Software Risk Assessment

Analyzing the Fallible Machine

The Open PSA Initiative

A Blending of Disciplines

- Systems Engineering
- Software Reliability
- Software System-Safety
- Probabilistic Risk Assessment
- Software Engineering
- Software Development Management
Essential Properties of Software

- Software is constructed from human thought, or reason.
- The material sciences of software are mathematics and cognitive sciences.
- Its physical existence is a set of instructions in an English-like code.
- Software does not wear out; but it does get over-maintained or over-changed.
- Each software construction is unique.
- Software is combinatorially explosive with respect to possible execution paths.
- When software fails, no deformations or breakage occur.
- Software fails with little, if any, advanced warning (we don’t hear the gears grinding).
- Software errors exemplify action at a distance.

Software System Failures

- Failures are at the intersections of system viewpoints

- Software system failures are disconnects between users, software writers, the operating environment, and the computing machine
Systems Engineering

• Hierarchical Approach
• Software is **NOT** a Standalone Device
• Software is a Component in a System
  – Software controlled devices
  – Software supported decisions
• Concern is Primarily with Systems Effects of Software Errors
  – Therac 25
  – Precision Errors

Software Reliability

• Software is a Product which does **NOT** perform Perfectly
• Both Parallels and Differences Between Hardware and Software Reliability
• Refers to a Collection of Concepts
  – Number of software errors
  – Software RAM
  – Software failure rate \( (F, R, z, H) \)
  – MTTF or MTBF
• Techniques which Lower Frequency and Consequences of Failures
• Measurement and Modeling of Probability of Failure
First, Some Definitions:

- **Fault**
  - Something wrong in a program
- **Error**
  - A manifestation of a FAULT during software operation
- **Failure**
  - An ERROR which interferes with system operation
- **Bug**
  - Any of the above
- **Feature**
  - A BUG to which users have become accustomed

Software Error Model Definitions

- **$E_T$**
  - Total number of errors in program
- **$E_C(\tau)$**
  - Total number of errors corrected after time $\tau$ of debugging
- **$E_R(\tau) = E_T - E_C(\tau)$**
  - Total number of errors remaining after time $\tau$ of debugging
- **$I_T$**
  - Total number of program instructions, or lines of code
Error Removal Rate for Two Large Programs

Reliability Definitions

- $R(t) = e^{-\lambda t}$
  - Constant Hazard Reliability Function

- $MTTF = \int_0^\infty t f(t) dt$
  - MTTF: Mean Time to Failure

- $MTTF = \int_0^\infty R(t) dt$
  - Alternatively, and more simply

- $MTTF = \int_0^\infty e^{-\lambda t} dt = e^{-\lambda t} \bigg|_0^\infty = \frac{1}{\lambda}$
  - For a constant hazard, we obtain the above
The Bathtub Curve of Failure Rates

- By studying this curve for electrical or mechanical failure rates we see that:
  - Burn-in minimizes early failures
  - Scheduled maintenance minimizes late failures
  - Operational failures happen at a constant rate

- Software renames things a bit:
  - Burn-in is code release
  - Wear-out is code maintenance

The Constant Hazard Model

- \( z(t) = \lambda \)
  - Hazard, or failure rate, function

- \( f(t) = \lambda e^{-\lambda t} \)
  - Failure density function

- \( R(t) = e^{-\lambda t} \)
  - Reliability function

- \( F(t) = 1 - R(t) \)
  - Unreliability function
The Constant Hazard Model

Normalizing the Software Error Model

- Cumulative Errors per 1000 Instructions for Program A
- We Normalize the Error Rate to the Number of Instructions in a program:
  \[ \frac{E_R(t)}{I_T} = \frac{E_T}{I_T} \cdot \frac{E_C(t)}{I_T} \]
- Notice that:
  \[ \varepsilon_1(\tau) = \frac{E_R(t)}{I_T} \text{ and } \varepsilon_2(\tau) = \frac{E_C(t)}{I_T} \]
Software Failure Rate Model

- Failure Rate is Proportional to Number of Remaining Errors
  - Intuitive
  - Model is Simple
  - Musa has Data to Support

\[ z(t) = K(e_1(t)) = K \left[ \frac{E_T}{I_T} - e_2(t) \right] \]

where normalized parameters \( \alpha \) and \( \beta \):

\[ \beta = \frac{E_T}{I_T}K \text{ and } \alpha = \frac{e_2(0)I_T}{E_T} \]

\[ z(t) = \beta(1 - \alpha t) \]

Example of Failure Rate Estimation

After 1 month of integration testing, a 25,000 line long program was found to have 12 errors. Since about 5 tests a day were run at 1 hour/test, the average MTTF was 8 hrs. After 2 months, 18 total errors were found with a MTTF of 12 hrs.

