Software Risk Assessment

Analyzing the Fallible Machine

The Open PSA Initiative

A Blending of Disciplines

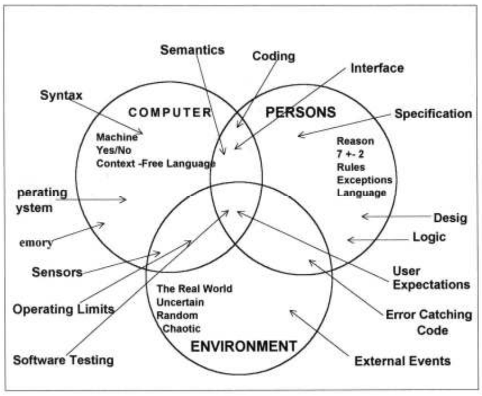
- Systems Engineering
- Software Reliability
- Software System-Safety
- Probabilistic Risk Assessment
- Software Engineering
- Software Development Management

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

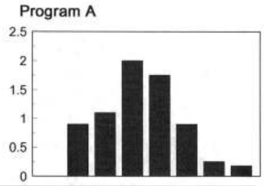
First, Some Definitions:

- 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

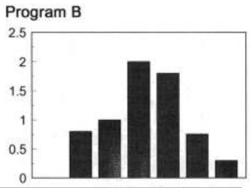
Software Error Model Definitions

- E_T
  - Total number of errors in program
- $E_C(\tau)$
  - Total number of errors corrected after time τ of debugging
- $E_R(\tau) = E_T - E_C(\tau)$
  - Total number of errors remaining after time τ of debugging
- I_T
  - Total number of program instructions, or lines of code

Error Removal Rate for Two Large Programs



Months of Testing/Debugging	0	1	2	3	4	5	6	7
Error Rate per 1000 Instructions	0.00	0.90	1.10	2.00	1.75	0.90	0.25	0.18

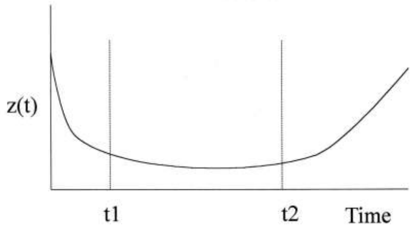


Months of Testing/Debugging	0	1	2	3	4	5	6
Error Rate per 1000 Instructions	0.00	0.80	1.00	2.00	1.80	0.75	0.30

Reliability Definitions

- $R(t) = e^{-\lambda t}$
  - Constant Hazard Reliability Function
- $\text{MTTF} = \int_0^\infty t f(t)dt$
  - MTTF: Mean Time to Failure
- $\text{MTTF} = \int_0^\infty R(t)dt$
  - Alternatively, and more simply
- $\text{MTTF} = \int_0^\infty e^{-\lambda t}dt = \frac{e^{-\lambda t}}{-\lambda} \Big|_1^\infty = \frac{1}{\lambda}$
  - For a constant hazard, we obtain the above

The Bathtub Curve of Failure Rates

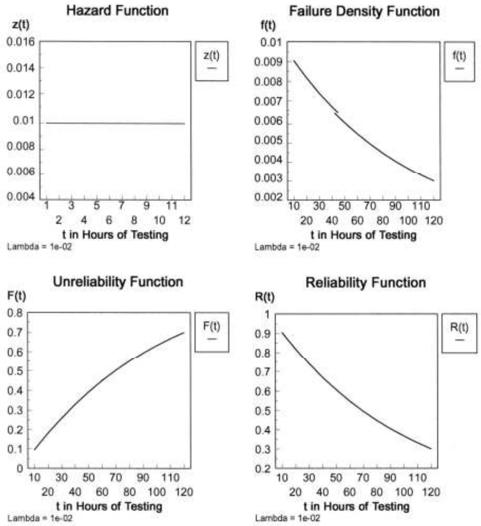


- By studying this curve for electrical or mechanical failure rates we see that:
  - Burn-in minimizes early failures
  - Scheduled maintenance minimizes late failures
  - Operational failures happen at a constant rate
- Software renames things a bit:
  - Burn-in is code release
  - Wear-out is code maintenance

