Strategy for Applying Software Reliability Growth Models (SRGM) to DOD Systems

Andy Long
Office of the Secretary of Defense (OSD),
Director, Operational Test & Evaluation (DOT&E)/Science Advisor
Scientific Research Corporation
January 2013
Purpose

Develop methodology for effectively applying software reliability growth models (SRGM) to track and predict software reliability growth by applying categorizations of software usage in DoD systems.

"All Models are Wrong - Some are Useful."
George E. P. Box
Software issues are common in programs of all types, and can often result in systems being rated as not operationally effective or not operationally suitable.

- Between 1997 and 2012, 12% of systems were found not reliable in IOT&E as result of failure modes attributed to software.

**Tasking from OSD/Dr. Catherine Warner, DOT&E Science Advisor:**
- Improve reliability tracking and growth modeling for all programs that incorporate software with focus on software centric systems, by allowing Program Managers (PMs) to:
  - better determine whether the current system state is sufficient for a product release;
  - determine if the release date should be deferred if existing software issues require resolution and corrective actions;
  - Make more informed tradeoffs during earlier development phases to optimize reliability within program constraints.
Categorizations of Software Usage Systems: Hybrid

- Modeled using traditional reliability growth modeling techniques for HW centric systems, because functionality results from HW and SW working together.

- Difficult to separate SW related failure from the function it relates to and the HW it controls.

- Reliability modeling of these systems is well-described by log-Poisson Non-Homogeneous Poisson Process (NHPP) models in the relation to time (e.g., the AMSAA Maturity Projection Model) due to their simplicity, convenience, and tractability.

Hybrid Systems
Categorizations of Software Usage Systems: Software Centric and Space Systems

(Where 2:3 means 2 are needed of 3 available assets)

- Use similar probabilistic models as do hybrid systems, but differ in that SRGM use design faults as their source for failures.
  - Design faults are typically usage dependent and time independent.
- SRGM specify the general form of the dependence of the failure process on the principal factors that affect it: fault introduction, fault removal and the operational environment.
- Space real-time data networks are very comparable to software centric systems for reliability modeling.
  - Satellite ground control centers use the same hardware and software infrastructures as conventional information technology applications.

Software Centric and Space Systems
Approach

• **Leverage existing work to model software reliability**
  – IEEE/AIAA P1633™ 2008, Recommended Practice on Software Reliability
    – Promotes a systems approach to software reliability predictions
    – Provides a sequence of steps for assessing software reliability that is independent of specific development processes and SWRGMs.

• **Use readily available tools:** "defect density" models; "software reliability growth" models to estimate reliability during design and integration testing

• **Focus is on reliability growth modeling for software centric and space systems**
Integration of IEEE 1633 into 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

Formal Testing

Data Collection
- Process Characteristics CMM Level & KSLOC Estimates
  * Initial IOS Estimates

Reliability Assessment
- Capability Maturity Model
  * Step 1
- Software Reliability Estimation/Performance Evaluation
- Reliability Models, Availability Assessment, Requirements Validation

Reliability Growth Models
- Rayleigh Model/SWEEP Tool
  * Step 2
- Estimate Number of Defects
- Software Reliability Estimation/Performance Evaluation
- Reliability Models, Availability Assessment, Requirements Validation

Defect Detection Data
- Test Execution Time & Time Until Failure Data Collection
- IOS Actual Measurements
- CASRE Tool Set Models
  * Step 3
- Reliability Growth Models
- Statistically Correlates Data to Function
- Software Reliability Estimation/Performance Evaluation
- Reliability Models, Availability Assessment, Requirements Validation

Approach

- Thousand Source Lines of Code (KSLOC)
- Software Error Estimation Program (SWEEP)
- Computer Aided Software Reliability Estimation (CASRE)
Step 1: Capability Maturity Model (Keene Model)

Theory
- The better the process capability ratings, the better the delivered code will perform.

