ISSRE
Software Reliability – 40 Years of Avoiding the Question

Russell W. Morris, Technical Fellow
Reliability, Maintainability and Systems Health
November 11-13, 2008
Why Software Reliability

- June 1985 - January 1987, Therac 25 Accidents (6)
- September 14, 1993 Airbus A320-211
- June 4, 1996 Ariane 501 Disaster
- May 2002, Air-traffic control software reliability
- June 2002, Impact of inadequate software testing on US economy,
- June 2002, Questions about New Air-Traffic Computer System,
- 2005-Present Unprotected personal data – examples include; Boeing, Visa, Sears and …
- Airbus A319, Flight 190, heading from Victoria to Toronto, Pilot came over the intercom to say there had been a computer failure and that they were flying the plane manually.
- August 2008, Atlanta Center - Software corruption results in loss of one-half of flights in the eastern united states and overloads the Salt

SWR has become the most critical element of today's capabilities

Excerpts from http://catless.ncl.ac.uk/Risks/index.25.html
Common Myth

The designers view

*Software is 100% reliable - i.e., it does exactly what it was designed to do.*
Avoiding the Question

• Software started much later than Hardware in a world that was hardware oriented
• Difficult for Programs to wrap their arms around Software having a reliability metric
• Commercial industry – very little failure reporting
• Focus has been hardware and only in the last 10 years has software seriously been moved to the forefront of reliability issues
• Systems models have been difficult to generate
• Field Failures are often not specifically identified as being software related
• Shear Number of languages creates problems – especially for system integration
### Sample of Software Languages

<table>
<thead>
<tr>
<th>Languages</th>
<th>Circa</th>
</tr>
</thead>
<tbody>
<tr>
<td>FORTRAN</td>
<td>1954</td>
</tr>
<tr>
<td>ALGOL</td>
<td>1958</td>
</tr>
<tr>
<td>LISP</td>
<td>1958</td>
</tr>
<tr>
<td>PASCAL</td>
<td>1970</td>
</tr>
<tr>
<td>PLM</td>
<td>1972</td>
</tr>
<tr>
<td>C</td>
<td>1976</td>
</tr>
<tr>
<td>ADA</td>
<td>1978</td>
</tr>
<tr>
<td>C++</td>
<td>1979</td>
</tr>
<tr>
<td>FORTH</td>
<td>1987</td>
</tr>
<tr>
<td>PASCAL</td>
<td>1983</td>
</tr>
<tr>
<td>PERL</td>
<td>1980’s</td>
</tr>
<tr>
<td>VISUAL BASIC</td>
<td>1991</td>
</tr>
<tr>
<td>JAVA</td>
<td>1995</td>
</tr>
<tr>
<td>C#</td>
<td>2003</td>
</tr>
</tbody>
</table>

There are over 100 languages used in a large number of applications.
Software is Ubiquitous

- Software today is showing up in almost every imaginable market
  - Ovens
  - Microwaves
  - Automobiles
  - Toys
  - Medical Equipment
  - Aircraft
  - Desktop and Laptop Computing
  - Cell Phones
  - Military Weapons
  - And more – much more
Software Reliability, Quality and Safety

• Software Reliability is often very closely tied to the safety of many systems. And these systems must be viewed in aggregate:
  • Commercial aircraft (<10^{-9}) – flight control
  • Military Aircraft (<10^{-5}) - flight control
  • Medical (<10^{-7}) – accuracy of diagnosis – patient safety
  • Automotive (<10^{-7}) – passenger safety
  • Military fire control systems – friendly fire

• Software Reliability The probability that the software will not cause the failure of the system for a specified time under specified conditions. This is based on system usage and an estimate of the number of faults built into the software.

