Fault Tolerant Computing

CS 530

Software Reliability: Static Factors

Yashwant K. Malaiya

Colorado State University
Wholistic Engineering for Software Reliability

Outline

• Techniques available in Software Reliability
• Software & Hardware Reliability
• Defect density & factors that control it
  ▪ Phase
  ▪ Programming team and process maturity
  ▪ Software Structure
  ▪ Requirement volatility
Time to go Wholistic

• We have data on different aspects of reliability to have reasonable hypotheses.
• We know limitations of the hypotheses.
• We have enough techniques & tools to start engineering.
• Accuracy **comparable or better** than established hardware reliability methods.
Why It’s Needed Now

- Reliability expectations growing fast
- Large projects, little time
- Quick changes in developing environments
- Reliance on a single technique not enough
- Pioneering work has already been done.
Why It’s Time: Emergence of SRE

• **Craft**: incremental intuitive refinement
• **Science**: *why* it is so
  - Observe, hypothesize, assess accuracy
• **Engineering**: *how* to get what we want
  - Approximate, integrate, evaluate
• Are we ready to engineer software reliability?
Learning from Hardware Reliability

• Hardware Reliability Methods: Well known, well established methods
  ▪ Now standard practice
  ▪ Used by government and industrial organizations worldwide
  ▪ Considered a well established science
Hardware Reliability: The Status (1)

• Earliest tube computers: MTTF comparable to some computation times!
• 1956 RCA TR-1100: component failure rate models
• 1959: MIL-HDBK-217A: common failure rate: $0.4 \times 10^{-6}$ for all ICs for all cases
• Revised about every 7 years
Hardware Reliability: The Status (2)

• Why use hardware reliability prediction?
  ▪ Feasibility Study: initial design
  ▪ Compare Design Alternatives: Reliability along with performance and cost
  ▪ Find Likely Problem Spots- high contributors to the product failure rate
  ▪ Track Reliability Improvements
Hardware vs Software Faults

- Hardware faults are generally field or manufacturing process defects.
- Software faults are due to incorrect design/implementation ("man-made").
- During debugging, bugs are removed thus reliability grows.
- Design defects on hardware are basically similar to software defects.
Hardware vs Software Reliability Methods: Use of models

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Model selection based on</th>
<th>Parameters estimated using</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Past experience with similar units</td>
<td>Past experience with similar units</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Software</th>
<th>Model selection based on</th>
<th>Parameters estimated using</th>
</tr>
</thead>
</table>
|          | Past experience* with similar units | Early: past experience with similar units  
|          |                           | Later: from the unit under test |

* Some researchers have suggested model selected using early test data from the software under test.
Basic Definitions

- **Defect**: requires a corrective action
- **Defect density**: defects per 1000 non-comment source lines (NC LOC).
- **Failure intensity**: rate at which failures are encountered during execution.
- **MTTF** (mean time to failure): inverse of failure intensity.

In this case mean is not taken over time, rather it is an ensemble average.
Basic Definitions (2)

• Reliability
  ▪ \( R(t) = p\{\text{no failures in time } (0,t)\} \)

• Transaction reliability: probability that a single transaction will be executed correctly.

• Test Time: may be measures in CPU time or some measure of testing effort.
Why is Defect Density Important?

- Important measurement of reliability
- Often used as release criteria.
- Typical values of defect density /1000 LOC mentioned in literature:

<table>
<thead>
<tr>
<th>Beginning Of Unit Testing</th>
<th>On Release</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequently Cited in literature</td>
<td>Highly Tested programs</td>
</tr>
<tr>
<td>16</td>
<td>2.0</td>
</tr>
</tbody>
</table>

- Long term trend: tolerable defect density limits have been gradually dropping, i.e. reliability expectations have risen.
Static and Dynamic Modeling

- Reliability at release depends on
  - Initial number of defects (parameter)
  - Effectiveness of defect removal process (parameter)
  - Operating environment

- **Static modeling**: estimate parameters before testing begins
  - Use static data like software size etc.

- **Dynamic modeling**: estimate parameters during testing
  - Record when defects are found etc.
  - *Time* or *coverage* based
What factors control defect density?

• Need to know for
  ▪ static estimation of initial defect density
  ▪ Find room for process improvement

• Static defect density models: The defect density is influenced by a number of factors $f_1, f_2$, etc. The models combine the impact of factors in two ways:
  ▪ Additive (ex: Takahashi-Kamayachi)
    \[ D = a_1 f_1 + a_2 f_2 + a_3 f_3 \ldots \]
  ▪ Multiplicative (ex. MIL-HDBK-217, COCOMO, RADC)
    \[ D = C \cdot f_1(f_1) \cdot f_2(f_2) \cdot f_3(f_3) \ldots \]
A Static Defect Density Model

- Li, Malaiya, Denton (93, 97)

\[ D = C \cdot F_{ph} \cdot F_{pt} \cdot F_{m} \cdot F_{s} \cdot F_{rv} \]

- \( C \) is constant of proportionality, based on prior data, used for calibration.

- Default value of each function \( F_i \) (submodel) is 1.

- Each function \( F_i \) is a function of some measure of the attribute.

Possible factors:
- Phase
- Programming Team
- Process Maturity
- Structure
- Requirement Volatility
Submodel: Phase Factor $F_{ph}$

- The table shows possible values, based on numbers reported in the literature (Musa, Gaffney, Piwowarski et al.)

