The Procurement of Software Dependent Systems

Making Systems Reliable through Software Reliability Engineering Techniques

13th MoD R&M Specialists' Seminar

April 3-4, 2003

Abbey Hotel

Malvern, Worcestershire, UK

Dr. David E. Peercy
Sandia National Laboratories
Chair, Society of Automotive Engineers G-11SW Committee
PO Box 5800, MS-0638
Albuquerque, New Mexico 87185-0638 USA
505-844-7965
depeerc@sandia.gov

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Presentation Objective

Provide some thoughts and discussion on the following questions:

1. What is procurement of a system?
2. What makes a system software dependent?
3. What does software reliability have to do with procurement of a software dependent system?
4. Can you provide me with an example of such a procurement?

Topics

- Procurement of a Software Dependent System
  - System Procurement and Software Dependence
  - Goals & Objectives of a Software Reliability Program
  - Examples of Software Failures
    - Myth: Software Reliability = 1 (can't fail)
- Software Reliability Program
  - SAE G-11 Software Committee Standards/Guides
    - JA1002, "Software Reliability Program Standard" July 1998.
    - JA1003, "Software Reliability Program Implementation Guide" (2003 Draft)
  - Principles: determine, meet, demonstrate customer requirements
  - Framework: Plan-Case structure
- Case Study Example
  - Aerospace Certification

System Procurement and Software Dependence

- What is procurement of a system?
  - Buy it with a purchase request, support contract
  - Build it through a contractual mechanism and development oversight
- What makes a system software dependent?
  - The system can not meet its requirements without the functionality provided by the software component
  - Dependency ranges from a low level to very high level of criticality
- What does software reliability have to do with procurement of a software dependent system?
  - System might fail due to execution of a software fault: impact can range from low to a very critical
  - So…. just how does one address this system failure due to software?

Goals and Objectives

Software Reliability Program

Given a system for which you must determine an approach to software reliability, you should be able to:

- Identify important software reliability issues
  - related to the software engineering life cycle, system engineering life cycle, and the system/software reliability engineering activities
- Plan for & provide evidence of software reliability
  - Understand methods for acquiring, engineering, and sustaining a software reliability capability
  - Determine performance-based software reliability requirements
- Derive elements of a software reliability program
  - Characteristics of design for reliability
  - Characteristics of operational reliability
  - Customer-supplier relationships: what's good enough?

Some Questions of Interest

- Why should software reliability be emphasized?
- How much does software reliability engineering cost?
- What are software reliability performance measures?
- In what system engineering phases is it important to consider software reliability issues?
- What are software reliability activities?
- Who conducts software reliability activities?
- What should a customer expect a supplier to provide as evidence of software reliability?
- What role does a certification authority play in all this?

Software Failure Examples

http://catless.ncl.ac.uk/Risks
Peter Neumann, Stanford University Professor
RISK site provides a voluminous list of risks, many of which are computer/software related - primarily interested in security and safety risks; summaries are provided with links to more detail.

- Therac 25 Accidents (6), June 1985 - January 1987
- Airbus A320-211, September 14, 1993
- Ariane 501 Disaster, June 4, 1996
- Friendly Fire Deaths, March 2002
- Air-traffic control software reliability, May 2002
- Impact of inadequate software testing on US economy, June 2002
- Army Training Accident, June 2002
- Questions about New Air-Traffic Computer System, June 2002
- Software "glitch" Changes the Colour of the Universe, March 2002
- Fun with Microsoft error messages

SAE G-11SW
Software Reliability Program

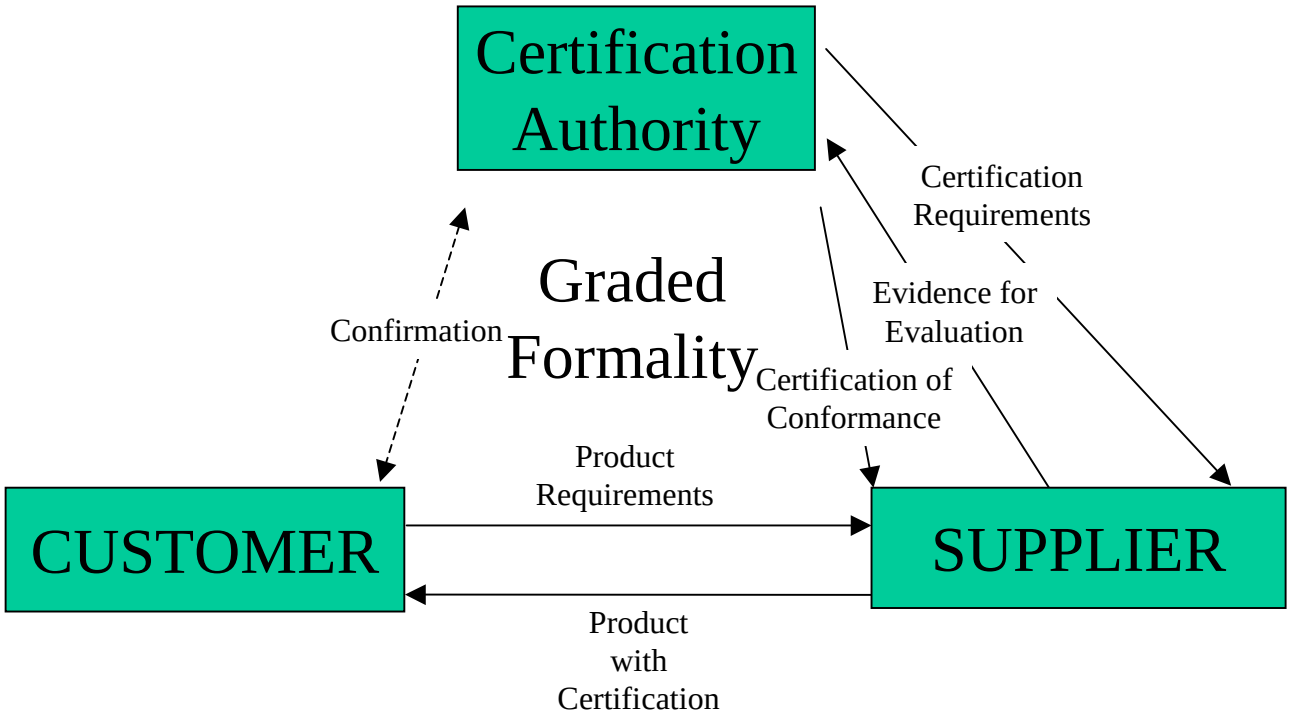
- JA1002
  - Published by SAE in 1998
  - Developed by G-11 RMSL Software Committee
  - Copies available from SAE (http//www.sae.org)
  - Basis for some aspects of example case study
- JA1003
  - Implementation guide for JA1002
  - Many methods and techniques are described along with application guidance for a software reliability program
  - Status - JA1003 full draft for ballot review by full G-11SW Software Committee in spring 2003

Example Plan and Case Templates

Why Set up a Software Reliability Program?