• Software Quality The application of control and disciplined to the execution of the software process such that all product requirements are met. Ex; CMMI level 3
System Reliability Engineering

Software can only be understood in the context of the target system, end user and application.
Review of an existing System

• Period covered 1 year
• All maintenance events
  • Failure is assumed if maintenance action initiated
• Assess for source
  • Hard failure
  • Soft failure
  • Operator failure
• Assumes average of 9 hour a day operation 5 days a week for 50 weeks per year
One Year Assessment of large system

System with >150,000 processor based systems within its operations.
  • Number of Calls
  • Types of Calls
  • System Affected (i.e., SW, HW, both)
  ...

Processor Based Applications
  • Network
  • Desktop
  • Laptop
  • CAD
  • ...

<table>
<thead>
<tr>
<th>Count of devices</th>
</tr>
</thead>
<tbody>
<tr>
<td>101128</td>
</tr>
<tr>
<td>51675</td>
</tr>
<tr>
<td>152803</td>
</tr>
</tbody>
</table>

MTBME – Mean Time Between Maintenance Events
One Year Assessment of large system

Data reviewed for relevance and applicability to SW and HW

<table>
<thead>
<tr>
<th>Category</th>
<th>Failures</th>
<th>Intensity</th>
<th>Fraction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loss System</td>
<td>1834</td>
<td>7.7e-6 /hrs</td>
<td>0.0203</td>
</tr>
<tr>
<td>Loss Group/Site</td>
<td>178,000</td>
<td>7.4e-4 /hrs</td>
<td>0.52</td>
</tr>
<tr>
<td>Loss Individual</td>
<td>13,269</td>
<td>5.6e-5 /hrs</td>
<td>0.04</td>
</tr>
<tr>
<td>Nuisance</td>
<td>151,412 (2)</td>
<td>6.4e-4 /hrs</td>
<td>0.44</td>
</tr>
</tbody>
</table>

Estimated Run Time

<table>
<thead>
<tr>
<th>Category</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware MTBME</td>
<td>597 hours</td>
</tr>
<tr>
<td>Software MTBME</td>
<td>339 hours</td>
</tr>
</tbody>
</table>

(1) There are approx 15 major networks operating – 24/7 – assuming that there are no duplicate calls

(2) There may be an additional 300,000 “nuisance” calls to add to the Cat 4’s
Failure Definition

Set and shape expectations by the early definition of what is or is not a failure

• **Failure**  The inability of a system or system component to perform a specified function within specified limits. A failure may be produced when a fault is encountered and a loss of expected function to the user results; or the termination of the ability of a functional unit to perform its required function; or a departure of program operation from program requirements.

• **Failure Rate**  Normally expressed as failures per unit time such as CPU time or operating hour. (derived as a result of an anticipated failure density (faults per KSLOC) and the processor (speed and fault exposure rate)) or the number of failures divided by the time interval over which the failures occurred.

• **Reliability**  The ability (probability) of a system or component to perform its required functions under stated conditions for a specified period of time.

Notice – no distinction as to what caused the failure
Hardware Reliability – Physics Based

• Temperature
• Sand and Dust
• Vibration
• Shock
• Power cycling
• Transient temperature
• Aerosols…, …

Mathematical Models Include Weibull, Exponential, Log-Normal and Normal.

In other words the physics of the user and operational environment
Software Reliability – Process Based

• Use of existing or off-the-shelf (OTS) SW
• Development of Requirements
• Skill level of designer
• Type and Level of testing
• Understanding concept of operation (flight control, etc.)
• Complexity of function
• What is the work schedule
• Amount of human interface, …, …

In other words the non-quantifiable user environment
Requirements Failures

• Major SW failures occur when requirements are not understood by Customer and Contractor.
• Requirements failures often do not show up until after the product is released.
  • Ariane V
  • A320 Airbus
  • Mars Probes
  • Autonomous Vehicle failure in a desert test
Reliability and Maintainability Requirements Tree – an example