<table>
<thead>
<tr>
<th>At beginning of phase</th>
<th>Multiplier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unit testing</td>
<td>4</td>
</tr>
<tr>
<td>Subsystem testing</td>
<td>2.5</td>
</tr>
<tr>
<td>System testing</td>
<td>1 (default)</td>
</tr>
<tr>
<td>Operation</td>
<td>0.35</td>
</tr>
</tbody>
</table>

- The values are given to give you an idea of variability. Actual values will depend on specific process.
Submodel: Programming Team Factor $F_{pt}$

- Based on a study by Takahashi, Kamayachi, who found that defect density declines by about 14% per year (up to seven years).

<table>
<thead>
<tr>
<th>Team’s average skill level</th>
<th>Multiplier</th>
</tr>
</thead>
<tbody>
<tr>
<td>High</td>
<td>0.4</td>
</tr>
<tr>
<td>Average</td>
<td>1 (default)</td>
</tr>
<tr>
<td>Low</td>
<td>2.5</td>
</tr>
</tbody>
</table>

- It is agreed that programming team skills have a significant impact. However measuring skill is hard, there are no good quantitative studies.
SEI- Capability Maturity Model

- Software Engineering Institute Capability Maturity Model (will use CMM for SEI-CMM)
- Begun in 1986 from SEI and Mitre
  - framework for government to assess contractors
- Based on
  - Statistical quality control (Deming’s TQM, Juran)
  - Quality management (Crosby)
  - Feedback from industry and government
## SEI Levels

<table>
<thead>
<tr>
<th>SEI Level</th>
<th>Key Feature</th>
<th>How many organizations?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Initial</td>
<td>ad hoc</td>
<td>75%</td>
</tr>
<tr>
<td>2. Repeatable</td>
<td>basic management</td>
<td>15%</td>
</tr>
<tr>
<td>3. Defined</td>
<td>standardized</td>
<td>8%</td>
</tr>
<tr>
<td>4. Managed</td>
<td>quantitative control</td>
<td>1.5%</td>
</tr>
<tr>
<td>5. Optimizing</td>
<td>continuous improvement</td>
<td>Handful (0.5%)</td>
</tr>
</tbody>
</table>

*Estimating software costs: bringing realism to estimating*  
By Capers Jones, 2007
Submodel: Process Maturity Factor $F_m$

- Based on Jones, Keene, Motorola data.

<table>
<thead>
<tr>
<th>SEI CMM Level</th>
<th>Multiplier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 1</td>
<td>1.5</td>
</tr>
<tr>
<td>Level 2</td>
<td>1 (default)</td>
</tr>
<tr>
<td>Level 3</td>
<td>0.4</td>
</tr>
<tr>
<td>Level 4</td>
<td>0.1</td>
</tr>
<tr>
<td>Level 5</td>
<td>0.05</td>
</tr>
</tbody>
</table>
Submodel: Structure Factor $F_s$ (Pt 1)

- Assembly code fraction: assuming assembly has 40% more defects
  - Factor = $1 + 0.4 \times \text{fraction in assembly}$
- **Complexity**: Complex modules are more **fault prone**, but there may be compensating factors, like people being more cautious when implementing them. No conclusive results are available that link measures like **cyclomatic complexity** with **defect density**.
- Note that by definition, defect density is defects divided by **software size**, which itself is a complexity metric. Question is: does adding other complexity metric help? Answer is: there is no compelling evidence.
Submodel: Structure Factor $F_s$ (Pt 2)

- **Module size**: Data from several projects suggest that very small modules have higher defect densities (Fig 1). Note that many projects have a large number of small modules (Distribution in Fig 2)

Submodel: Requirement volatility

Factor $F_{rv}$

- Impact depends on degree of changes and when they occur.
- Most impact when changes occur near the end of testing.
- Malaiya & Denton: ISSRE 99
Using the Defect Density Model

- Calibrate submodels before use using data from a project as similar as possible.
- Constant C can range between 6-20 (Musa).
- Static models are very valuable, but high accuracy is not expected.
- Useful when dynamic test data is not yet significant.
Static Model: Example

\[ D = C \cdot F_{ph} \cdot F_{pt} \cdot F_{m} \cdot F_{s} \cdot F_{rv} \]

• For an organization, \( C \) is between 12 and 16. The team has average skills and SEI maturity level is II. About 20% of code in assembly. Other factors are average (or \textit{same as past projects}).

Estimate defect density at beginning of subsystem test phase.

• Upper estimate = \( 16 \times 2.5 \times 1 \times 1 \times (1 + 0.4 \times 0.20) \times 1 = 43.2/\text{KSLOC} \)

• Lower estimate = \( 12 \times 2.5 \times 1 \times 1 \times (1 + 0.4 \times 0.20) \times 1 = 32.4/\text{KLOC} \)

Here the structure factor is \( 1 + 0.4 \times 0.20 \) because of some assembly code. Factor 2.5 is for the beginning of the subsystem phase.
Static Models: Limitations

- Other multiplicative models like the COCOMO cost estimation model would have similar limitations.
- The parameter values are based on past projects, which may have been somewhat different.
- Calibration will be accurate only if data from somewhat similar projects was used.
- Some factors may be statistically correlated, for example Programming team and Capability Maturity factors.
- Still such models can be very useful at the beginning of projects for planning the test effort.