- Ensure product reliability meets user needs
  - estimate/predict software and integrated system reliability
- Improve time to market for products
  - detect and prevent propagation of development/support defects
  - improve test process and reduce extraneous testing time
- Reduce product cost
  - reduce defects, time to develop, and corrective maintenance
  - improve productivity
- Improve customer satisfaction
  - reduce delivered defects, target specific high priority areas
- Reduce/mitigate risks
  - target specific high risk functional areas - safety, security
  - reduce likelihood of defects being delivered to the customer

Customer-Supplier-Certification Authority

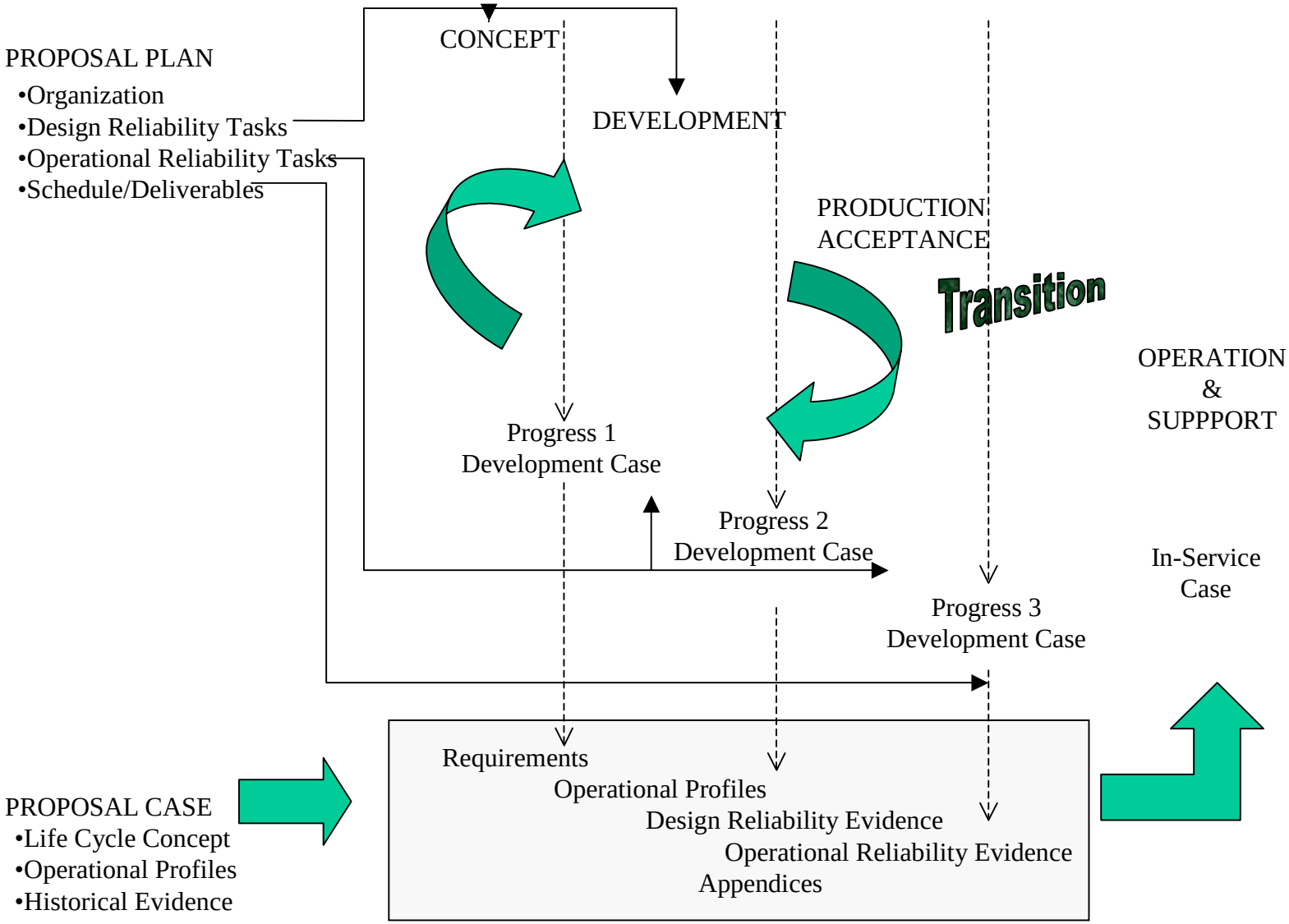


Bottom Line

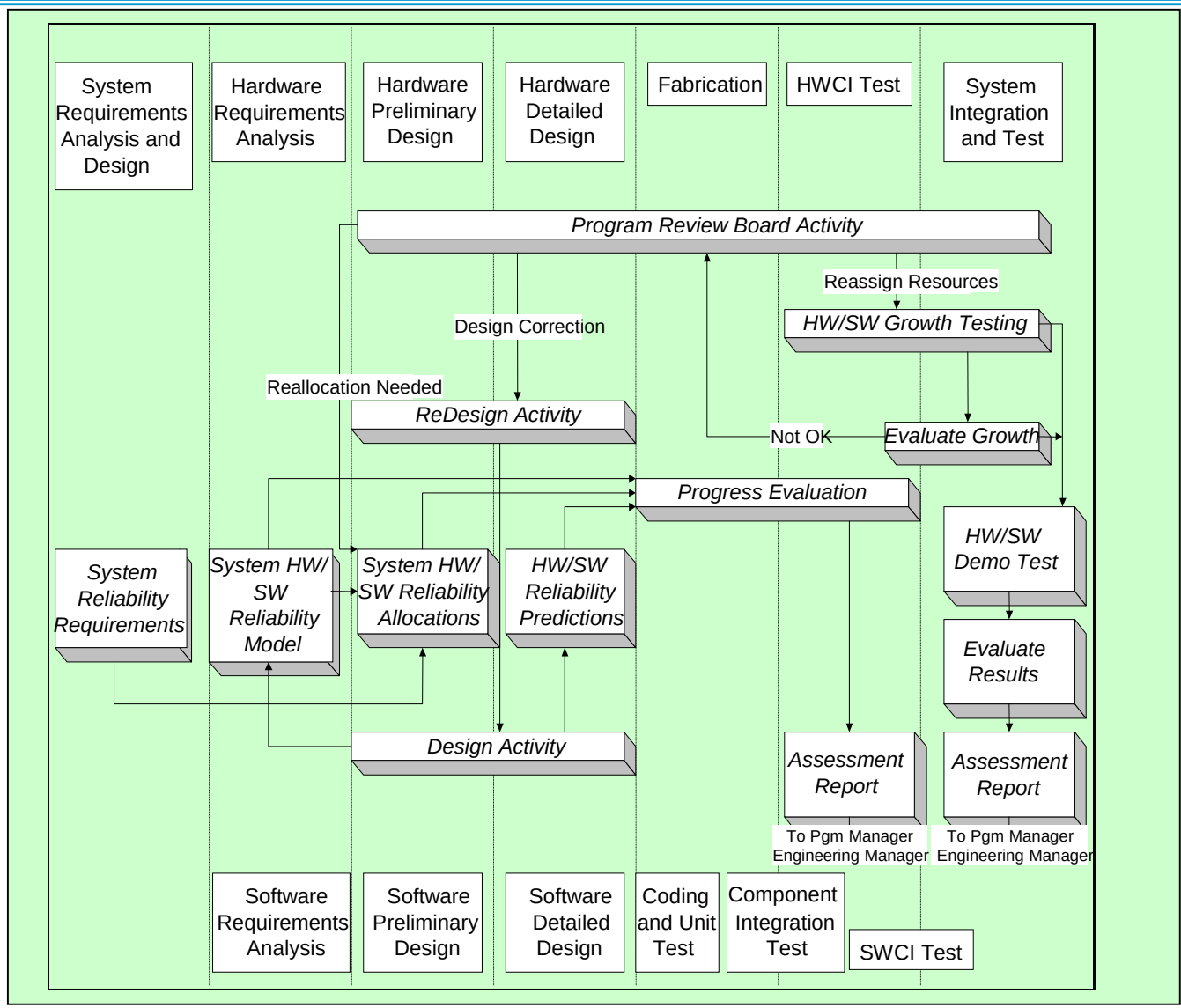
What [software reliability] evidence is appropriate for the product?

