Software Dependability

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

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

<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>0.99999</td>
<td>5min 15s</td>
</tr>
<tr>
<td>0.9999</td>
<td>52min 34s</td>
</tr>
<tr>
<td>0.999</td>
<td>8h 46min</td>
</tr>
<tr>
<td>0.99</td>
<td>3d 16h</td>
</tr>
</tbody>
</table>

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

• I guasti sistematici del software sono dovuti a errori di progetto o programmazione. Sono i cosiddetti “Bachi” (Bugs)
• *(Bohrbugs vs. Heisenbugs)*
• Se ne può valutare l’incidenza statistica sul software installato, ma non prevederne l’occorrenza

• Software safety - DEF-STAN 00-55:
• “Where safety is dependent on the safety related software (SRS) fully meeting its requirements, demonstrating safety is equivalent to demonstrating correctness with respect to the Software Requirement”. 
Software Dependability Assessment

Difficulties

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

• What is software dependability
  → dependability measures?
  • Number of faults, fault density, complexity?
  • MTTF, failure intensity, failure rate?
How to measure software dependability?

Static measures

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
...

(by Karama Kanoun)
Static measures
Software faults metrics

- Number of faults
- Fault density
  - number of faults for KLOC
  - number of faults for FP

→ a posteriori measures
→ Prediction possible only under the assumption that similar systems show the same fault density.
→ similar = produced with the same development process ??
Software faults metrics

Some figures from safety critical domains:

Automotive domain

• 2 .. 3 bugs per KLOC

• Typical car: 100MLOC \rightarrow 200Kbugs
## Avionics

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

(Reproduced from McCall 87)
How many „Bugs“ do we have to expect?

- Typical production type SW has 1 ... 10 bugs per 1.000 lines of code (LOC).
- Very mature, long-term, well proven software: 0.5 bugs per 1.000 LOC
- Highest software quality ever reported:
  - Less than 1 bug per 10.000 LOC
  - At cost of more than 1.000 US$ per LoC (1977)
  - US Space Shuttle with 3 m LOC costing 3b US$ (out of 12b$ total R&D)
  - Cost level not typical for the railway sector (< 100€/LoC)

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

**Question:** What happens to bug counts over the SW life cycle?
FIGURE A. The amount of flight code that is flown to land spacecraft on Mars has grown exponentially in the last 36 years. Its Compound Annual Growth Rate comes out at roughly 1.20—close to the median value of 1.16 from previous columns.
Fig. 2. Propagation of residual defects (bugs) as a result of defect insertion and defect removal rates during several stages of the software production process, according to a NASA research on high assurance software for flight control units for each 1000 lines of code (TLOC) [12].
## Business vs. safety-critical

<table>
<thead>
<tr>
<th></th>
<th>fault density (at 100 LOC per FP)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(Fault per KLOC)</td>
</tr>
<tr>
<td></td>
<td>(Fault per FP)</td>
</tr>
<tr>
<td>Automotive</td>
<td>3</td>
</tr>
<tr>
<td>Aviation, av.</td>
<td>9,7</td>
</tr>
<tr>
<td>Typical low</td>
<td>1</td>
</tr>
<tr>
<td>Typical high</td>
<td>10</td>
</tr>
<tr>
<td>Mature</td>
<td>0,5</td>
</tr>
<tr>
<td>Shuttle</td>
<td>0,1</td>
</tr>
<tr>
<td>Agile 1</td>
<td>1,28</td>
</tr>
<tr>
<td>Agile 2</td>
<td>4,8</td>
</tr>
<tr>
<td>Agile 3</td>
<td>22</td>
</tr>
<tr>
<td>Traditional, best</td>
<td>200</td>
</tr>
<tr>
<td>Traditional, average</td>
<td>450</td>
</tr>
</tbody>
</table>

Business ref.: [www.leanagilepartners.com/publications.html](http://www.leanagilepartners.com/publications.html)
Lowering fault density

- Improving the development process helps to lower fault density.
- **Software Process Improvements**
  - **Motorola** (Arlington Heights), mix of methods:
    - Fault density reduction = 50% within 3.5 years
  - **Raytheon** (Electronic Systems), CMM:
    - Rework cost divided by 2 after two years of experience
    - Productivity increase = 190%
    - Product quality: multiplied by 4
- **Tradeoff between development (and verification) costs w.r.t. fault density reduction**
Lowering fault density

safety-critical domain

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

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

\[ R(t) = e^{-\lambda t} \]

• MTTF - Mean Time To Failure

\[ MTTF = \frac{1}{\lambda} \]

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

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

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

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

• Using a model to predict failure intensity, MTTF, or residual failure number on the basis of collected data
Conceptual View of Prediction

(by Lionel Briand)
Cumulative failures

Increasing TTF --> Reliability growth

(by Lionel Briand)
Software reliability models

• Assumptions
  – At time 0 there are N faults
  – Every fault is independent from the others and has the same probability of causing a failure during the testing phase
  – Every fault detected during testing is removed in null time

• Jelinski-Moranda model
  – Failure intensity at a given time is proportional to the number of residual faults at that time:

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

• Musa model
  – Failure intensity depending on execution time:

\[ Z(t) = \phi \cdot f \cdot (N - n_t) \]
... criteria for stopping testing

(by Lionel Briand)
Software Reliability Models

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

Pro’s
• Reliability can be specified
• Objective and direct assessment of reliability
• Prediction of time to delivery

Con’s
• Usage model may be difficult to devise
• Selection of reliability growth model difficult
• Measurement of time crucial but may be difficult in some contexts
• Not easily applicable to safety-critical systems as very high reliability requirements would take years of testing to verify.
Ultra High Reliability / Safety Prediction

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

Ultra high reliability demands can be not testable or demonstrable.

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

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

Safety critical Systems often require reliability figures of $10^{-7}$ to $10^{-9}$ FPH.
Software Reliability Models:

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

- Random distribution of input data to a long-running system
- Resource consumption phenomena in a long-running system

- *In both cases:*
  - Collecting data on intervals between failures
  - Estimating future MTTF
  - (assuming exponential failure distribution)
  - Refining estimates with time
  - Maintenance:
    - Periodic maintenance with $T < \text{MTTF}$
    - Just shutdown & restart at failure:
      \[
      \text{Availability} = \frac{\text{MTTF}}{\text{MTTF} + \text{Restart}\_\text{time}}
      \]
  - Possibility of accelerated operation in simulation runs