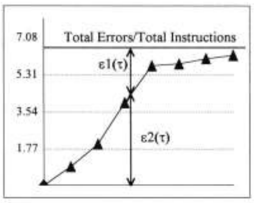
The Constant Hazard Model

- $z(t) = \lambda$
  - 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:

$$\frac{E_R(\tau)}{I_T} = \frac{E_T}{I_T} - \frac{E_C(\tau)}{I_T}$$

- Notice that:

$$\varepsilon_1(\tau) = \frac{E_R(\tau)}{I_T} \text{ and } \varepsilon_2(\tau) = \frac{E_C(\tau)}{I_T}$$

Software Failure Rate Model

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

$$z(t) = K(\varepsilon_1(\tau)) = K\left[\frac{E_T}{I_T} - \varepsilon_2(\tau)\right]$$

we define normalized parameters α and β:

$$\beta = \frac{E_T}{I_T}K \text{ and } \alpha = \frac{\varepsilon_2(\tau)I_T}{E_T}$$

so:

$$z(t) = \beta(1 - \alpha t)$$

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

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

$$.125 = \frac{K}{2.5\times10^4}[E_T - 12]$$

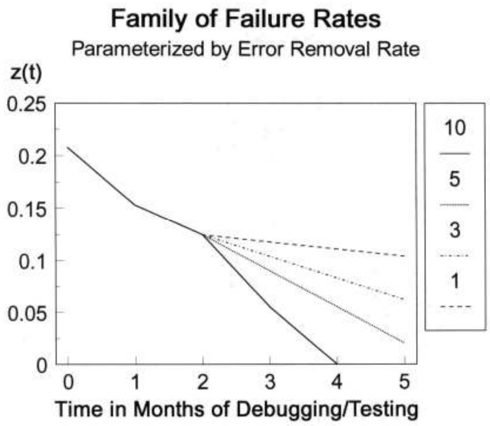
$$.083 = \frac{K}{2.5\times10^4}[E_T - 18]$$

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

$$.125 = \frac{K}{2.5\times10^4}[30 - 12] \Rightarrow K = 173.6$$

$$z(\tau) = 6.94 \times 10^{-3}[30 - E_C(\tau)]$$

Estimation Example



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

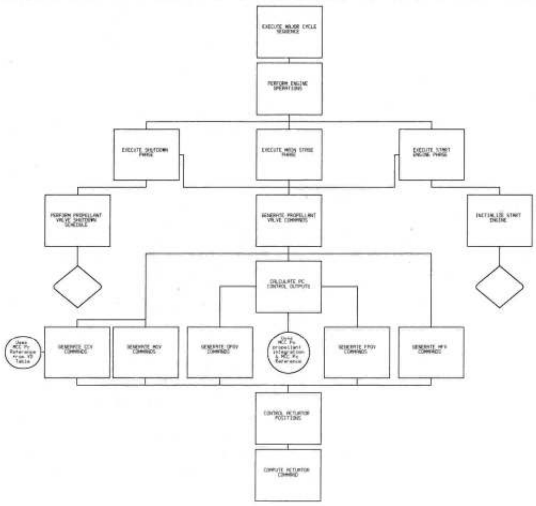
Tools and Techniques for Software System-Safety Analysis