SEI process capability levels:
- Level 1: Initial (adhoc)
- Level 2: Repeatable (policies)
- Level 3: Defined (documented)
- Level 4: Managed (measured and capable)
- Level 5: Optimized

Data
- SEI level for the developing organization,
- Number of months for the software failure rate to stabilize,
- Number of failure replications expected prior to fault removal,
- Fault activation rate,
- Percentage of all failures that are critical failures, and the
- Projected run time per month.

Assumption
- SEI Development Level Correlate to Latent Faults

Enables
- Transforms the latent fault density into in an exponential reliability growth curve over time to estimate reliability

Useful to Flowdown or Decompose Requirements to Lower Tiers
Step 2: Rayleigh Models/Software Error Estimation Program (SWEEP) Capabilities

Theory
- Rayleigh Model: Weibull distribution with shape parameter, \( m = 2 \).

Assumptions
- The defect rate observed during the development process is positively correlated with the defect rate in the field.
- Given the same error injection rate, if more defects are discovered and removed earlier, fewer will remain in later stages.

Enables
- Predicting and tracking the rate at which defects will be found;
- Predicting the latent defect content of software products;
- Analyzing estimated errors injected in each phase of the software development cycle;
- Measuring percentage of critical failures

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

Useful To Predict the Software’s Latent Fault Density

Rayleigh/SWEEP Tool
Step 3: Computer Aided Software Reliability Estimation (CASRE)

- Software reliability measurement tool that runs in the Microsoft Windows environment...developed by Dr. Allen Nikora at JPL.
- The modeling and analysis capabilities are provided by the public-domain software reliability package, Statistical Modeling and Estimation of Reliability Functions for Software (SMERFS).
  - 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 fielding.
- CASRE should be applied to modules for which you expect to see at least 40 or 50 failures--Experience shows that at the start of software test, modules having more than 2K source lines of code (2KSLOC) will tend to have enough faults to produce at least 40 to 50 failures.
CASRE Reliability Growth Models

- The exponential model is regarded as the basic form of SWRGM
  - Since the early 1970s, more than a hundred models have been proposed
  - Few have been tested in practical environments; even fewer are in use.

- Models currently being used by type, form, and assumptions are:

<table>
<thead>
<tr>
<th>Model Types:</th>
<th>CASRE Model Forms:</th>
<th>Assumptions:</th>
</tr>
</thead>
<tbody>
<tr>
<td>I. Predict Time Between Failures</td>
<td>Execution Time Models</td>
<td>- There are N unknown software faults at the start of testing</td>
</tr>
<tr>
<td>Input Data (Typically from Problem Reports):</td>
<td>Geometric</td>
<td>- Failures occur randomly</td>
</tr>
<tr>
<td>• Error Number (integer)</td>
<td>Jelinski-Moranda</td>
<td>- All faults contribute equally to failure</td>
</tr>
<tr>
<td>• Time since last failure (floating point)</td>
<td>Littlewood-Verrall Linear</td>
<td>- Fix time is negligibly small</td>
</tr>
<tr>
<td>• Error Severity (integer)</td>
<td>Littlewood-Verrall Quadratic</td>
<td>- Fix is perfect for each fault</td>
</tr>
<tr>
<td>Parameters (for most models):</td>
<td>Musa Basic</td>
<td>- Testing intervals are independent of each other</td>
</tr>
<tr>
<td>$\mu(t)=\alpha F(t)$, wh: $\alpha$ is the expected no. of defects and</td>
<td>Musa-Okumoto</td>
<td>- Testing during intervals is reasonably homogeneous</td>
</tr>
<tr>
<td>$F(t)$ is a CDF; $F(0)=0, F(\infty)=1, \mu(\infty)=\alpha$</td>
<td>Non-homogeneous Poisson (NHPP)</td>
<td>- Number of defects detected is independent of each other</td>
</tr>
<tr>
<td>II. Predict Failures Count</td>
<td>Generalized Poisson (interval weight optional)</td>
<td>- Testing intervals are independent of each other</td>
</tr>
<tr>
<td>Input Data (Typically from Problem Reports):</td>
<td>Schneidewind</td>
<td>- Testing during intervals is reasonably homogeneous</td>
</tr>
<tr>
<td>• Interval Number</td>
<td>Schneidewind (combines 1st s-1 intervals)</td>
<td>- Number of defects detected is independent of each other</td>
</tr>
<tr>
<td>• Number of Errors</td>
<td>Shick-Wolverton</td>
<td>- Testing intervals are independent of each other</td>
</tr>
<tr>
<td>• Interval Length</td>
<td>Yamada S-shaped</td>
<td>- Testing during intervals is reasonably homogeneous</td>
</tr>
<tr>
<td>• Error Severity</td>
<td>Non-homogeneous Poisson (NHPP)</td>
<td>- Number of defects detected is independent of each other</td>
</tr>
<tr>
<td>Parameters (for most models):</td>
<td></td>
<td>- Testing intervals are independent of each other</td>
</tr>
<tr>
<td>$\mu(t)=\alpha F(t)$, wh: $\alpha$ is the expected no. of defects and</td>
<td></td>
<td>- Testing during intervals is reasonably homogeneous</td>
</tr>
<tr>
<td>$F(t)$ is a CDF; $F(0)=0, F(\infty)=1, \mu(\infty)=\alpha$</td>
<td></td>
<td>- Number of defects detected is independent of each other</td>
</tr>
</tbody>
</table>
## CASRE Model Selection Rules for Picking The “Best Fit Model”