- Estimation of Model Parameters

\[ z(\tau) = \frac{1}{MTTF} = \frac{K}{I_T} [E_T - E_C(\tau)] \]

\[ .125 = \frac{K}{2.5 \times 10^4} [E_T - 12] \]

\[ .083 = \frac{K}{2.5 \times 10^4} [E_T - 18] \]

\[ \frac{125}{.083} = 1.5 \approx \frac{E_T - 12}{E_T - 18} \Rightarrow E_T = 30 \]

\[ .125 = \frac{K}{2.5 \times 10^4} [30 - 12] \Rightarrow K = 173.6 \]

\[ z(\tau) = 6.94 \times 10^{-3} [30 - E_C(\tau)] \]
Estimation will change if a different error removal rate is assumed or if it changes during testing.

Reliability and Failure Rates

- **Specification**
  - Failure rates should be based on measurements during field operation of similar successful and unsuccessful software.

- **Prediction**
  - Early prediction requires a similar project for study and for failure rate model parameters; in short, a database.

- **Measurement**
  - Record running hours of program during test and simulation; count software failures; calculate failure rate during development at several points.

- **Confirmation**
  - Confirm, or update, model during field test and early operation.
Software System-Safety

- Software Hazard Identification
- Hazard Assessment
  - System Impact
  - Consequence Criticality
  - Mitigation Possibilities
- Software Failure Mode Identification
- Design Software “devices” to Eliminate or Control Hazards

Tools and Techniques for Software System-Safety Analysis

- Integrated into the Software Life-cycle
- Analysis is Iterative
- Code Walkthroughs
- Safety Critical Functions and Variable Analysis
- HAZOPs
- Fault Tree Analysis
- Software Root Cause Analysis
Code Walkthroughs

- A Structured Way to Represent code
- Frame or Timing Diagrams
- Flow Diagrams
- Interaction Diagrams
- Mathematical Proofs

Principal Segments for Propellant Valve Control
Mathematical Proofs

- Floyd/Hoare (1975)
  - I/O Assertion Method
  - Loop Invariant Statements
  - Symbolic Execution
  - Hand/Automated Proofs
  - Assertions and Invariants Extremely Difficult to Formulate
- D. Parnas (1991)
  - Rewrite Requirements into A-7 Event and Condition Tables
  - Handproof using Functional Abstraction called Program Function Tables
  - Only Proves that Program Fulfills Requirements
  - At Ontario-Hydro, 7,000 LOC took 30 man/years
- Proof Procedure is as much in question as the Code
Safety Critical Variable
Analysis

**Variable Name:** Oxidizer_Coeff0

**Definition:** Oxidizer coefficient; loaded with adaptation data

<table>
<thead>
<tr>
<th>Units:</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>psi/sec/lb</td>
<td>na</td>
<td>na</td>
</tr>
</tbody>
</table>

**Data Type:** Constant

<table>
<thead>
<tr>
<th>Expert:</th>
<th>Global</th>
</tr>
</thead>
</table>

| Value: | 2.92104 |

**Function Name:** Calculate_Mixture_Ratio (287)

<table>
<thead>
<tr>
<th>Variable Affected</th>
<th>Line</th>
<th>How Affected</th>
<th>Function Performed</th>
<th>Candidate for Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oxidizer Flow Rate</td>
<td>19</td>
<td>Increase</td>
<td>[Oxidizer_Coeff0]</td>
<td>As</td>
</tr>
<tr>
<td>Lox_Mass_Flow</td>
<td>18</td>
<td>Increase</td>
<td>[Oxidizer_Flow_Rate]</td>
<td>As</td>
</tr>
<tr>
<td>Oxidizer_Flow Rate</td>
<td>19</td>
<td>Increase</td>
<td>[Oxidizer_Flow_Rate]</td>
<td>As</td>
</tr>
<tr>
<td>Lox_Mass_Ratio</td>
<td>18</td>
<td>Increase</td>
<td>[Oxidizer_Flow_Rate]</td>
<td>As</td>
</tr>
<tr>
<td>Mixture_Ratio</td>
<td>15</td>
<td>Increase</td>
<td>[Oxidizer_Flow_Rate]</td>
<td>As</td>
</tr>
</tbody>
</table>

*Note: Mixture Ratio is the only variable which needs trace continued for assessing impacts of Oxidizer_Coeff0*

If Lox_Mass_Flow is 0, then Mixture_Ratio is set to 15.9993513719
15
Software Hazard Analysis Report

Identification No. ___________

Title

Software Affected

Mission Phase/Major Mode

Hazard Description/Discussion

Recommended Hazard Controls

Action/Status-Disposition

Hazard Tracking Documents

As Of Date ______ Status ______ Originator ______ Date ______

Closure Sign-off __________ Date ______

Software Fault Tree
Software Error Root Cause Analysis

• For Each Project the Following Data Should be Kept
  – Life-cycle used
  – Start and end date for each phase
  – Effort spent in each phase
  – Development Environment
  – Programming experience
  – SEI Index
  – Number of lines of code
  – Source language
  – Target hardware

• For Each Failure, a Summary and Data Record Should be Created
Some Observations

- **Software Reliability**
  - Quantifies Failure Rate
  - Does NOT look at code
  - Relies on Measurement
  - Applies to the MACRO level
  - Results are "Hard"

- **Software System-Safety**
  - Does NOT Quantify
  - Analyzes the code
  - Relies on Judgment
  - Applies Hierarchically
  - Results are "Soft"

