Software Reliability Growth Approach

Lou Gullo
October 28, 2010
Overview

1. Value Added Benefits
2. Software Reliability Definitions
3. How SW Reliability Affects System Reliability
4. Software Reliability Prediction & Measurement Process
5. Tiered Software Development/Test Approach
6. IEEE 1633 (Steps 1, 2 and 3)
7. Sample CASRE Data
8. IOS and Ao (Steps 4 and 5)
9. Sample Results to Demonstrate Growth
10. Software Reliability Innovation – Path Forward
11. SW Reliability Reference Books
Value Added Benefits

- Reduce cost of failures later in the software development process
- Track failure trends of probabilistic conditions (e.g. race conditions) and systemic process-related issues
- Drive software design corrective actions to improve reliability results in a lower customer Total Cost of Ownership (TCO)
Software Reliability Definitions

- The American Institute of Aeronautics and Astronautics (AIAA) - “the application of statistical techniques to data collected during system development and operation to specify, predict, estimate, and assess the reliability of software-based systems.”

- IEEE 1633:
  - (A) The probability that software will not cause the failure of a system for a specified time under specified conditions.
  - (B) The ability of a program to perform a required function under stated conditions for a stated period of time.

- IEC 62628:
  - Software Dependability - ability of the software to perform as and when required when integrated in system operation

NOTE: Software Dependability includes Software Reliability as well as other measures of software performance and capability
How SW Reliability Affects System Reliability
Top-Level Event (System View)

- Fault Trees generated for TLE’s in System Safety Analyses

**COLOR CODE KEY**
- EXISTING WCE HAZARD
- UNKNOW NW CE HAZARD
- EXTERNAL TO CE HAZARD

---

**HW / SW Failure Effects**

**HW / SW Failure Modes**

**SW FMECA is Source Data at Bottom of SW Fault Trees**

---

**Element Hazards**

---

**Top Level Event (1 of 156)**

---
Integration of the Software Reliability into System Development Process

Software Development Process

Information Flow

Requirements Analyses → Detailed Design → Coding → Unit Test → Software Integ. Test → System Integ. Test → System Qual. Test

Information Feedback for Correcting Defects

Data Collection

Process Characteristics CMM Level & KSLOC Estimates
- Initial IOS Estimates

Capability Maturity Model
- Step 1

Software Development Defects Data Collection
- Updated IOS Estimates

Rayleigh Model/SWEEP Tool
- Step 2

CASRE Tool Set Models
- Step 3

Test Execution Time & Time Until Failure Data Collection
- IOS Actual Measurements

Reliability Assessment

Software Reliability Estimation/Performance Evaluation
- Reliability Models, Availability Assessment, Requirements Validation
Tiered System/Software Development/Test Approach

Tier 1

Multiple Sub-Systems / Thread Level (Development/Testing)

Tier 2

Sub-System Level (Development/Testing)

Tier 3

SCI / SRS Level (Development/Testing)

SAT (System Acceptance Test)

Multiple levels of SW Integration and Testing performed
IEEE 1633 – IEEE Recommended Practice on Software Reliability (SR)

- Developed by the IEEE Reliability Society in 2008
- Purpose of IEEE 1633
  - Promotes a systems approach to SR predictions
  - Although there are some distinctive characteristics of aerospace software, the principles of reliability are generic, and the results can be beneficial to practitioners in any industry.
How IEEE 1633 Aligns with SW Development Process

3 step process leveraging IEEE 1633:

- **Step 1 – Keene Model** for early software predictions
  - Weighs SEI CMMI Process Capability (e.g. CMMI Level 5 achieved by IDS) to Software Size (e.g. 10KSLOCs)

- **Step 2 – SWEEP Tool** for tracking growth of Software Trouble Reports (STRs) and Design Change Orders

- **Step 3 – CASRE Tool** for tracking failures in test
The Capability Maturity Model provides a preliminary prediction based on:

- Estimated size of the code in KSLOC
- Software Engineering Institute’s (SEI) Capability Maturity Model (CMM) rating of the software developer
- The assertion is that the software process capability is a predictor of the latent faults shipped with the code.

The higher the SEI Level the more efficient and Organization is in detecting defects early in development

The better the process, the better the process capability ratings and the better the delivered code, developed under that process, will perform….defects will be lower.
Keene Process-Based (a priori) SW Reliability Model (CMM Model) Inputs

- This model provides MTBF and Ao predictions for each Ensemble. These were used to confirm that the Ao requirements were reachable.

- These predictions are somewhat approximate, and so further refinement is needed in the later stages of the process.

<table>
<thead>
<tr>
<th>PROCESS INPUT PARAMETERS</th>
<th>Inputs</th>
<th>Range</th>
<th>Input Instructions:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Required</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KSLOCs</td>
<td>441.7</td>
<td>&gt;0</td>
<td>Number of 1,000 lines of source code (KSLOCs).</td>
</tr>
<tr>
<td>SEI Level - Develp</td>
<td>3</td>
<td>1-5</td>
<td>SEI Level factor (1-5).</td>
</tr>
<tr>
<td>SEI Level - Maint.</td>
<td>3</td>
<td>1-5</td>
<td>SEI Level factor (1-5).</td>
</tr>
<tr>
<td>Months to maturity</td>
<td>20</td>
<td>&lt;=48</td>
<td>Number of months to maturity or failure rate plateau.</td>
</tr>
<tr>
<td>Use hrs/week</td>
<td>168</td>
<td>&lt;=168</td>
<td>Number of operational hours per week.</td>
</tr>
<tr>
<td>% Fault Activation</td>
<td>100</td>
<td>&lt;=100</td>
<td>Average % population exhibiting fault activation.</td>
</tr>
<tr>
<td>Fault Latency</td>
<td>2</td>
<td>&gt;=1</td>
<td>Ave. # of fault reoccurrences/failing-site until corrected.</td>
</tr>
<tr>
<td>% Sev 1&amp;2 Fail</td>
<td>10</td>
<td>&lt;=100</td>
<td>Average % severity 1 and 2 or % countable failures.</td>
</tr>
<tr>
<td>MTTR</td>
<td>10</td>
<td>&gt;0</td>
<td>Average time to restore system (minutes)</td>
</tr>
</tbody>
</table>