How do we know this evidence is good enough?

Conceptual Framework
Reliability Plan-Case



System Reliability Tasks



Bottom Line Objective
Reduce Programmatic Risk

- Program Decision Process
  - Defect tracking/reliability tracking supports phase/iteration completion decision
  - Defect tracking/reliability supports prioritization of activities
- Program Key Progress Parameters
  - Defect tracking/reliability supports schedule progress estimation
  - Early defect removal supports higher likelihood of meeting schedule
- Program Key Cost Parameters
  - Defect tracking/reliability supports effort estimation
  - Early defect removal means less rework in later phases
    - factor of 10 by phase in effort to remove defects
- Program Quality Indicator
  - Defect tracking/reliability provides a key indicator of product quality

Bottom Line Objective
Improve Customer Satisfaction

- Customer Issues
  - Performance - Reliability/Meets Customer Expectations
  - Schedule - On Time
  - Effort - Within Budget
  - Risk - Managed for Change

To Summarize

Software reliability program provides:

- *Customer Focus*
  - *balances customers' reliability needs with their desire for functionality, low cost, and timely delivery*
- *Planning*
  - *identifies which methods are used for defect prevention, detection, and removal throughout the life cycle in order to understand where defects are introduced and determine where failures might be triggered*
- *Case Evidence*
  - *measures of defects and failures with respect to prevention, detection, and removal throughout the life cycle*
  - *confidence level that the software will not contribute to any system failures over a specified time and specified operational use*

Backup Slides

- Terminology (Reliability, Failure/Fault/Error)
- Software Failure Categories (RTCA DO178)
- System HW/SW Reliability Program Relationship
- Design for Reliability Focus
- Defects: across life cycle, release defect density; tracking defects, delivered defects
- SEI CMM Level, Example Methods, FMECA/FTA/FRACAS
- Operational Reliability Focus, Failure Intensity, Reliability Growth Model
- System/software reliability integration, HW/SW Reliability (AND, OR), Simple Exercise
- Software Reliability Model (Musa), Equations, Simple Example
- References

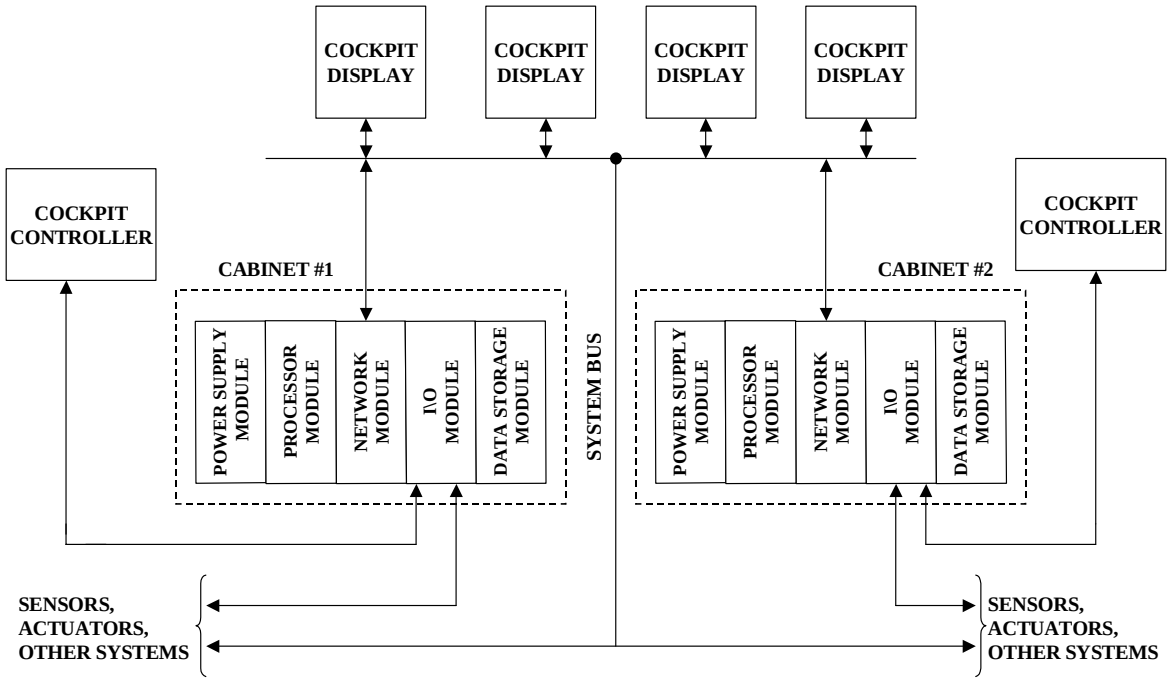
Case Study in Software Reliability

Case Study Objectives

- Define “Simple” Software Reliability Plan
  - Customer and Supplier Viewpoint/FAA Certification Context
  - Define Elements of Software Reliability Plan
  - Design Activities
  - Test Activities
  - Certification Activities
  - Operational Activities
  - Support Activities
- Provide Hypothetical Software Reliability Case
  - Design Defect Data and Analysis
  - Test Defect Data and Predicted Reliability
  - Operational FRACAS data gathering and Estimated Reliability Growth
  - Support Block Releases with Updated Reliability Predications

Example IMA System With Hardware Elements

- Integrated Modular Avionics (IMA) hardware element



- "Load Control (LC)" software program is in a PROM in the Processor Module
  - purpose is to load operational software into the Processor Module and ensure load is completed successfully

IMA Assumptions

- IMA Inc is Supplier/Manufacturer
  - Delivers complete IMA - hardware element including Load Control SW
  - Focus for this case study is on the Load Control software program delivered as part of the IMA element that enables the Processor Module to be loaded with operational software
- NSIA Air is Customer
  - Establishes contractual relationship with IMA, Inc to receive IMA units for aircraft system integration during production builds
  - Accepts IMA product in accordance with contractual acceptance criteria
  - Performs appropriate systems integration, checkout, and flight testing during system design and production/manufacturing process
- FAA is Certification Authority
  - Certifies to Technical Standard Order TSO-C153, IMA Hardware Elements
    - Includes certification of the Load Control software to DO-178B requirements and alternative elements in accordance with the Certification Authorities Software Team (CAST) publication [CAST5-00] “Guidelines for Proposing Alternate Means of Compliance to DO-178B,” June 2000.