- **Environments**
  - Operational Profile
    - MTBF\textsubscript{inherent}
    - MTBF\textsubscript{induced}
    - MTBF\textsubscript{CND}
    - MTBSM
  - MTBCF
  - MTBOMF
  - MTBFA
  - MTBMA

- Mission Duration
- Mission Reliability
- Architecture
- PLOA
- Fleet Size
- Availability\textsubscript{operational}
- MLDT

- MTBS/PM
- MTBF\textsubscript{dormant}

- MTBF\textsubscript{inherent}
- MTBR
- MTTR\textsubscript{inherent}

- Availability\textsubscript{inherent}
- MMH/OPH
- Crew Size

- MMax

- FD/FI
- Accessibility

Given
- Derived

Off the shelf fill rate

Operational Profile

Environments

Operational Profile

MTBF\textsubscript{inherent}

MTTR\textsubscript{inherent}

MMH/OPH

Crew Size
Software Reliability Drivers

- SLOC
- Execution Time
- Complexity
- Maturity of Designers
- Familiarity with Operation
- Understanding Requirements
- Level of Detail of ICD
- Development of Requirements

- Type and Level of Testing
- Language(s) used
- Use of Error Trapping
- Exception Handling
- Skill level of designer
- Is it Monday morning or Friday afternoon
- What is the work schedule
- Amount of human interface . . . . . . . .
Software Reliability Concerns

• Common Software
  • Common software will fail identically for the same set of input conditions – hardware, as a general rule, won’t
  • Has a potential impact on planned redundancy and or fault management.
• No data to indicate the number of ‘common’ problem calls
• May represent 50% or more of the total number of problem reports
• Relies on the capabilities of the technical force (highly variable)
• Etc.
Elements of SW Failure

1. errors are the source of defects,

2. defects are faults in code (or data),

3. faults eventually lead to failures in the system

4. Failures in the system take on a variety of attributes – generally all bad
Errors and other Problems

- Requirements
- Usage
- Complexity
- Software Specifications
- Design
- Testing
- Maintenance
- System Integration
- User

- COTS integration
  - Unknown design rules
  - Sometimes unknown language
  - Little or no documentation
  - HW differences
Reliability Program Plan

A Reliability Program is a structured approach to identify and track software or hardware reliability metrics using consistent, traceable, and repeatable processes.

- Define Requirements
- Establish how the requirements will be met
- Demonstrate that the requirements have been met

SAE JA 1000, SAE JA 1002, SAE JA1003
Define Requirements

• It is an imperative for SW, more so than HW, to ensure that the requirements are understood by both the customer and the contractor or supplier.

• All subsequent effort is often focused just on compliance to the requirements. Failures associated with requirements often do not manifest themselves until after delivery and are extremely expensive to fix.

**THE** major cost to most programs is failure to either properly define the requirement or correctly interpret the requirement. The result of this is that the problems are not found until **AFTER** the design is complete and it is in the field.
Define Metrics For Assessment

- MTBF (mean time between failure)
- MFHBF (mean flight hour between failure)
- MFHBOMF (mean flight hour between operational mission failure)
- Failure Intensity – The number of expected failures per unit time.
- P(t) (Probability of success with time) for xx hours under battlefield conditions
- Q(t) (Probability of failure with time) for yy hours under office environment
- Probability of successfully operating with a given confidence level
- MTBSCF Mean Time Between Critical (or safety critical) failure

The statistics of the data give you the confidence in the ability of the system to perform to a specification.