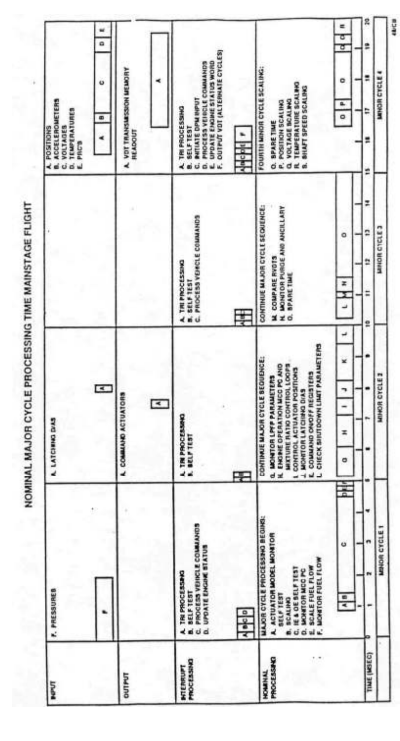
- Integrated into the Software Life-cycle
- Analysis is Iterative
- Code Walkthroughs
- Safety Critical Functions and Variable Analysis
- HAZOPs
- Fault Tree Analysis
- Software Root Cause Analysis

Code Walkthroughs

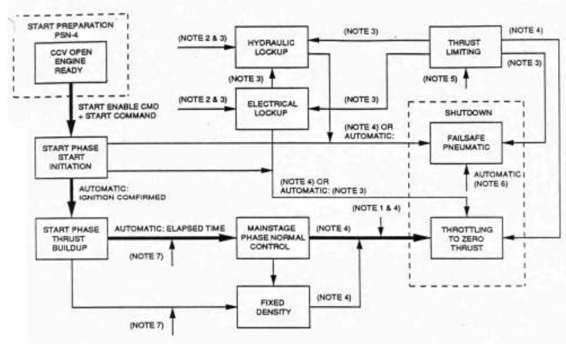
- A Structured Way to Represent code
- Frame or Timing Diagrams
- Flow Diagrams
- Interaction Diagrams
- Mathematical Proofs

Principal Segments for Propellant Valve Control





CP406R0002F (V2.5), PART I, VOLUME 2

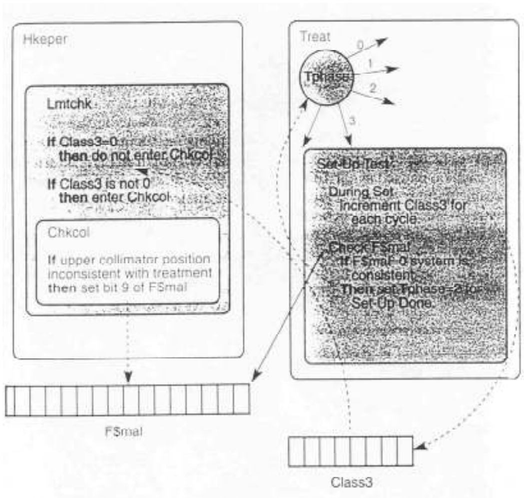


NOTES:

1. FROM ANY MODE OF START OR NON-LOCKUP MODE OF MAINSTAGE.
2. FROM ANY NON-LOCKUP MODE OF MAINSTAGE.
3. FAILURE RESPONSES AS DEFINED UNDER 3.2.4.
4. SHUTDOWN ENABLE COMMAND FOLLOWED BY SHUTDOWN COMMAND.
5. FROM MAINSTAGE NORMAL CONTROL OR FIXED DENSITY MODE WHEN OPOV POSITION COMMAND IS LIMITED FOR 3 CONSECUTIVE MAJOR CYCLES.
6. SHUTDOWN PNEUMATICALLY IN FRT-1 COMMAND, 3.2.3.2.4.1.1.1.
7. FROM THRUST LIMITING MODE WHEN OPOV POSITION COMMAND IS NOT LIMITED FOR 3 CONSECUTIVE MAJOR CYCLES.
8. SOLID BLOCKS REPRESENT ENGINE OPERATING MODES.
9. HEAVY LINES REPRESENT NORMAL OPERATING SEQUENCE.
10. LIGHTER LINES REPRESENT OPTIONAL CAPABILITIES OR DEVIATIONS DUE TO MALFUNCTION DETECTION. THE PATHS SHOWN ARE NOT ALL-INCLUSIVE. THE FULL SET OF POSSIBILITIES IS DEFINED BY THE COMMAND ACCEPTANCE LOGIC AND THE FAILURE RESPONSE SPECIFIED IN THE TEXT.