NSIA Air and IMA,Inc Roles

- NSIA Air Customer
  - Requires certification by FAA (HW and SW)
    - Per TSO-C153
      - If software to enable future software loading and/or electronic part marking is included in the hardware element, the software level applied under this TSO must be commensurate with the installation safety assessment and documented in the installation procedures and limitations.
    - Per RTCA DO178B for Load Control software
      - Plan for Software Aspects of Certification (PSAC), Software Configuration Index, and Software Accomplishment Summary
    - Per Various FAA Guidelines and CAST position papers (e.g., CAST-5, Alternatives)
  - Requires certain Key Performance Parameters be met
    - Per contractual agreement between NSIA Air and IMA
    - One KPP is IMA reliability performance allocated to Load Control software
      - Requires evidence of how the software reliability has been integrated into the KPP
    - Requires Supplier Survey to review certification evidence & how the KPPs have been met
- IMA Contractor
  - Develops and Supports IMA product; uses Product Service History data
  - Coordinates with the NSIA Air contact at Aircraft Certification Office (ACO) – part of FAA Certification Authority
    - Obtains IMA certification - including Load Control software
  - Completes NSIA Air Supplier Contractual Requirements - Survey
    - Specifically, provides SW reliability case evidence as part of the SW verification plan

Load Control Software Concept

- Initial delivery of the IMA unit
  - Load Module software package in a PROM chip within the Processor Module.
  - IMA Processor Module designed for operational software replacement; design is double NVRAM memory concept; initial operational software is loaded into NVRAM location #1; subsequent updated operational software is loaded into “other” NVRAM location.
    - First updated version of operational software is loaded by the Load Module software into NVRAM location 2 when activated by an external command to the Processor Module; next updated version is loaded in NVRAM location 1, and so forth.
  - When activated, the Load Module software
    - authenticates the new operational software using digital signature technology
    - loads the authenticated software into the “other NVRAM location” and switches the “active location” for the software to this NVRAM location.
    - if load is successful sends “successful” message to the “external controller”
    - if load is not successful, the “active location” is not changed and a “failed” message with applicable reason is sent to the “external controller”

Assumptions

- Load Control Software Level is “A”
  - New functional design using digital signature technology
  - Security and Safety concerns
  - Functional importance of the Processor Module in relation to the IMA unit is considered to be Level A
- Load Control Software Estimates
  - Requirements specification(~35 pages)
  - Design specification (~100 pages)
  - Source code is ~5,000 non-commented source lines of code (ncsloc) of C++ and around 17 function points
  - V&V plan and results (combined, ~200 pages)
  - Plan for Software Aspects of Certification/Qualification (~40 pages)
  - Software Accomplishment Summary (~20 pages)
  - Technical Order for Field User Guidance (~40 pages)

Determine FAA Involvement

- Level of FAA Involvement (LOFI) - MEDIUM
  - Step 1: Software is assessed at Level A
    - Level Of FAA Involvement (LOFI) is High or Medium
  - Step 2: Complete Other Relevant Criteria
    - Total Score Result (TSR) = 141, so LOFI is tentatively set at Medium
  - Step 3: Any Exceptions?
    - Software project has no issues that require new FAA policy, so LOFI is set at Medium
- Strategy Notes
  - For Plan for Software Aspects of Certification (PSAC), include software reliability planning information matrix in general format for software reliability plan - reference more detailed mapping of DO178B appendices to the Plan
  - For V&V Plan, use software reliability matrix evidence (reference details elsewhere) and general format for software reliability case - reference more detailed mapping of DO178B appendices to the Case
  - For Software Accomplishments Summary (SAS), use summary per NSIA Air acquisition contract so as to satisfy both the FAA Certification and NSIA Air - the ultimate user

NSIA Air Acquisition Approach

- Acquisition Contract
  - Graded Formality Based on LOFI
  - Certification Requirements, with caveats to include software reliability evidence as part of V&V evidence
  - Key Performance Parameters Evidence - focus on software reliability assessment
  - Supplier Survey Expectations - complete customer verification of all evidence
  - Schedule, cost profiles based on estimated 5,000 ncsloc C++
    - 26 person months, ~$300K, 18 months

Plan Elements

- Life Cycle Activities
  - System: FMEA, FTA, Reliability Allocation
  - Software Reliability allocation = 0.90 per execution hour, which corresponds to approximately 2500 flight hours between operational software updates (which would involve execution of the Load Control software for approximately 15 seconds)
- Plan Matrix
  - Columns:
    - Life cycle activity, Reliability Activity, Claim, Evidence, Rationale, References to Details
  - Rows:
    - Requirements, Design, Code, Unit test, System/integration test, FAA Certification, NSIA Air Review Activities

Case Elements

Specific Details of Life Cycle Analysis

- Inspection Data (defects, person hours, source of defects, estimate of defect removal efficiency)
- Testing Data (unit coverage, integration/system defect/operational time)
- Software Reliability KPP
  - defect removal efficiency estimate (req,des,code,other)
  - predicted defect delivery (based on Neufelder)
  - estimated defect delivery (based on integration/system testing data) - Motorola demo
  - estimated operational failure rate (based on reliability growth model-Motorola)
- FAA Certification Involvement
  - Reviews
  - Analysis
  - Certification Approval

Example Inspection Data

Review/ Test Activity	Size	Total # Defects	Major Defects	Total Person Hours	Major Defect Density	#Major Source SRS	#Major Source SDD	#Major Source Code	#Major Source SVP
SRS	30 pages	75	12	30	0.16	12	0	0	0
SDD	90 pages	123	45	40	0.37	4	41	0	0
Code	4500 ncsloc	158	72	112	16.0	4	6	62	0
SVP	190 pages	122	60	80	0.31	1	7	15	37
Totals	N/A	364	59	262	N/A	21	54	77	37
DRE Known Prior to Test						0.57	0.76	0.81	1.00

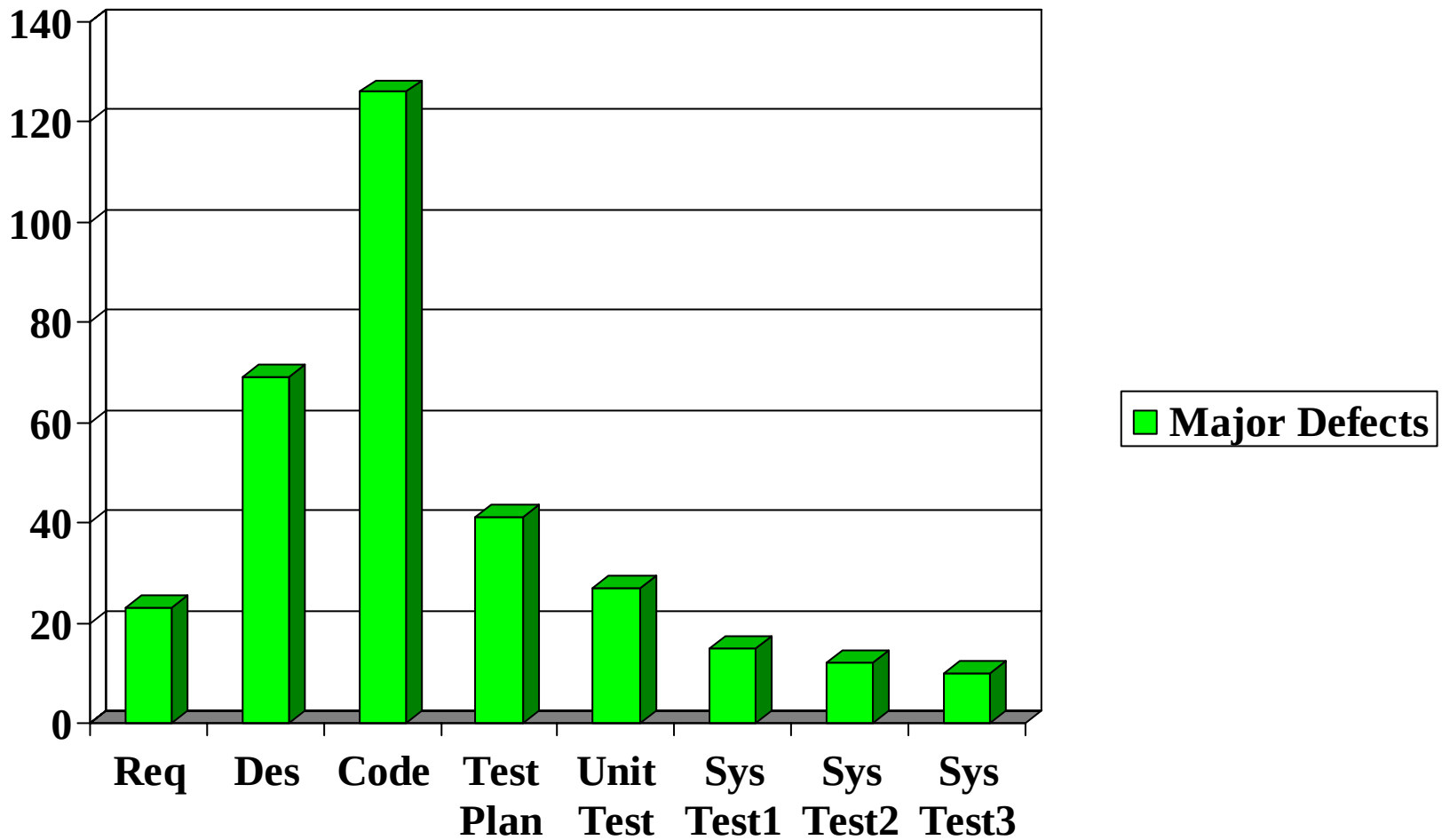
Example Inspection Data with Results of Testing