In a stochastic process there is no such thing as failure free – it can not be tested, verified or validated.
Meet the Requirements

• Establish an integrated Reliability Program Plan
• If a Software Development Program (SDP) exists, add the software tasks, metric collection to SDP or provide a relationship to both the SDP and the integrated Reliability Program Plan. Define the Hardware and Software tasks and metrics to be developed and tracked
• Obtain customer approval of plan
• Establish regular assessments and reviews with customer
Capability Maturity Model Integration CMMI<sup>SM</sup>

**Staged Level Model**
- Level 1 = Initial (Chaos)
- Level 2 = Managed (individuals)
- Level 3 = Defined (institutionalized)
- Level 4 = Quantitatively Managed
- Level 5 = Optimizing (Improvements are fed back to processes)

**Continuous Level Model**
- Level 0 = Incomplete
- Level 1 = Performed
- Level 2 = Managed
- Level 3 = Defined
- Level 4 = Quantitatively Managed
- Level 5 = Optimizing

*Quality processes do not guarantee a reliable product*
Design for SW Reliability

- Understand the System SW requirements
- Understand the customer’s need and usage
- Plan for CMMI level 3 or better (controlled process)
- Integrated effort focuses on parametric testing
- Fault tolerance
- Fault or exception handling
- Understanding Validation & Verification requirements
- Peer reviews
- Lab testing – Monte Carlo – nominal case used only to validate basic calculation correctness
- Self detection
- Design in Error Trapping (inputs, outputs, time of occurrence, etc.)
- Software is very sensitive to how it is used
SW Allocations

• Benefits
  • Provides a starting point for supplier assessment
  • If properly backed by engineering attributes it can be used to form the basis of a SWR specification

• Drawbacks
  • COTS is a major unknown
  • Excessive factors affecting estimation process
  • Contracts have already been let

• Concerns
  • Variability of the SW/HW ratio
SWR Models

- Jelinski and Moranda (Jelinski, 1972)
- Schick and Wolverton (Schick, 1978)
- Jelinski-Moranda geometric (Moranda, 1979)
- Moranda geometric Poisson (Littlewood, 1979)
- Modified Schick and Wolverton (Sukert, 1977)
- Goel and Okumoto imperfect debugging (Goel, 1979)
- Nonlinear Regression
- Belady and Lehman (Belady, 1976)
- Miller and Sofer (Miller, 1985)
- Coutinho model (Coutinho, 1973)
- Wall and Ferguson model (Wall, 1977)
- Musa exponential (Musa, 1987)
- Goel and Okumoto NHPP (Goel, 1979)
- S-shaped growth (Ohba, 1984; Yamada, 1983, 1984a)
- Hyperexponential growth (Huang, 1984; Ohba, 1984; Yamada, 1984b)
- Discrete reliability growth (Yamada, 1985)
- Markov with imperfect debugging (Goel, 1979)
- Littlewood Markov (Littlewood, 1979)
- Software safety (Tokuno, 1997; Yamada, 1998)
Each type of testing is limited by the number of errors that can be exposed due to the limits of the test set-up.

The biggest challenge for software reliability will be reducing interoperability defects.

### General Characteristics

<table>
<thead>
<tr>
<th></th>
<th>CSCI Reliability est.</th>
<th>System Reliability est.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logic errors</td>
<td>.995</td>
<td>.90</td>
</tr>
<tr>
<td>Coding errors</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rqmts errors (SRS)</td>
<td>.9995</td>
<td>.99</td>
</tr>
<tr>
<td>Logic errors</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rqmts errors (SSS)</td>
<td>.9999</td>
<td>.995</td>
</tr>
<tr>
<td>Interop. errors</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Timing errors</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Resource errors</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fielded reliability is here.
RiAC Prediction Method

$$\lambda_o = \frac{Kr_i W_o}{I}$$

where

$ri$ = host processor speed  
$K$ = Fault exposure ratio – default value of $4.2 \times 10^{-7}$  
$W_o$ = Estimate of the number of initial faults  
The initial default value is 6 per 1000 LOC  
$I$ = Number of object instructions times the expansion ratio – 4.5 for Ada ($I = SLOC \times Er$)

$$\lambda(t) = \lambda_o \exp^{-\{\beta t\}}$$

where

$\lambda(t)$ = projected reliability failure rate at time $t$  
$\lambda_o$ = initial failure rate  
$t$ = CPU execution time  
$\beta = \frac{B \lambda_o}{W_o}$

where

$B$ = fault detection ratio – default is 0.955

$\lambda_o$ is the initial failure rate based on the history.  
$\lambda(t)$ is the failure rate at time ‘$t$’.  
$\lambda$ is the Failure Rate from release to end of life.