FIGURE 4

START AND MAINSTAGE PHASE



Mathematical Proofs

- Floyd/Hoare (1975)
  - I/O Assertion Method
  - Loop Invarient 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: **Oxidzer_Coeff0**

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

Units: **psia/sec/lb** **Minimum:** **na** **Maximum:** **na**

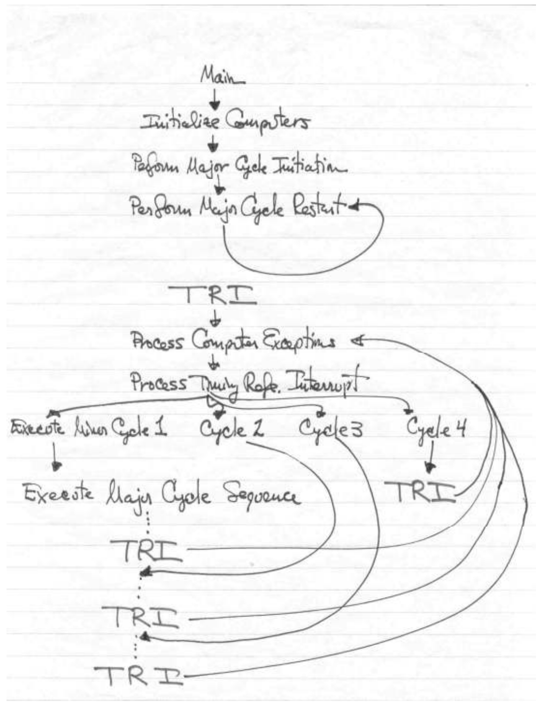
Data Type: **Constant** **Extent:** **Global** **Value:** **2.92104**

Function Name: **Calculate_Mixture_Ratio (285)**

Variable Affected	Line	How Affected	Function Performed	Candidate for Analysis
Oxidizer_Flow_Rate_Coeff	17	Assignment	f(Oxidizer_Coeff0)	no
Lox_Mass_Flow	18	Assignment	f(Oxidizer_Flow_Rate_Coeff)	no
Oxidizer_Flow_Rate	19	Assignment	f(Lox_Mass_Flow)	no
Mixture_Ratio	23	Assignment	f(Oxidizer_Flow_Rate)	yes
Mixture_Ratio	22	Boolean	f(Lox_Mass_Flow)[1]	yes

[1] If Lox_Mass_Flow is >>> then Mixture_Ratio is set to 15.999511719

Note: Mixture_Ratio is the only variable which needs trace continued for assessing impacts of Oxidizer_Coeff0



Form PLG-F21 | PLG, Inc. SFA | Sheet 1 of

Job No: | Job Title:

Task/Subject:

Prepared By: | Date: | Checked By: | Date:

TRACE of Oxidizer_CoeffØ

SEGMENTS

① Calculate Mixture Ratio 285

LINE		
17	Oxidizer_Flow_Rate_Coeff	= f(Oxidizer_CoeffØ)
18	Lox_Mass_Flow	= f(Oxidizer_Flow_Rate_Coeff)
19	Oxidizer_Flow_Rate	= f(LOX_Mass_Flow)
23	Mixture_Ratio	= f(")

Note: that Oxidizer_Flow_Rate is never used or set again in OP, why?? (ans: it is in the VDT

Limit Checks: if LOX_Mass_Flow be very large, the calculation for Mixture_Ratio might overflow. If so, Mixture_Ratio is set to a coded value of 15.9995 11719