Review/ Test Activity	Size	Total # Defects	Major Defects	Total Person Hours	Major Defect Density	#Major Source SRS	#Major Source SDD	#Major Source Code	#Major Source SVP	#Major Source Unit Test	#Major Source Sys Test
SRS	30 pages	75	12	30	0.16	12	0	0	0	0	0
SDD	90 pages	123	45	40	0.37	4	41	0	0	0	0
Code	4500 ncsloc	158	72	112	16.0	4	6	62	0	0	0
SVP	190 pages	122	60	80	0.31	1	7	15	37	0	0
Totals	N/A	364	59	262	N/A	21	54	77	37	0	0
DRE Known Prior to Test						0.57	0.76	0.81	1.00	0	0
Unit Test	4500 ncsloc	78	27	250	5.55	1	4	18	2	2	0
SysTest	4500 ncsloc	84	57	25.0	12.67	1	11	31	2	6	2
DRE Known After Test						0.52	0.59	0.55	0.95	0.25	1.00

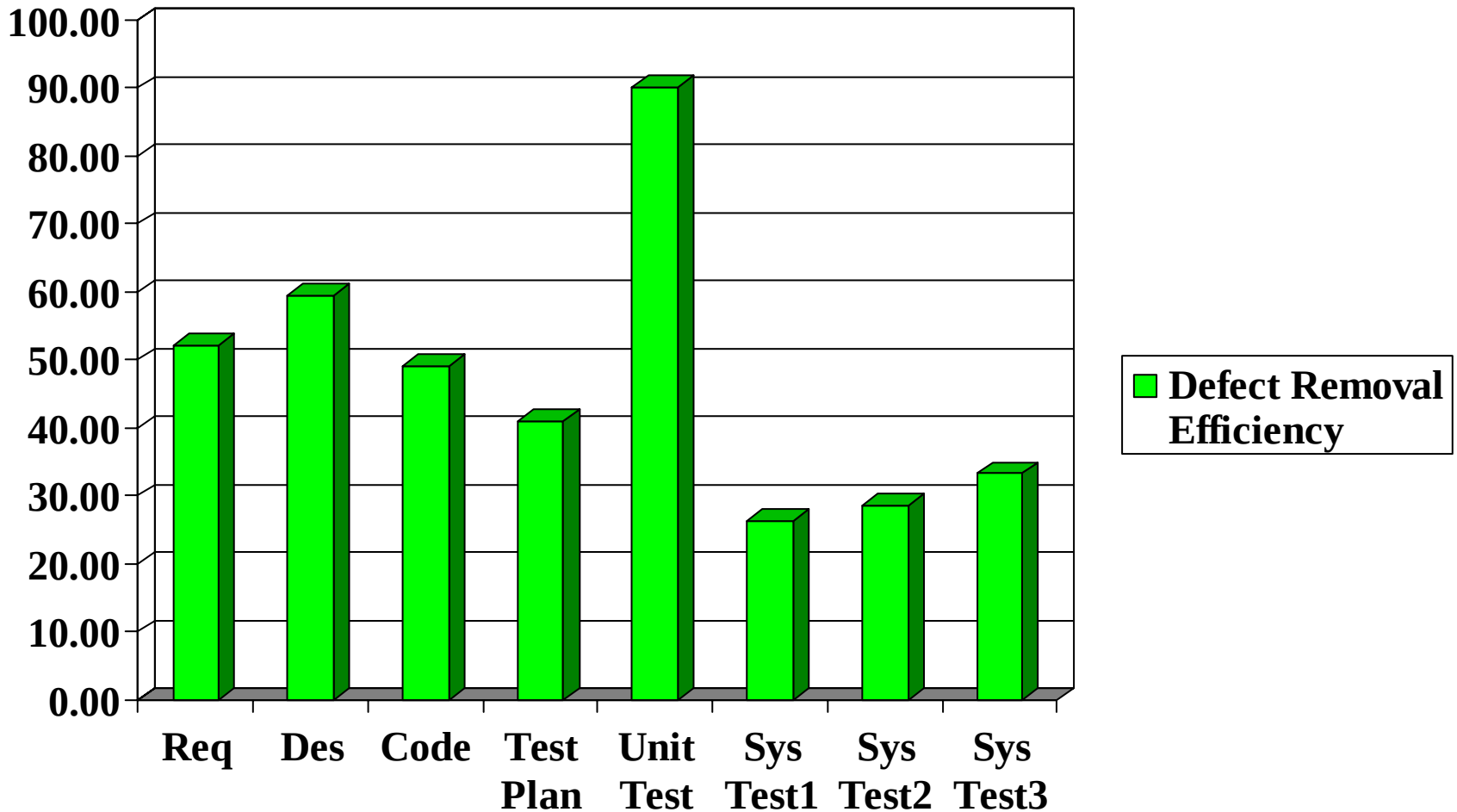
Example System/Integration Test Data

Test Phase	Failures (Total)	Failures (Major)	Execution Test Time	MDefects Source SRS	MDefects Source SDD	MDefects Source Code	MDefects Source SVP	MDefects Source UnitTest	MDefects Source SysTest
SysTest1	25	15	0.5	2	2	8	1	1	1
Sys Test2	15	12	0.5	1	1	6	1	2	1
Sys Test3	11	10	1.0	0	2	7	0	1	0
Sys Test4	10	8	2.0	1	1	4	0	2	0
Sys Test5	7	3	1.0	0	1	2	0	0	0
Sys Test6	10	5	3.0	0	1	4	0	0	0
Sys Test7	4	3	5.0	0	0	3	0	0	0
Sys Test8	1	1	7.0	0	0	1	0	0	0
Sys Test9	1	0	5.0	0	0	0	0	0	0
Totals	84	57	25.0	4	8	35	2	6	2

Plot of Defect Data by Activity



Example Defect Removal Efficiency



Note: this “efficiency” is just in removing potential defects in the artifact being reviewed or tested