Useful to Flowdown or Decompose Requirements to Lower Tiers
The SWEEP tool enables you to:

- Predict and track the rate at which defects will be found
- Predict the latent defect content of software products.
- Analyze estimated errors injected in each phase of the software development cycle
- Determine the detection effectiveness and leakage of errors to subsequent phases.
- Measure percentage of critical failures that feedback into the Keene model

SWEEP Data Collection

- Data is typically collected using Software Trouble Reports (STR)
- Data can be organized by development phase or time increments.
SWEEP Model Theory

- The SWEEP Tool uses the Rayleigh Model based on the Rayleigh Distribution
- The Rayleigh Distribution is a special case of the Weibull Distribution
- Model Assumptions
  - The defect rate observed during the development process is positively correlated with the defect rate in the field (The more area under the curve, the higher the field defect rate).
  - Given the same error injection rate, if more defects are discovered and removed earlier, fewer will remain in later stages.
- Reference Reading Metrics and Models in Software Quality Engineering, Addison Wesley Publishing (Author: Stephen Kan)
SWEEP Tool Output (Sample)

Estimated Total Errors: 1121.94
Projected Latent Errors: 0.7918
Selected Peak Location: 2.1
Error Discovery Efficiency: 99.93%

All Data is Notional
CASRE (Computer Aided Software Reliability Estimation)

Step 3

- CASRE (Computer Aided Software Reliability Estimation) is a software reliability measurement tool that runs in the Microsoft Windows environment...developed by Allen Nikora at JPL.

- The modeling and analysis capabilities of CASRE are provided by the public-domain software reliability package SMERFS (Statistical Modeling and Estimation of Reliability Functions for Software).

- In implementing CASRE, the original SMERFS user interface has been discarded, and the SMERFS modeling libraries have been linked into the user interface developed for CASRE.

- CASRE is typically applied starting after unit test and continuing through system test, acceptance test, and operations.

- You should only apply CASRE to modules for which you expect to see at least 40 or 50 failures. If you expect to see fewer failures, you may reduce the accuracy of your estimates.

- Experience shows that at the start of software test, modules having more than about 2000 source lines of executable code will tend to have enough faults to produce at least 40 to 50 failures.
CASRE Data Input

Two types of data files CASRE can accept

1. Times between successive failures.
   - Error Number (integer)
   - Time since last failure (floating point)
   - Error Severity (integer)

2. Failure counts per test interval and test interval length.
   - Interval Number
   - Number of Errors
   - Interval Length
   - Error Severity

Information to enhance the accuracy of model predictions:
- Date and time at which each failure was found, and the test interval during which the software was run that produced that failure.
- Date and time at which the testing method changed. The reason for this is that the perceived reliability of the system depends on how it is executed.
- Date and time at which the test environment changed. The reason for collecting this information is to more accurately characterize the length of a test interval.
- Date and time at which the software being tested changes significantly.
- Severity of each failure.
Sample CASRE Data Models and Output

Module A Actuals Vs CASRE Model Fit

All Data is Notional

Times Between Failures Models
- Geometric
- Jelinski-Moranda
- Littlewood-Verrall Linear
- Littlewood-Verrall Quadratic*
- Musa Basic
- Musa-Okumoto *

Failure Counts Models
- Generalized Poisson
- Schneidewind
- Shick-Wolverton
- Yamada S-shaped

Combination Models
- Dynamically-Weighted Linear Combination (DLC/S/4)
- Equally-Weighted Linear Combination (ELC)
- Median-Weighted Linear Combination (MLC)
- Unequally-Weighted Linear Combination (ULC)
Sample Results to Demonstrate Growth

Peaks demonstrate the Test Analyze and Fix (TAAF) Methodology

Demonstrates Traditional Growth Curve
IOS and Ao Calculations
Added Steps 4 and 5

Keene Model
Early Predictions

KSLOC
CMMI Level

SWEEP Tool
Change Analysis

STRs
Change Orders

CASRE Tool
Reliability
Calculation

Failure Count
Time to Failure
MTBF

Interruption of
Service (IOS)
Calculation

Recovery Time
Fail Over Type

Operational
Availability (Ao)
Calculation

Start Time
Stop Time
Runtime
IOS (MTTR)
MTBF

Closed Loop System with Step 5 Feedback to Steps 1 -3
Software Reliability Innovation – Path Forward

Initiatives to Increase SW Reliability Growth and Accelerate Deliveries of Mature / Dependable SW to the Warfighter:

- Continue to develop capabilities to detect software stress points earlier in the software life cycle
- Continue decreasing SW fault density significantly during SW production, prior to testing
- Continue improvement of software reliability growth testing processes and tools
- Continue to develop new standards or sustain existing standards (e.g. IEEE 1633 and IEC 62628)
- Develop more rigorous software development processes
SW Reliability Reference Books and Standards

- Metrics and Models in Software Quality Engineering, Stephen Kan, Addison Wesley Publishing
- IEEE 1633: Recommended Practice on Software Reliability (SR)
- IEC 62628: Guidance on Software Aspects of Dependability
Biography


Louis J Gullo
Sr Principal Systems Engineer
Lou.Gullo@Raytheon.com
401-842-4139