System Reliability toolkit, Reliability Information Analysis Center, Utica, NY, 2005  
After Musa 1999
Model Limitations

• Majority of models assume apriori knowledge of the number of faults or fault density
• Models do not account for SW development process of adding modules or CSCIs with each build
• Even with intensive effort – at the end of the development period \( F(t) \neq 0 \)

Since there will be faults present after the SW is released – the customer will try to eliminate these. To find correspondingly smaller number of defects in very large population will take inordinate amounts of time – 80/20 rule and money.
CASRE (Computer Aided Software Reliability Estimation)

- CASRE incorporates the mathematical modeling capabilities of the public domain tool SMERFS (Statistical Modeling and Estimation of Reliability Functions for Software), and runs in a Microsoft Windows environment. (Open Channel Foundation)
- Data analysis to find best fit curve for the various models available – based on a number of faults found and corrected.
- Acts very much like hardware failure data analysis
  - Uses time of arrival
  - Level of effect (data parsing)
  - Demonstrates Reliability growth through integration of time to failure metric

http://www.openchannelfoundation.org/projects/CASRE_3.0
Assure the Requirements have been Met

• Assist with the development of the Software Problem Report form and method to include Reliability elements.
• Participate in the test plan development and understand what is being tested and how that relates to the agreed to metrics
• Acquire data from SPR process and develop reliability metrics and reports
• Software Change Requests – SCRs – are a requirements or specification change that must be reassessed.
Software Validation and Verification

Software is verified or validated by Testing, Simulation and or Demonstration

- Assessment of Compliance
  - Peer Reviews
  - Fault Trees and FMEAs
- Testing
  - Formal Methods
  - Tables
  - Scenario-Based Testing
  - Thread-Based Testing
  - Task-Based Testing
  - Random Testing
  - Monte Carlo

No sequence of testing or reviews provides 100% coverage

<table>
<thead>
<tr>
<th>Defect Removal Activity</th>
<th>Defect Removal Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Informal design reviews</td>
<td>25% to 40%</td>
</tr>
<tr>
<td>Formal design inspections</td>
<td>45% to 65%</td>
</tr>
<tr>
<td>Informal code reviews</td>
<td>20% to 35%</td>
</tr>
<tr>
<td>Formal code inspections</td>
<td>45% to 70%</td>
</tr>
<tr>
<td>Unit test</td>
<td>15% to 50%</td>
</tr>
<tr>
<td>New function test</td>
<td>20% to 35%</td>
</tr>
<tr>
<td>Regression test</td>
<td>15% to 30%</td>
</tr>
<tr>
<td>Integration test</td>
<td>25% to 40%</td>
</tr>
<tr>
<td>System test</td>
<td>25% to 55%</td>
</tr>
<tr>
<td>Low-volume Beta test (&lt; 10 clients)</td>
<td>25% to 40%</td>
</tr>
<tr>
<td>Hi-volume Beta test (&gt;1000 clients)</td>
<td>60% to 85%</td>
</tr>
</tbody>
</table>

All Phases Total:
- Military Average: 0.96
- System Software: 0.94
- Commercial Software: 0.90
- MIS: 0.73
SW Reliability Growth Process

The software data acquisition process is used to establish the reliability of the as designed product. It generally includes elements of:

– Total module or CSCI SLOCs
– Total Failures to date
– Failures fixed to date
– Total run time of the system
– Number of modules

![Basic SW Reliability FRACAS Process Diagram]

Failure Review Board
• Relevant
• Non- Relevant
• Chargeable and Non-Chargeable

Program Execution Time (Cumulative)