Example Reliability Calculations

- Predicted Delivery Defect Density [Neufelder]
  - Process Score = X = sum of scores on 125 parameters (~1400 to 2800 range)
  - Predicted Delivered Defect Density per KNCSLOC(assembler) = D_a
  - $D_a = 0.00000017*X^2 – 0.00100439*X + 1.58463875$
  - $D_L = a* D_a$ = predicted delivered defect density in KNSLOC for language "L"; a = ~ 7 for C++
  - Assuming for IMA, the process score X = 2330 (LOFI based) we compute:
    - $D_a = 0.16732$ $D_{C++} = 7* 0.16732 = 1.1713$
- Converting to Reliability
  - N_0 = Inherent # delivered defects = $D_{C++}*4.5 = ~5.27$
  - Q = Ratio between N_0 and failures per time based on historical data/testing data (~0.254 for IMA testing) = ~ 20.75
  - $\lambda(t) = N_0 * \exp(-Q*t/ N_0) / t = 5.27 * \exp (-20.75t/5.27)/t$
  - $R(t) = \exp (- \lambda t)$
  - For t = 1 cpu hr, $\lambda(t) = 5.27 * \exp (-20.75/5.27) = ~0.103$
  - $R = \exp (-0.103) = ~0.902$ – which would meet the reliability goal of 0.90

Neufelder, "The Naked Truth About Software Engineering in the Semiconductor Equipment Industry-Revision 3," 2002 published by SoftRel

Example Reliability Calculations

- Predicted Delivery Failure Rate [Musa]
  - Using the Test Data a few slides back

Test Phase	Failures (Total)	Failures (Major)	Execution Test Time
SysTest1	25	15	0.5
Sys Test2	15	12	0.5
Sys Test3	11	10	1.0
Sys Test4	10	8	2.0
Sys Test5	7	3	1.0
Sys Test6	10	5	3.0
Sys Test7	4	3	5.0
Sys Test8	1	1	7.0
Sys Test9	1	0	5.0
Totals	84	57	25.0

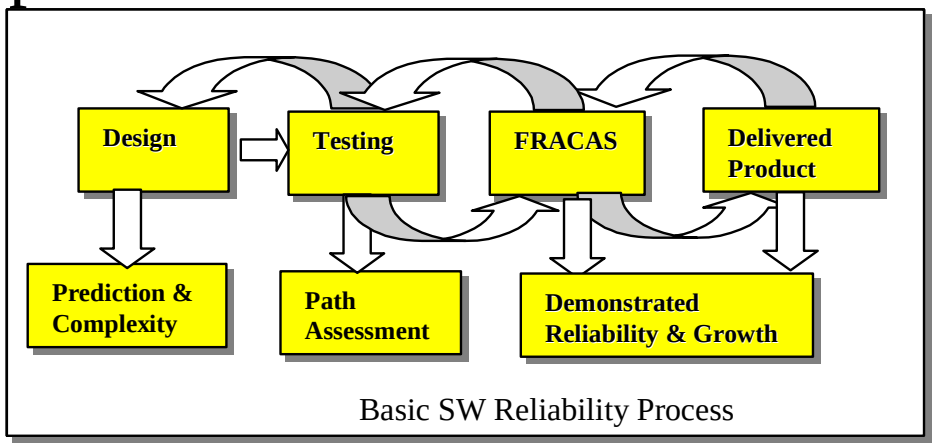
  - Need to have two estimated values
    - Initial failure rate: λ_0 (use initial test data: 15/0.5 = 27 per cpu hour
    - Total expected failures (System test + Operational Use: ν_0 ~60)

Example Reliability Calculations

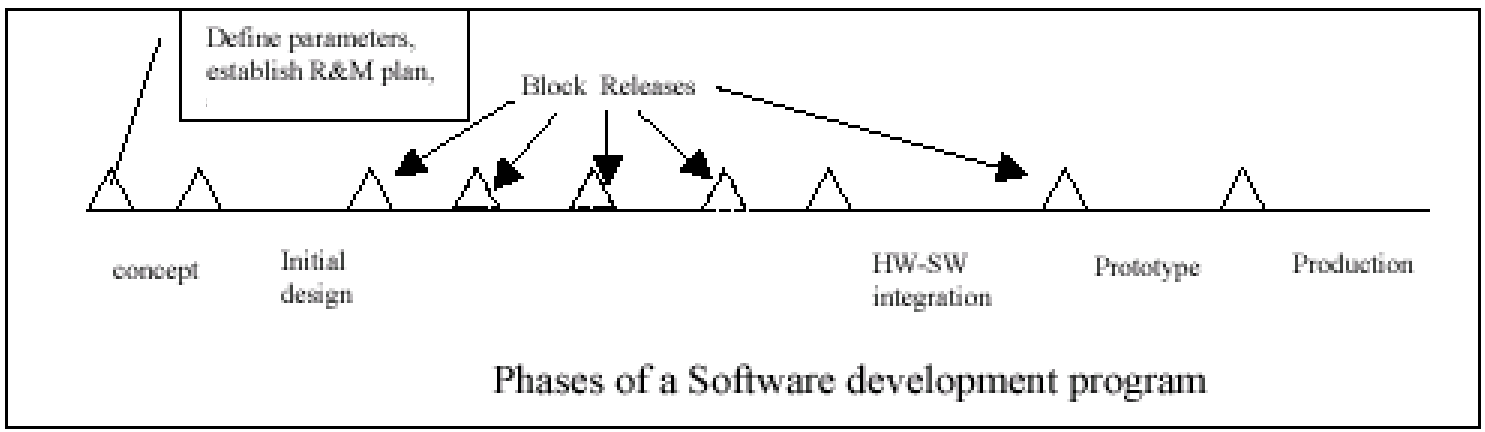
- Predicted Delivery Failure Rate [Musa]
  - λ_0 = 30 per cpu hour; ν_0 = 60; t = 25 cpu hr
  - $\lambda(t) = \lambda_0 * \exp [- (\lambda_0 / \nu_0)*25] = 30*\exp[-(30/60)*25] = 0.000112$
  - For t= 1 hr, $R = \exp (- \lambda t) = \exp (-0.000112) = 0.999888$
- Simple Motorola Model
  - Build data set to match test data
  - Check out 1/MTTF (delivered failure rate)
  - Check out graphs
  - Check out correlation/confidence
  - Comparison with Musa Model results