Since Mixture_Ratio is GLOBAL, we must continue the TRACE

TRACE of Mixture_Ratio

SEGMENTS

① Perform MR Control Loop 296

Line		
12	MR_Integrator_Input	= f(Mixture_Ratio)
13	MR_Proportional_Output	= f(MR_Integrator_Input)
22, 27	MR_Integrator_Input_Past	= f(")
23, 26	MR_Integrator_Output	= f(" , MR_Integrator_Input_Pa

Note: (a) MR_Integrator_Output depends on this and previous Major Cycle's value of MR_Integrator_Input; GLOBAL, but only used locally.

(b) Both MR_Proportional_Output and MR_Integrator_Output are GLOBAL, the TRACE continues

Form PLG-F21 | PLG, Inc. SFA | Sheet 2 of

Job No: | Job Title:

Task/Subject:

Prepared By: | Date: | Checked By: | Date:

TRACE of MR_Proportional_Output, MR_Integrator_Output

SEGMENTS

① Generate FPOV Commands 303
② Initialize Control Loop 290
③ Initialize Start Engine 259
④ Perform Shutdown The Function 269
⑤ Perform Start/Main transition 261

① Generate FPOV Commands

Line		
12	FPOV_Goal	= f(MR_Proportional_Output, MR_Integrator_Ou

Limit Check: if FPOV_Goal > FPOV_Cmd_Limit, then FPOV_Goal is set to that limit.

Note: FPOV GOAL is used in Perform MR Control Loop, but not in a way that effects other variables than those already being considered.

② Initialize Control Loop

Line		
12	MR_Integrator_Output	= f(MR_Proportional_Output)

Note: nothing new will be added to the TRACE here.

③ Initialize Start Engine

Line		
7, 8	MR_Integrator_Output	= Ø
	MR_Proportional_Output	= Ø

Note: these variables are initialized only

Form PLG-F21 PLG, Inc. Sheet 3 of

Job No: Job Title: SFA

Task/Subject:

Prepared By: Date: Checked By: Date:

④ Perform Shutdown Time Function

Line 9, 10 Same as Initialize Start Engine

⑤ Perform Start to Mainstage Transition

Line 18 MR_Integrator_Output = f(MR_Proportional_Output)

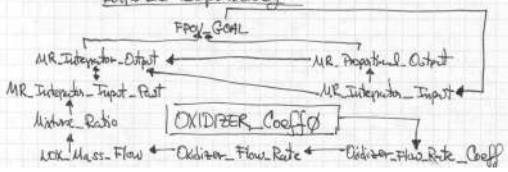
Note: Again nothing new added to path

Summary

SEGMENTS

1. Calculate Mixture Ratio
2. Perform MR Control Loop
3. Generate FPOV Commands
4. Initialize Control Loop
5. Perform Shutdown Time Functions
6. Initialize Start Engine
7. Perform Start to Mainstage Transition

Variable Dependency



Form PLG-F21 PLG, Inc. Sheet 4 of

Job No: Job Title: SFA

Task/Subject:

Prepared By: Date: Checked By: Date:

But that's not the end of it. We must make sure that none of the variables in the trace have other names equated to them.

FPOV_Goal has aliases!

(1) FPOV_Goal EQUATES TO FPOV_Goal_Ch_A

(2) FPOV_Goal is a child of Actuator_Goal

(3) FPOV_Goal_Ch_A is a child of Actuator_Goal_Ch_A

(4) Actuator_Goal_Ch_A is a child of Actuator_Goal_Ch

TRACE of Actuator_Goal

~~SEGMENT~~ REQUEST to Ramp 741

Line Actuator_Goal is set to Act_Goal

Note: no interactions or dependencies

TRACE of Actuator_Goal_Ch_A

Segment Control Actuator Positions 465

Line 14 If Actuator_Goal_Ch_A ≠ Actuator_Cmd_Ch_A
① Compute new value
② Set new value flag

