Wholistic Engineering for Software Reliability

Outline

Why it’s time…
- Demarcating, measuring, counting: definitions
- Science & engineering of reliability growth
- Those pesky residual defects
- Components & systems
- The toolbox

Reliable Software: The Whole Picture

Techniques Available

<table>
<thead>
<tr>
<th>Phase</th>
<th>Controlling</th>
<th>Assessing</th>
<th>Feedback</th>
</tr>
</thead>
<tbody>
<tr>
<td>Development</td>
<td>√</td>
<td>long-term</td>
<td>long-term</td>
</tr>
<tr>
<td>Test</td>
<td>√</td>
<td>√ conventional</td>
<td>√</td>
</tr>
<tr>
<td>Deployment</td>
<td>√</td>
<td>long-term</td>
<td>long-term</td>
</tr>
</tbody>
</table>

- √: Project-specific techniques available
- Constraints: cost, time, functionality, performance
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

- Well known, well established methods
- Now standard practice
- Used by government and industrial org worldwide
- Considered a *hard science* compared with software reliability
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)

- Failure rates predicted often higher by a factor of 2-4, occasionally by an order of magnitude.
- Constant failure-rate, the bathtub curve, the Arrhenius relationship have been questioned.
Hardware Reliability: The Status (3)

- 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 Reliability

<table>
<thead>
<tr>
<th></th>
<th>Models</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Hardware</strong></td>
<td>Past experience with similar units</td>
<td>Past experience with similar units</td>
</tr>
</tbody>
</table>
| **Software** | Past experience* with similar units  | **Early**: past experience with similar units  
|           |                                       | **Later**: from the same unit        |

* Also suggested: from the same unit
Wholistic Engineering for Software Reliability

Outline

• Why it’s time…
• Demarcating, measuring, counting: definitions
• Science & engineering of reliability growth
• Those pesky residual defects
• Components & systems
• The toolbox

Basic Definitions

• Defect: requires a corrective action
• Defect density: defects per 1000 non-comment source lines.
• Failure intensity: rate at which failures are encountered during execution.
• MTTF (mean time to failure): inverse of failure intensity
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.
- Time: may be measures in CPU time or some measure of testing effort.

Limited use in SRE

Why is Defect Density Important?

- Important measurement of reliability
- Often used as release criteria

<table>
<thead>
<tr>
<th>Beginning Of Unit Testing</th>
<th>Frequently Cited</th>
<th>Release</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Frequently Cited</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>2.0</td>
</tr>
</tbody>
</table>
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**:
  - *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)*
    \[ D = C.F_1(f_1).F_2(f_2).F_3(f_3)\ldots \]
A Static Defect Density Model

- Li, Malaiya, Denton (93, 97)

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

- \( C \) is constant of proportionality, based on prior data.
- Default value of each function (submodel) is 1.
- Calibration based on past, similar projects

---

Submodel: Phase Factor \( F_{ph} \)

- Based on 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>
Submodel: Programming Team Factor $F_{pt}$

- Based on Takahashi, Kamayachi. Decline by 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>

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
Level 1 - Initial

- Process is ad hoc, occasionally chaotic
- Few defined processes
- Ineffective planning
- Reaction-driven management
- Unpredictable
- Success due to heroic efforts
- ~80% of software organizations worldwide

Level 2 - Repeatable

- Can repeat earlier successes
- Basic project management processes
  - Track cost, schedule, functionality
- Realistic project commitments based on results of previous projects
- Still has frequent quality problems
- Stable planning and tracking
- ~15% of software organizations worldwide
Level 3 - Defined

- Documented, standardized, and integrated management and engineering activities
- Projects tailor organization’s standard to develop own process
- Software Engineering Process Group (SEPG)
- *Stable foundation for software engineering and management*
- ~5% of software organizations worldwide

Level 4 - Managed

- Detailed measures of process and product quality collected organization wide
- Measures used to evaluate processes and products
- Statistical process control to reduce variance
- *Process is measured and operates within limits*
- < 1% of software organizations worldwide
Level 5 - Optimizing

- Organization focuses on continuous process improvement
- Goal is to prevent defects by analyzing cause
- Cost benefit analysis of new technologies
- Identify and quickly transfer best practices
- Only a handful of organizations worldwide

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 $F_s$

- Assembly code fraction: assuming assembly has 40% more defects
  - Factor = $1 + 0.4 \times \text{fraction in assembly}$
- Module size: research reported at ISSRE 2000!
- Complexity: Complex modules are more fault prone, but there may be compensating factors.

Submodel: Requirement volatility Factor $F_{ph}$

- 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

For an organization, C is between 12 and 16. Average team and SEI maturity level is II. About 20% of code in assembly. Other factors are average (or same as past projects).

Estimate defect density at beginning of subsystem test phase.

- Upper estimate = 16 × 2.5 × 1 × 1 × (1 + 0.4 × 0.2) × 1 = 43.2/KSLOC
- Lower estimate = 12 × 2.5 × 1 × 1 × (1 + 0.4 × 0.2) × 1 = 32.4/KLOC