FRACAS and Support Data Collection

- Development Transitions to In-Service Support



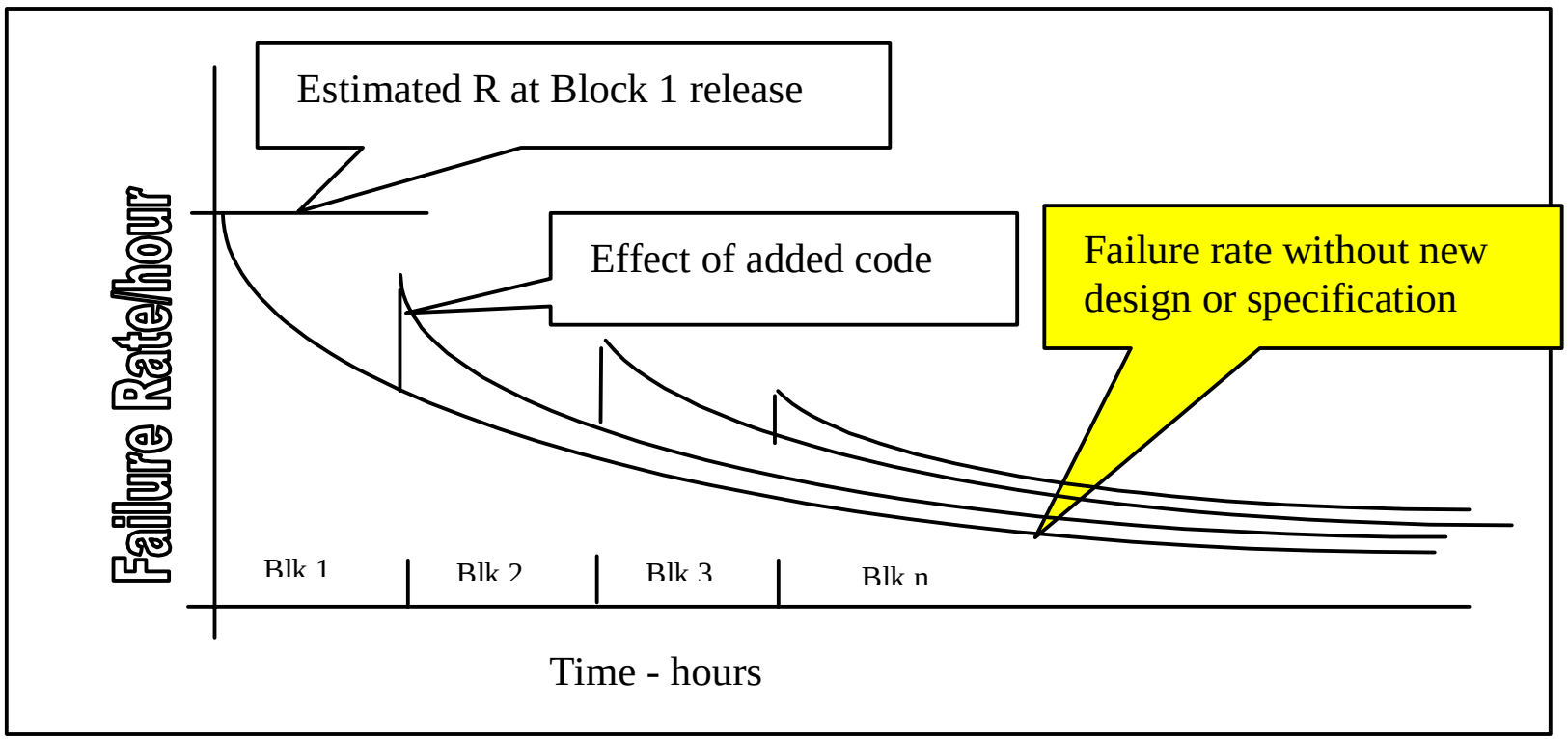
Basic SW Reliability Process



Phases of a Software development program

FRACAS and Support Data Collection

- FRACAS Data Collection Supports a Sustained Reliability Program During In-Service
- Establishes a Service History Record



FAA and NSIA Air Reviews

- Initial Review
  - PSAC Plan Matrix
  - NSIA Air negotiation for reliability planning information
  - NSIA Air supplier survey for “Top 10 practices”
    - Information from ASE assessment of software level and other criteria acceptable
- Verification Reviews
  - Inspection Data
  - Testing Data
  - Reliability Data
  - NSIA Air check of KPP
- Final Certification Review
  - SAS
  - Certification Approval

Case Study Summary

- Customer
  - Select Acquisition Approach based on Graded Formality
  - Determine Formality based on FAA LOFI Guidance
  - Piggyback on Certification Evidence
  - Consider breadth of life cycle data as software reliability evidence
- Supplier
  - Negotiate with customer(s) concerning requirements
    - Determine, meet, demonstrate
  - Use framework of Certification Plan (PSAC) and Case (SVP, Verification Test) augmented by Software Reliability Plan/Case evidence as appropriate
- Certification Authority
  - May be Customer, May be FAA
  - DO-178B is primarily a qualitative process assessment, should augment evidence with additional safety and reliability evidence for critical applications
  - FAA Guidelines provide steps, but tailoring is recommended

Discussion Q&A

- The Floor is Open! Thoughts, Questions?
  - How can illustrated case study approaches be used in your applications?
  - Which methods seem to be applicable to your work?
  - What are the Cost implications?

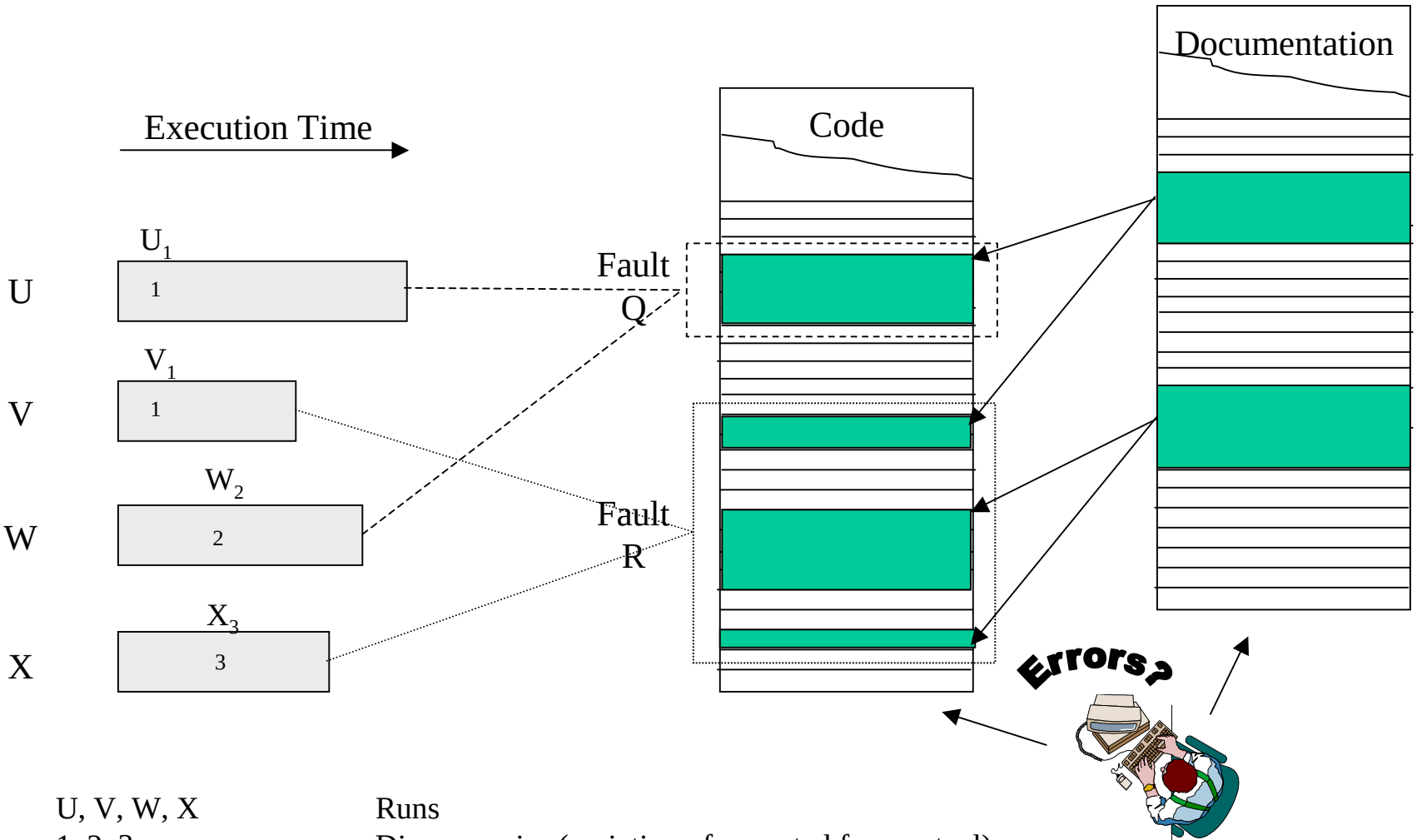
Backup Slides

- Terminology (Reliability, Failure/Fault/Error)
- Software Failure Categories (RTCA DO178)
- System HW/SW Reliability Program Relationship
- Design for Reliability Focus
- Defects: across life cycle, release defect density; tracking defects, delivered defects
- SEI CMM Level, Example Methods, FMECA/FTA/FRACAS
- Operational Reliability Focus, Failure Intensity, Reliability Growth Model
- System/software reliability integration, HW/SW Reliability (AND, OR), Simple Exercise
- Software Reliability Model (Musa), Equations, Simple Example
- References

Terminology

- Operational Profile
  - the set of functions a computer program is required to perform - further broken down by input data when it affects execution - along with the probabilities of occurrence
- Failure Intensity:
  - the number of failures occurring in a given time period (e.g., 1000 CPU hours, 1000 calendar hours)
- Fault Density:
  - number of faults per unit; unit might be source lines of code, function points, components, etc.