52 If new value
① Send output : ~~OE~~ ISSUE_OE_Commands

→ *CHANGE in Value* Δ

Software Hazard Analysis Report

Identification No. ____________

Title

Software Affected

Mission Phase/Major Mode

Hazard Description/Discussion

Recommended Hazard Controls

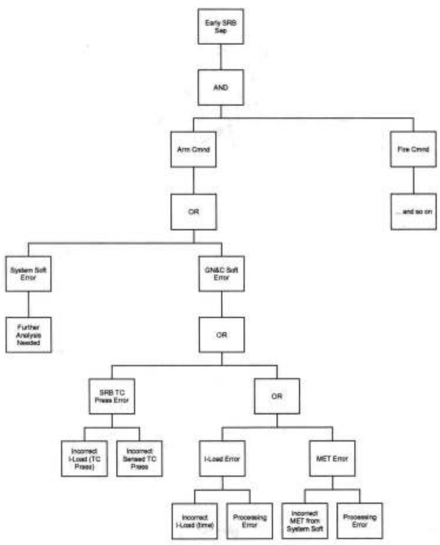
Action/Status/Disposition

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

Software Error Root Cause Analysis Report

SERCA Number:

Date/Time: **Severity of Failure:**

Failure Description:

Activity Being Performed:

Method of Detection:

Type of Failure: **Unit Complexity:** **Unit Size:**

Fill in AT LEAST ONE of the following:

CPU hours since last failure: **Runs since last failure:**
Calendar hours since last failure: **Labor hours since last failure:**
Hours/Interval and number of errors in this interval:

Type of software fix:

Fill in AT LEAST ONE of the following:

