
</table>
## Example of damage severity (civil avionics)

<table>
<thead>
<tr>
<th>Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Catastrophic</strong></td>
<td>Failure conditions which would prevent continued safe flight and landing</td>
</tr>
</tbody>
</table>
| **Hazardous / severe-major** | Failure conditions which would reduce the capability of the aircraft or the ability of the crew to cope with adverse operating conditions to the extent that there would be:  
  - A large reduction in safety margins or functional capabilities;  
  - Physical distress or higher workload such that the flight crew could not be relied on to perform their tasks accurately or completely; or  
  - Serious or fatal injury to a relatively small number of the occupants. |
| **Major** | Failure conditions which would reduce the capability of the aircraft or the ability of the crew to cope with adverse operating conditions to the extent that there would be, for example, a significant reduction in safety margins or functional capabilities, a significant increase in crew workload or in conditions impairing crew efficiency, or some discomfort to occupants. |
| **Minor** | Failure conditions which would not significantly reduce aircraft safety, and which involve crew actions that are well within their capabilities. Minor failure conditions include, for example, a slight increase in crew workload, such as routine flight plan changes or some inconvenience to occupants. |
| **No effect** | Failure conditions which do not effect the operational capability of the aircraft or increase pilot workload. |
### Risk classes

<table>
<thead>
<tr>
<th>Probability of occurrence</th>
<th>Severity</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Minor</td>
</tr>
<tr>
<td>Frequent</td>
<td></td>
</tr>
<tr>
<td>Probable</td>
<td></td>
</tr>
<tr>
<td>Occasional</td>
<td>Negligible</td>
</tr>
<tr>
<td>Infrequent</td>
<td></td>
</tr>
<tr>
<td>Improbable</td>
<td></td>
</tr>
</tbody>
</table>

- **Negligible**: Tolerable
- **Undesirable**: Undesirable
- **Unacceptable**: Unacceptable
### Risk classes → Criticality

<table>
<thead>
<tr>
<th>Probability of occurrence</th>
<th>Minor</th>
<th>Significant</th>
<th>Major</th>
<th>Catastrophic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequent</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Probable</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Occasional</td>
<td></td>
<td></td>
<td></td>
<td>Intermediate</td>
</tr>
<tr>
<td>Infrequent</td>
<td></td>
<td></td>
<td></td>
<td>Low</td>
</tr>
<tr>
<td>Improbable</td>
<td></td>
<td></td>
<td></td>
<td>Trivial</td>
</tr>
</tbody>
</table>

**Severity**
- Minor
- Significant
- Major
- Catastrophic

**Criticality**
- Low
- Intermediate
- High
Utilisation of techniques and approaches

M : Mandatory

SR : Strongly recommended. If technique or approach not used, reasons have to be made explicit

R : Recommended

— : No recommendation in favor or against

NR : Explicitly not recommended. If technique or approach used, reasons have to be made explicit
**Actor independency**

Criticality

- **Trivial**
  - Designer/Implementer
  - Verifier
  - Verifier

- **Low**
  - Designer/Implementer
  - Verifier
  - Verifier

- **Intermediate and High**
  - Designer/Implementer
  - Verifier
  - Verifier

Can be the same person(s) Can belong to the same company
From Dependability to Resilience
Resilience

- in dependability and security of computing systems

- Adjective Resilient
  - In use for 30+ years
  - Recently, escalating use ➔ buzzword
  - Used essentially as synonym to fault tolerant

  «The two key attributes here are dependability and robustness. [...] A computing system can be said to be *robust* if it retains its ability to deliver service in conditions which are beyond its normal domain of operation»

- Fault and change tolerance

- in other domains
  - Material science
  - Social psychology
  - Child psychiatry and psychology
  - Ecology
  - Business
  - Industrial safety

Adaptation to changes, and getting back after a setback
At stake: Maintain dependability in spite of changes

Dependability: Ability to deliver service that can justifiably be trusted

Resilience: persistence of service delivery that can justifiably be trusted, when facing changes

Nature
- Functional
- Environmental
- Technological

Prospect
- Foreseen
- Foreseeable
- Unforeseen

Timing
- Short term
- Medium term
- Long term

Threat evolution

The definition does not exclude the possibility of failure

Alternate definition of dependability:
Ability to avoid unacceptably frequent or severe service failures

Resilience: persistence of avoidance of unacceptably frequent or severe service failures, when facing changes

Resilience: persistence of dependability when facing changes
Changes

Nature
- Functional
- Environmental
- Technological

Prospect
- Foreseen, as in new versioning
- Foreseeable, as in the advent of new hardware platforms
- Unforeseen, as drastic changes in service requests or new types of threats

Timing
- Short term, e.g., seconds to hours, as in dynamically changing systems (spontaneous, or ‘ad-hoc’, networks of mobile nodes and sensors, etc.)
- Medium term, e.g., hours to months, as in new versioning or reconfigurations
- Long term, e.g., months to years, as in reorganizations resulting from merging of systems in company acquisitions, or from coupling of systems in military coalitions

Threat evolution, e.g.,
- ever increasing proportion of transient hardware faults,
- ever-evolving and growing attacks,
- mismatches between the modifications that implement the changes and the former status of the system
Context: emerging **ubiquitous systems**

- immense information systems, perhaps involving everything from super-computers, huge server "farms" and giant data centers, to myriads of small mobile computers (billions) and tiny embedded devices (trillions)

Characteristic: perpetual **evolution**

- Development: dynamic componentization via service discovery and assembly
Technologies for resilience

Evolutionary changes → Evolvability
  ➡ Adaptation

Trusted service → Assessability
  ➡ Verification and evaluation

Ubiquitous systems → Usability
  ➡ Human and system users

Complex systems → Diversity
  ➡ Taking advantage of existing diversity for avoiding single points of failure, and augmenting diversity

Means for dependability

- Fault Prevention
- Fault Tolerance
- Fault Removal
- Fault Forecasting

Evolvability | Assessability | Usability | Diversity
---|---|---|---
- | - | - | -
- | - | - | -
- | - | - | -
- | - | - | -
Conclusion
dependency
vanishing substitutes for computers

integration

funnel factor
decreasing natural robustness

unavoidability of faults
ill-mastered complexity

performance
References

(in addition to those mentioned in the slides)