The Trouble with Testing

- **Hecht's Law (1992)**
  - Infrequently Executed Code has a High Failure Rate
    - Redundancy Management
    - Exception Handling
    - Initialization
    - Calibration
  - Off-nominal Conditions are a Prominent Cause of Failure in Well-tested Systems
  - Test Profile must be Rich in Off-nominal Conditions
  - Software Telemetry Needed

- **The Butler/Finelli Observations (1991)**
  - Reliability Quantification to Low Levels is Statistically Infeasible
  - Separately Programmed Versions do **NOT** Fail Independently
A Testing Example

Confidence in Software Failure Rate

\[ z(t) = 1e-05/hr. \]

Level of Confidence

Propagation of Uncertainties

Software Risk Assessment Methodology

- Elements of PSA
  - Uncertainties
  - Bayes’ Theorem
  - Scenarios
    - Master Logic Diagrams (MLD)
    - Event Sequence Diagrams (ESD)
    - Initiating Event Fault Trees
  - Propagation of Uncertainties
- Synthesis into a Programmatic Approach
  - Hierarchical Analysis
  - Quantitative Methods from Software Reliability
  - Qualitative Methods from Software System-Safety
Uncertainties about Failure Rates

Criteria for Calculation of an Application Similarity Index

- Design
- Operational Profile
- Type of Project
- Size of Code
- Complexity Metrics
- Development Environment
- Language Used
- Total Test Time

Similarity index is calculated by solving for the principal eigenvector of an application comparison matrix.
Complexity Metrics

- Measures
  - Size of Code
  - Number of Branches
  - Program Structure
  - Flow of Control

- Used on the Subprogram Level

- Commonly Used Metrics
  - SLOC
  - McCabe’s Cyclomatic Complexity
  - NPATH
  - Halstead Software Science

- Highly Correlated with Each other

- Moderate Correlation between Complexity Thresholds and Failure Rates
Example of Bayesian Update of Software Failure Rate

<table>
<thead>
<tr>
<th>Frequency</th>
<th>Prior Probability of Failure Rate</th>
<th>Prior Cumulative</th>
<th>Update Probability of Failure Rate</th>
<th>Update Cumulative</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5000</td>
<td>5.548-02</td>
<td>5.548-02</td>
<td>1.250-02</td>
<td>6.800-02</td>
</tr>
<tr>
<td>0.6000</td>
<td>6.750-02</td>
<td>1.200-02</td>
<td>3.000-02</td>
<td>6.000-02</td>
</tr>
<tr>
<td>0.7000</td>
<td>1.500-02</td>
<td>4.500-02</td>
<td>6.100-02</td>
<td>4.900-02</td>
</tr>
<tr>
<td>0.8000</td>
<td>7.550-02</td>
<td>1.050-02</td>
<td>1.000-02</td>
<td>7.900-02</td>
</tr>
<tr>
<td>0.9000</td>
<td>3.250-02</td>
<td>0.500-02</td>
<td>3.250-02</td>
<td>3.500-02</td>
</tr>
<tr>
<td>1.0000</td>
<td>1.000-02</td>
<td>0.250-02</td>
<td>1.000-02</td>
<td>1.000-02</td>
</tr>
<tr>
<td>1.1000</td>
<td>0.000-02</td>
<td>0.000-02</td>
<td>0.000-02</td>
<td>0.000-02</td>
</tr>
<tr>
<td>1.2000</td>
<td>0.000-02</td>
<td>0.000-02</td>
<td>0.000-02</td>
<td>0.000-02</td>
</tr>
</tbody>
</table>

Draft MLD for OP
Draft Engine Level ESD for OP Initiated Scenario
Main Stage Closed Loop Control

Draft Example Fault Tree for Initiating Event
Propagation of Uncertainties

• Event Sequence Diagrams are Used to Construct Event Trees
• Pivotal Events become Top Events
• Software Pivotal Events are Intermixed with Phenomenal and Failure Pivotal Events
• Each Path through an Event Tree Depicts a Possible Scenario
• Probability Distributions for each Scenario are created by Monte Carlo Simulation

Steps in an SRA

• Review Software
  — Purpose
  — Functions
  — Construction
  — Operation
  — Software Engineering
• Review Controls
  — Reliability Models
  — Failure Data
  — Problem Reports
  — Testing Program
• Develop Scenarios
  — Add to Existing or Create New Scenarios
  — Employ
    • MLDs
    • ESDs
    • SFTs
    • Event Trees
Steps in an SRA
(Continued)

• Collect Data
  – Four types of data
    • Operational failure data and time
    • Test data and time
    • Failure Rate from similar applications
    • Judgment from expertise and experience
  – Develop prior distributions
  – Modify with test data
  – Develop likelihood functions
  – Create application specific data
  – Probability (Critical Failure | Failure)

• Allocate failure rate over the software functions
  – By fraction of time during nominal processing
  – By Proportion of Similar Failures
  – By Complexity of Functions in Scenario
  – By Expert Elicitation

• Quantify Scenarios