CPU hours required to fix: **Runs to fix:** **Calendar hours to fix:**

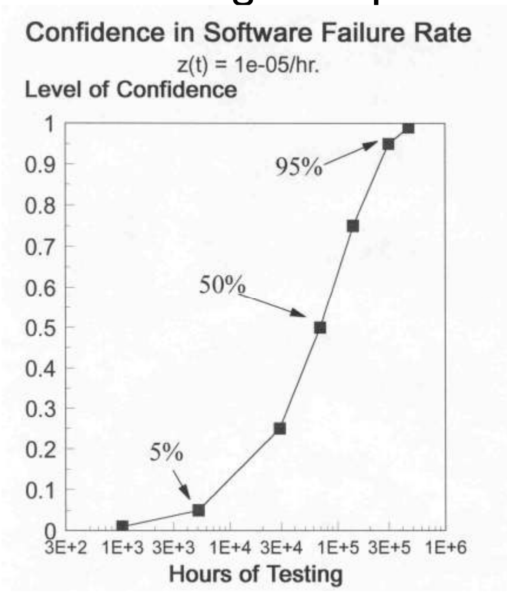
Some Observations

- 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

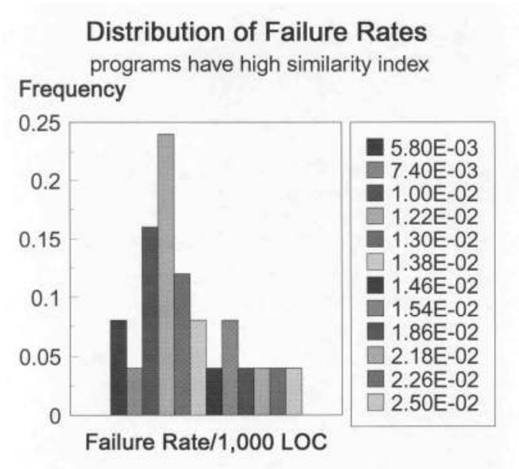
A Testing Example



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

Number of Ada Files measured : 959 FOSB21

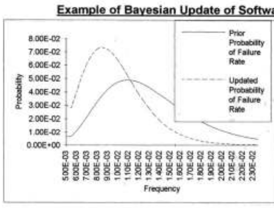
n1 (total) :	12814	n1 (average) :	13.4
n2 (total) :	24127	n2 (average) :	25.2
n (total) :	36941	n (average) :	38.5
N1 (total) :	81571	N1 (average) :	85.1
N2 (total) :	60125	N2 (average) :	62.7
N (total) :	141696	N (average) :	147.8
N^ (total) :	189260.8	N^ (average) :	197.4
V (total) :	943270.9	V (average) :	983.6
ALOC (total) :	19602	ALOC (average) :	20.4
Ss (total) :	27639	Ss (average) :	28.8
TLOC (total) :	113670	TLOC (average) :	118.5
C (total) :	4188.3	C (average) :	4.4
VARS (total) :	17930	VARS (average) :	18.7
v(G) (total) :	4401	v(G) (average) :	4.6
Lambda (total) :	4394.5	Lambda (average) :	4.6
T^ (total) :	558.5	T^ (average) :	0.6

Ada Software Metrics Descriptions:

```
n1      -  Number of Unique Operators
n2      -  Number of Unique Operands
n       -  Vocabulary (Number of Unique Operators + Number of Unique Operands)
N1      -  Total Number of Operator Occurences
N2      -  Total Number of Operand Occurences
N       -  Size (Token count = Total Number of Operator + Operand Occurences)
N^      -  Estimated Program Length
           (Length Equation = (n1 x log2(n1) + n2 x log2(n2)))
V       -  Volume (in bits = N x log2(n), A program needs approximately
           log2(n) bits to represent the Vocabulary n)
ALOC    -  Ada Lines of Code (counts 1 Ada line of code per semi-colon)
Ss      -  Lines of Code (counts all non-blank and non-comment lines)
TLOC    -  Total Lines of Code (all lines of code including blank and comment li
nes)
C       -  Language Constant (N / Ss (for Fortran C is approximately 7.0))
VARS    -  Number of Unique Variables (n2 - unique constants - labels)
v(G)    -  Cyclomatic Complexity (McCabe's metric - number of distinct basic
           paths through a program)
Lambda  -  Language Level Constant
T^      -  Estimated Programming Time (in man-hours) = Effort/Beta (where
           Beta = Stroud's Number = number of elementary mental
           discriminations/second)
```

Bayesian Update

Example of Bayesian Update of Software Failure Rate

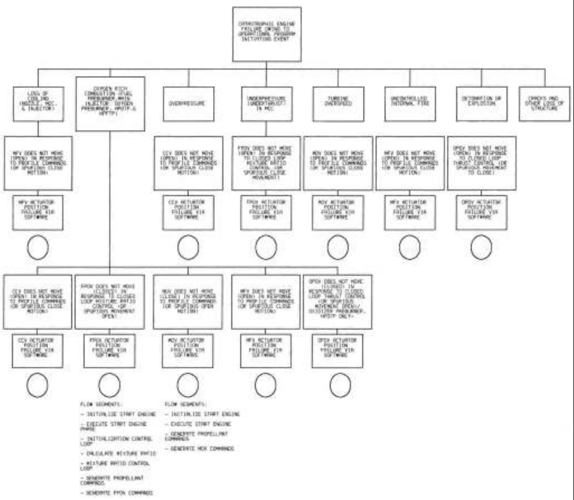


UPDATE PARAMETERS	Buttons
0 failures 250 time/trials	

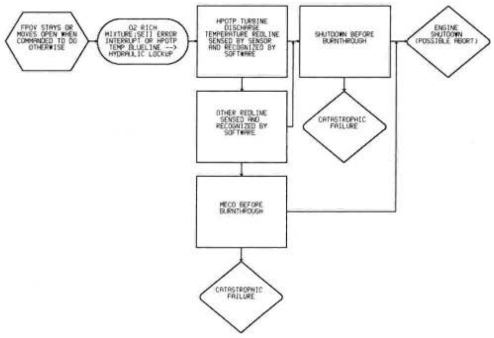
	PRIOR	UPDATE
Mean	1.23E-02	9.70E-03
Median	1.22E-02	9.03E-03
Mode	1.10E-02	8.50E-03
5th %	6.74E-03	5.35E-03
20th %	9.00E-03	6.92E-03
80th %	1.64E-02	1.17E-02
95th %	2.19E-02	1.49E-02
σ	7.39E-02	5.83E-02