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Estimate the model parameters using statistical techniques such as maximum likelihood or least-squares methods. The maximum likelihood technique solves for optimal parameter values. The least squares method solves for parameter values that best fit a curve to the data.</td>
</tr>
<tr>
<td>2.</td>
<td>Goodness-of-fit Test (i.e., Kolmogorov-Smirnov (K-S) test or Chi-Square) on the model Tests the null hypothesis that the appropriate model at an alpha confidence level of 5% fits the data. A “Yes” response indicates the model does fit the data.</td>
</tr>
<tr>
<td>3.</td>
<td>Prequential Likelihood Ratio (PLR) Given a prior belief that either model A or B is equally appropriate, the PLR defines the likelihood that model A will produce more accurate estimates than model B.</td>
</tr>
<tr>
<td>4.</td>
<td>Model Bias Quantifies the extent to which a model consistently makes predictions that are larger or smaller than those observed.</td>
</tr>
<tr>
<td>5.</td>
<td>Model Bias Trend Determine extent bias changes over time</td>
</tr>
</tbody>
</table>
Software Reliability Assessment for an Example System

• The modeled system is a navigation system. The following software modules of varying lines of code and critical failure events were analyzed using CASRE:

<table>
<thead>
<tr>
<th>Module</th>
<th>KSLOC</th>
<th># Events</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>203.9</td>
<td>117</td>
</tr>
<tr>
<td>B</td>
<td>43.3</td>
<td>13</td>
</tr>
<tr>
<td>C</td>
<td>74.8</td>
<td>18</td>
</tr>
<tr>
<td>D</td>
<td>145.9</td>
<td>29</td>
</tr>
</tbody>
</table>

• Data modeled here are for SW centric systems. Peterson et alii modeled five software modules using both Time Between Failures and Failures Count models. They conclude that “There was not enough data to support a high significance level of fit for the failure count models, so without further analysis, the inter-fail time models (Execution Time Models) are preferred by inspection.”

• So, this example will be limited to reliability growth modeling using only the time between failure models.

