Fault Tolerant Computing

CS 530

Software Reliability: Static Factors

Yashwant K. Malaiya

Colorado State University
Class Notes

- PSA1 due Tu 10/2/2018
- Project Proposal due Th 10/4/2018
- Midterm Fri 10/12/2018, Fri
  - OC (+ local distance): Here
  - Distance: take-home, due Wed Oct 17
- Coming up
  - Project Progress Report 11/1/2018
  - Project Slides 11/13/2018
  - Final Report 12/5/2018, will use Vericite
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
  ▪ Reuse
Wholistic Engineering for Software Reliability

**Losses from Software Failures (USD)**

1,715,430,778,504

Software Fail Watch: 5th Edition, Tricentis
Year 2017
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 to or better than established hardware reliability methods.
If you build it, they will come

Yeah, I'm just writing the code now.
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></th>
<th>Model selection based on</th>
<th>Parameters estimated using</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>
<tr>
<td><strong>Software</strong></td>
<td>Past experience* with similar units</td>
<td><strong>Early</strong>: past experience with similar units</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Later</strong>: from the unit under test</td>
</tr>
</tbody>
</table>

* Some researchers have suggested model selected using early test data from the software under test.
Bugs May Delay Dbase IV Shipment to Fall, Ashton-Tate Says

By Mark Brownstein and Scott Mace

LOS ANGELES — Ashton-Tate told shareholders last week that the still bug-ridden Dbase IV could miss its projected July shipping date but will definitely be out by September 30.

Blaming the possible delay on undiscovered bugs, Ashton-Tate chairman and CEO Edward J. Esber Jr. said, “We can fix all the bugs that we’ve already found before July,” and he suggested Ashton-Tate could ship Dbase IV in July “if we don’t find any more new bugs.”

“Finding and fixing bugs is the only issue” that could cause further delay, said Roy E. Folk, Ashton-Tate vice president and general manager of software development, at the annual shareholders’ meeting. Folk challenged speculation that the product takes most of the 640K of available RAM under DOS, saying it can run a 100K program on a network.

In fact, Ashton-Tate is fixing bugs by correcting code errors instead of using program patches that add to the program’s size, Folk said.

Esber also spoke of other upcoming Dbase IV versions, including a Presentation Manager release and Dbase IV, Version 1.1, which will take advantage of the Ashton-Tate/Microsoft SQL Server announced in January.

“Dbase IV/PM will feature a graphical user interface,” Esber said. “You will also see further extensions in language and some additional functionality.”

“The direction we’re taking beyond Dbase IV should maintain our lead in the Dbase area,” Esber said.

While chief financial officer George L. Farinsky said Ashton-Tate could weather any sales losses resulting from the later shipping date, developers of Dbase add-ons are faring considerably poorer. Sources close to Wallsoft Systems of New York say the developer of the Dbase code generator and template language UI Programmer is for sale. The firm has been hurt by the announcement that Dbase IV will include a UI-like code generator and template language. Although Wallsoft plans to release a second-generation update superior to that announced by Ashton-Tate, it still awaits Dbase IV.

“We’ve got 500 of the Fortune 500 companies saying, ‘Your product looks good, but we want to wait to see Dbase IV.’” said Martin Rinehart, Wallsoft’s president. Rinehart, who earlier this year led an independent developers’ effort to create a Dbase standard, would not comment on reports of his company’s troubles. He was among Dbase add-on developers who joined Ashton-Tate at the Dbase IV announcement and are now being hit hard by Dbase IV’s delay.

Bytel Corp. of Berkeley, California, also acknowledged orders are “down considerably” for its Dbase code generator, Genifer.

None of the add-on vendors greeted last week’s announcement of a further Dbase IV delay, though some claim business is still good. “It’s been for us the best spring ever,” said John Henderson, president of Concentric Data Systems Inc., in Westboro, Massachusetts, which makes the R&R Relational Report Writer for Dbase.

But even Henderson warned that “we can’t deal with uncertainty. The nature of these announcements is really personally disturbing. If sales weren’t doing so well, I might be singing a different tune.”

Microsoft Ships Bora of OS/2
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.

Note: NASA space shuttle controversy: see appendix.
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*
  - *Finding 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)*
    \[ D = a_1 f_1 + a_2 f_2 + a_3 f_3 + \cdots \]
  - **Multiplicative** *(ex. MIL-HDBK-217, COCOMO, RADC)*
    \[ D = C F_1(f_1) F_2(f_2) F_3(f_3) + \cdots \]
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 a 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 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 and 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](#)

![Graph showing equivalent defect density over time with different probability values](#)

\[(p_1, p_2) = (0.10, 0.10), (0.15, 0.05), (0.05, 0.15)\]
Software Evolution

Successive versions of a program

- Bug fixes
- Added functionality
- Added support for new requirements

Results in

- Added code
- Modified code
Regression testing: after revisions

Regression:
"when you fix one bug, you introduce several newer bugs."
Reuse factor: A simple analysis

- $u$: fraction of software reused
- $d_r, d_n$: defect density of reused software, defect density of new software, $d_r < d_n$
- Total defects = $[u \cdot d_r + (1-u) \cdot d_n]S$
  
  Where $S$ is software size

- If there was no reuse, defects would be $d_nS$

- Normalizing,
  - Reuse factor $F_r(u, d_r/d_n) = [u \cdot d_r / d_n +(1-u)]$
  - $F_r$ is 1 if there is no reuse, $<1$ if reuse.
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 (we will discuss this soon) yet available is not yet significant.
Static Model: Example

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

• 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 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.
Appendix
NASA Space Shuttle Defect Density?

  - NASA Space Shuttle Avionics have a defect density of 0.1 failures/KLOC (Edward Joyce, "Is Error-free Software Possible?" Datamation, Feb.18, 1989). Leading-edge software companies have a defect density of 0.2 failures/KLOC.

  - A few projects - for example, the space-shuttle software - have achieved a level of 0 defects in 500,000 lines of code using a system of format development methods, peer reviews, and statistical testing."

- Nancy G. Leveson, Software and the Challenge of Flight Control, Chap 7 of Space Shuttle Legacy: How We Did It/What We Learned, 2013.
  - A mythology has arisen about the Shuttle software with claims being made about it being “perfect software” and “bug-free” or having “zero-defects”

- They Write the Right Stuff, Charles Fishman, Fast Company, 12.31.96
  - It is perfect, as perfect as human beings have achieved. Consider these stats: the last three versions of the program — each 420,000 lines long—had just one error each. The last 11 versions of this software had a total of 17 errors. Ten years ago the shuttle group was considered world-class. Since then, it has cut its own error rate by 90%.

- NASA Space Shuttle project lasted from April 12, 1981 to July 21, 2011