Frequency	Prior Probability of Failure Rate	Prior Cumulative	Updated Probability of Failure Rate	Updated Cumulative
5.00E-03	6.42E-03	6.42E-03	3.02E-02	3.02E-02
5.50E-03	6.75E-03	1.32E-02	2.80E-02	5.82E-02
6.00E-03	1.08E-02	2.40E-02	3.96E-02	9.78E-02
6.50E-03	1.58E-02	3.98E-02	5.10E-02	1.49E-01
7.00E-03	2.13E-02	6.11E-02	6.08E-02	2.10E-01
7.50E-03	2.70E-02	8.81E-02	6.80E-02	2.78E-01
8.00E-03	3.25E-02	1.21E-01	7.23E-02	3.50E-01
8.50E-03	3.75E-02	1.58E-01	7.36E-02	4.23E-01
9.00E-03	4.18E-02	2.00E-01	7.22E-02	4.96E-01
9.50E-03	4.50E-02	2.45E-01	6.87E-02	5.64E-01
1.00E-02	4.73E-02	2.92E-01	6.37E-02	6.28E-01
1.05E-02	4.86E-02	3.41E-01	5.78E-02	6.86E-01
1.10E-02	4.90E-02	3.90E-01	5.14E-02	7.37E-01
1.15E-02	4.85E-02	4.38E-01	4.49E-02	7.82E-01
1.20E-02	4.73E-02	4.86E-01	3.87E-02	8.21E-01
1.25E-02	4.56E-02	5.31E-01	3.29E-02	8.54E-01
1.30E-02	4.34E-02	5.75E-01	2.76E-02	8.81E-01
1.35E-02	4.09E-02	6.16E-01	2.30E-02	9.04E-01
1.40E-02	3.83E-02	6.54E-01	1.90E-02	9.23E-01
1.45E-02	3.55E-02	6.89E-01	1.55E-02	9.39E-01
1.50E-02	3.27E-02	7.22E-01	1.26E-02	9.51E-01
1.55E-02	2.99E-02	7.52E-01	1.02E-02	9.62E-01
1.60E-02	2.73E-02	7.79E-01	8.20E-03	9.70E-01
1.65E-02	2.47E-02	8.04E-01	6.55E-03	9.76E-01
1.70E-02	2.23E-02	8.26E-01	5.22E-03	9.82E-01
1.75E-02	2.00E-02	8.46E-01	4.14E-03	9.86E-01
1.80E-02	1.79E-02	8.64E-01	3.27E-03	9.89E-01
1.85E-02	1.60E-02	8.80E-01	2.58E-03	9.92E-01
1.90E-02	1.43E-02	8.95E-01	2.03E-03	9.94E-01
1.95E-02	1.27E-02	9.07E-01	1.59E-03	9.95E-01

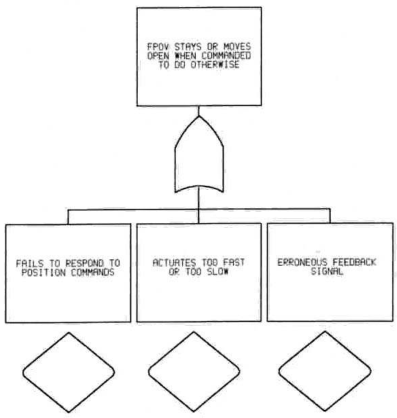
Draft MLD for OP



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