Example System
## Goodness of Fit and Model Ranking Results

<table>
<thead>
<tr>
<th>Module</th>
<th>Model Name</th>
<th>KS Distance</th>
<th>5.% Fit</th>
<th>Significance (%)</th>
<th>Rank</th>
<th>Estimated Time To Failure at Last Interval (Hrs.) (Failure Rate For Critical Priority 1,2, &amp; 3 Failures)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Quadratic LV</td>
<td>0.0930</td>
<td>Yes</td>
<td>25.8%</td>
<td>1</td>
<td>513</td>
</tr>
<tr>
<td></td>
<td>Linear LV</td>
<td>0.1019</td>
<td>Yes</td>
<td>16.4%</td>
<td>2</td>
<td>429</td>
</tr>
<tr>
<td>B</td>
<td>Quadratic LV</td>
<td>0.2438</td>
<td>Yes</td>
<td>36.4%</td>
<td>1</td>
<td>2053</td>
</tr>
<tr>
<td></td>
<td>Musa- Okumoto</td>
<td>0.2132</td>
<td>Yes</td>
<td>53.6%</td>
<td>2</td>
<td>2491</td>
</tr>
<tr>
<td></td>
<td>Geometric</td>
<td>0.2318</td>
<td>Yes</td>
<td>42.7%</td>
<td>3</td>
<td>3411</td>
</tr>
<tr>
<td></td>
<td>Jelinski-Moranda</td>
<td>0.2141</td>
<td>Yes</td>
<td>53.0%</td>
<td>4</td>
<td>2072</td>
</tr>
<tr>
<td></td>
<td>Musa Basic</td>
<td>0.1827</td>
<td>Yes</td>
<td>75.0%</td>
<td>5</td>
<td>2320</td>
</tr>
<tr>
<td>C</td>
<td>Quadratic LV</td>
<td>0.2179</td>
<td>Yes</td>
<td>31.4%</td>
<td>1</td>
<td>2053</td>
</tr>
<tr>
<td></td>
<td>Musa- Okumoto</td>
<td>0.2053</td>
<td>Yes</td>
<td>38.4%</td>
<td>2</td>
<td>2861</td>
</tr>
<tr>
<td></td>
<td>Geometric</td>
<td>0.2507</td>
<td>Yes</td>
<td>17.5%</td>
<td>3</td>
<td>2862</td>
</tr>
<tr>
<td></td>
<td>Jelinski-Moranda</td>
<td>0.2031</td>
<td>Yes</td>
<td>39.7%</td>
<td>4</td>
<td>2861</td>
</tr>
<tr>
<td></td>
<td>Musa Basic</td>
<td>0.1929</td>
<td>Yes</td>
<td>46.3%</td>
<td>5</td>
<td>9480</td>
</tr>
<tr>
<td>D</td>
<td>Quadratic LV</td>
<td>0.1392</td>
<td>Yes</td>
<td>59.5%</td>
<td>1</td>
<td>750</td>
</tr>
<tr>
<td></td>
<td>Geometric</td>
<td>0.2356</td>
<td>Yes</td>
<td>6.7%</td>
<td>2</td>
<td>638</td>
</tr>
<tr>
<td></td>
<td>Linear LV</td>
<td>0.0809</td>
<td>Yes</td>
<td>&gt; 82.5%</td>
<td>3</td>
<td>984</td>
</tr>
</tbody>
</table>

### Selection of Best CASRE Model
Actuals vs. CASRE Tool Predictions

Plot of Actual Vs Predicted
Module A

Plot of Actual Vs Predicted
Module B

CASRE Tool Predictions Vs. Actuals
Summary/Conclusions

• The concerted applications of the reliability models, Keene/CMM, Rayleigh/SWEEP, and CASRE make the best use of available data to predict the code reliability at the various stages of development.
• Simple models perform as well or better than complex models—software reliability SMES [refs 4, 5, 8] indicate the simple exponential model tend to outperform more complex models in terms of both stability and predictive ability.
• Execution time is the best measure of the amount of testing.
• Problem reports are a good surrogate for defects.
• The great advantage of CASRE is that it consolidates many well-established reliability models in one place and automates the determination of various ranking criteria for each model.

What Makes A Model Useful?
1. Stability During the Test Period and Remains Stable Until the End of the Test
2. Reasonable Prediction of the Number of Defects that Will Be Discovered In Field Use
Additional Slides
References

Actuals vs. CASRE Tool Predictions

Plot of Actual Vs Predicted Module C

Plot of Actual Vs Predicted Module D
Fraction of Non-Critical Failures Vs. Time

Fraction of Priority 3 vs Time for Module A

Fraction of Priority 3 vs Time for Module B

Ordered Event Number Over Time