Observed Failures in Each Time Interval
(Failures/Unit Time or Failure Intensity)

Prediction Model

Current Failure Intensity

Failure Intensity Objective

NOTE: assumes defects are being removed over time
SW reliability failure rate data tends to exhibit an exponential decay related to the TAAAF (test analyze and fix) approach where failures are identified and removed.
Warning re: Off-The-Shelf

- Software Reliability is *grossly* affected by the use of existing, off-the-shelf packages and how it is used. For off the shelf:
  - There is little or no input to the design process,
  - No assessment of the development (TAAF) process and
  - No data to indicate the actual level of reliability received or achieved.

- SW has 3 different dominate types of faults that compound any analysis
  - Requirements
  - Interface
  - Usage

Many programs have started with aspirations of using greater than 80% reuse and have ended up with less than 15% at the end of the D&D period.
A Few Final Observations

• Without data the best that can be hoped for is calibrated thumbnail held at arms length
• Failure to think system from the concept phase will result in an incomplete system at delivery
• The customer needs to be as smart as the supplier or contractor – that is all levels of customer
• Engineers can make anything work - Once
Summary

• The last 40 years have seen huge changes in the art and science of computing – unfortunately SW has not kept pace with hardware
  • The intent of the Software Reliability effort is to identify and repair latent faults before the equipment is fielded.
  • Design assessments and predictions are needed to provide more than the warm fuzzies that SWR is being accounted for in the overall system

BUT

You can prove that Software faults do exist, you can not prove that they do not exist!
Myth - Busted

The Customer’s View
(And Reality)

The customer doesn’t care what failed or why –
If it failed - it is **UNRELIABLE!**
References

• Neumann, Peter, http://catless.ncl.ac.uk/Risks/index.25.html, Stanford University - RISK site provides a list of risks, including software related – many are security and safety risks.
• SAE JA 1002 Software Reliability Program Standard
• SAE JA 1003 Software Reliability Program Implementation Guide
• AIAA R-013 1992 Recommended Practice Software Reliability
• Mil-Hdbk-338 Electronic Reliability Design Handbook
• IEEE Standard 610.12, Standard Glossary of Software Engineering Terminology
• Safeware, Nancy G. Leveson, Addison-Wesley, 4th printing, January 2000
• 830-1998, Recommended Practice for Software Requirements Specifications;
• 982.1-1988, Standard Dictionary of Measures to Produce Reliable Software;
• Rosenberg L., Hammer T., Shaw J., Abstract, Software Metrics and Reliability, 9th International Symposium, November, 1998, Germany
Contact Information

Russell W. Morris
206-655-5127
800-344-8615
russell.w.morris@boeing.com
MS 4C71, PO Box 3707
Seattle WA, 98124
Abstract

During the last 40 years, there has been an explosion of technical complexity of both hardware and software. Hardware, characterized by physics and empirical data, provides the reliability and systems engineers with a capability to estimate the expected reliability. Software has however managed to avoid this type of model development in part because the factors affecting reliability are not measurable by physical data. Software reliability is characterized by data gathered during systems integration and test. This data has attributes and parameters such as defect density, capability of programming of the software engineering team, the experience of the engineering team, the understanding of the application be the designers, the language used and more. Models developed by J. Musa, N. F. Schneidewind, B. Littlewood, K. Okumoto, et al, make use of both target platform characteristics as well as the history of the product as it proceeds through the design, and systems test and integration. The latter is the one element that has the most significant impact on the reliability of the final product, and is the one that has the least amount of supplier data. Software reliability is more than the processes advocated by CMMI (Capability Maturity Model® Integration) and is susceptible to esoteric and infinitely harder parameters to measure. Many suppliers do not keep data on the development metrics that provide insight into the reliability of their product. This greatly affects the reliability of products such as commercial off the shelf (COTs) equipment. The author discusses some of the elements that affect software reliability and compares some of the differences when trying to estimate reliability of today’s systems.