Failure, Fault, Error Concept



U, V, W, X	Runs
1, 2, 3	Discrepancies (variation of expected from actual)
U_1, V_1 , W_2 , X_3	Failures (discrepancy that does not satisfy user requirement)

Software Failure Severity Categories(RTCA/DO178B)

Category	Severity	Description
1	Catastrophic	Failure conditions which would prevent continued safe flight and landing.
2	Hazardous/Severe -Major	Failure conditions which would reduce the capability of the aircraft or the ability of the crew to cope with adverse operating conditions to the extent that there would be: - large reduction in safety margins or functional capabilities - physical distress or higher workload such that the flight crew could not be relied on to perform their tasks accurately or completely - adverse effects on occupants including serious or potentially fatal injuries to a small number of those occupants
3	Major	ditto intro statement as in #2 - significant reduction in safety margins or functional capabilities - significant increase in crew workload or conditions impairing crew efficiency - discomfort to occupants, possibly including injuries
4	Minor	Failure conditions which would not significantly reduce aircraft safety, and which would involve crew actions that are well within their capabilities. "Slight" rather than "significant"
5	No effect	Failure conditions which do not affect the operational capability of the aircraft or increase crew workload.

Top Ten Practices Correlated to Defect Density

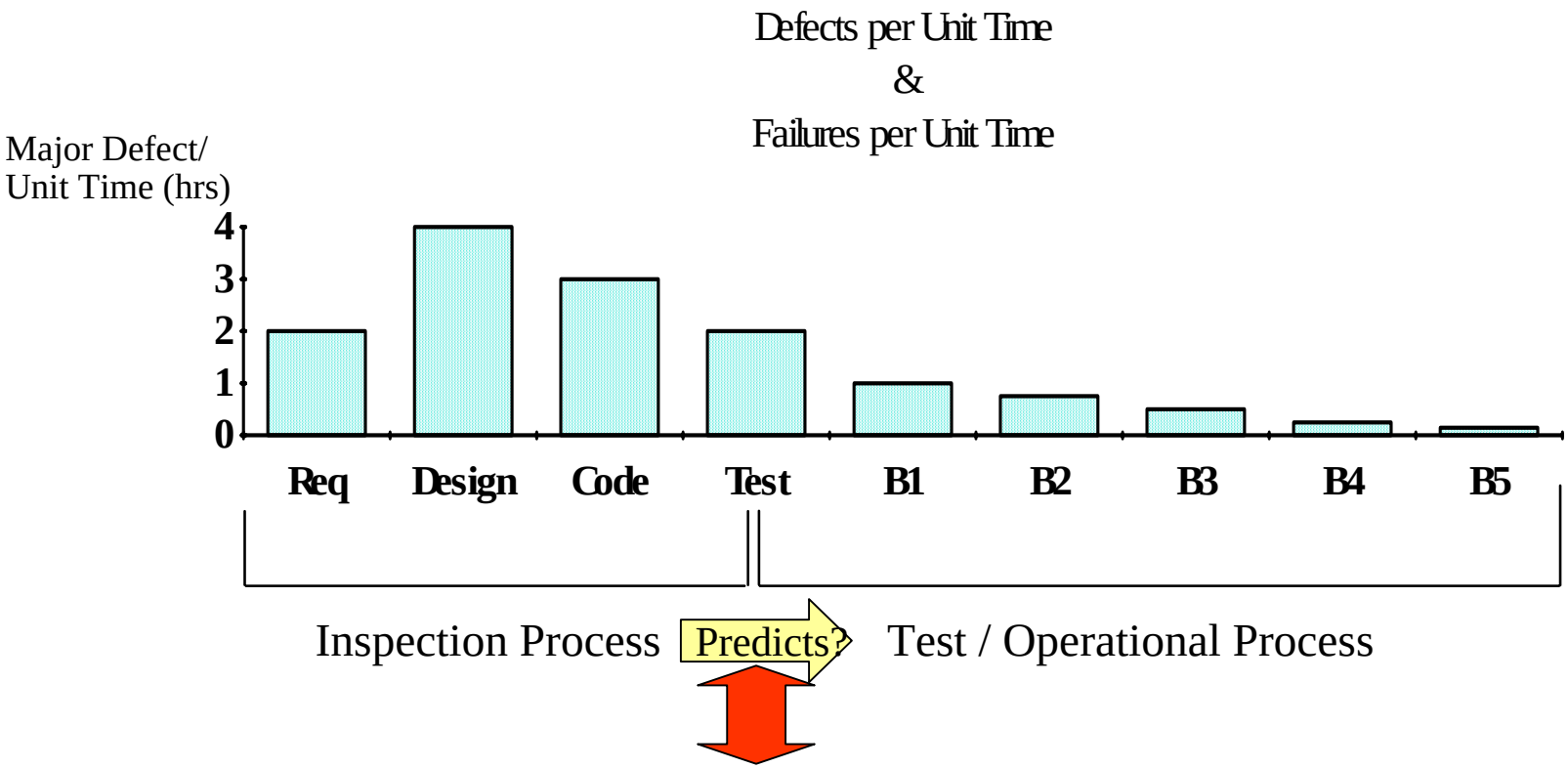
Neufelder has developed a predictive model with 125 development practices correlated to lower delivered defects. [NEUFELDER]

Practice	Correlation to Defect Density	Phase of SW Life Cycle
All requirements are mapped to system tests	-0.891739	Requirements
Requirements are reviewed before designing or coding	-0.851721	Requirements
System test beds are used	-0.847479	Testing
Test plan started at least one phase of the life cycle before testing begins	-0.823243	All up to testing
Testers use a FRACAS (defect tracking system) to determine what to test/retest	-0.806090	Testing
All upgrades made after a system test are regression tested	-0.782629	Maintenance
Corrective action releases per year <= 4	-0.777758	Maintenance
All modifications made after a system test are regression tested	-0.773434	Testing
FRACAS used for tracking all corrective actions	-0.746663	Maintenance
Walk-thrus are performed for all phases of life cycle	-0.743187	All phases

Design for Reliability Focus

- Improve Software Design Characteristics
  - the set of attributes that bear on the capability of software to maintain its level of performance under stated conditions for a stated period of time
- Implement System and Software Engineering Practices
  - life cycle activity defect reduction and prevention
    - requirements, design, implementation, test, operation, support
  - process improvement to reduce likelihood of product defects
    - SEI SW-CMM, Software Capability Maturity Model
    - ISO SPICE, Software Process Improvement Capability Determination
    - Practice standards and guidelines: Software Reliability Plan and Case
    - Software Operational Reliability Engineering Program
- Track Design Measurement Defect Data
  - inspection defect data
  - defect density
  - percentage defect removal efficiency for each life cycle phase

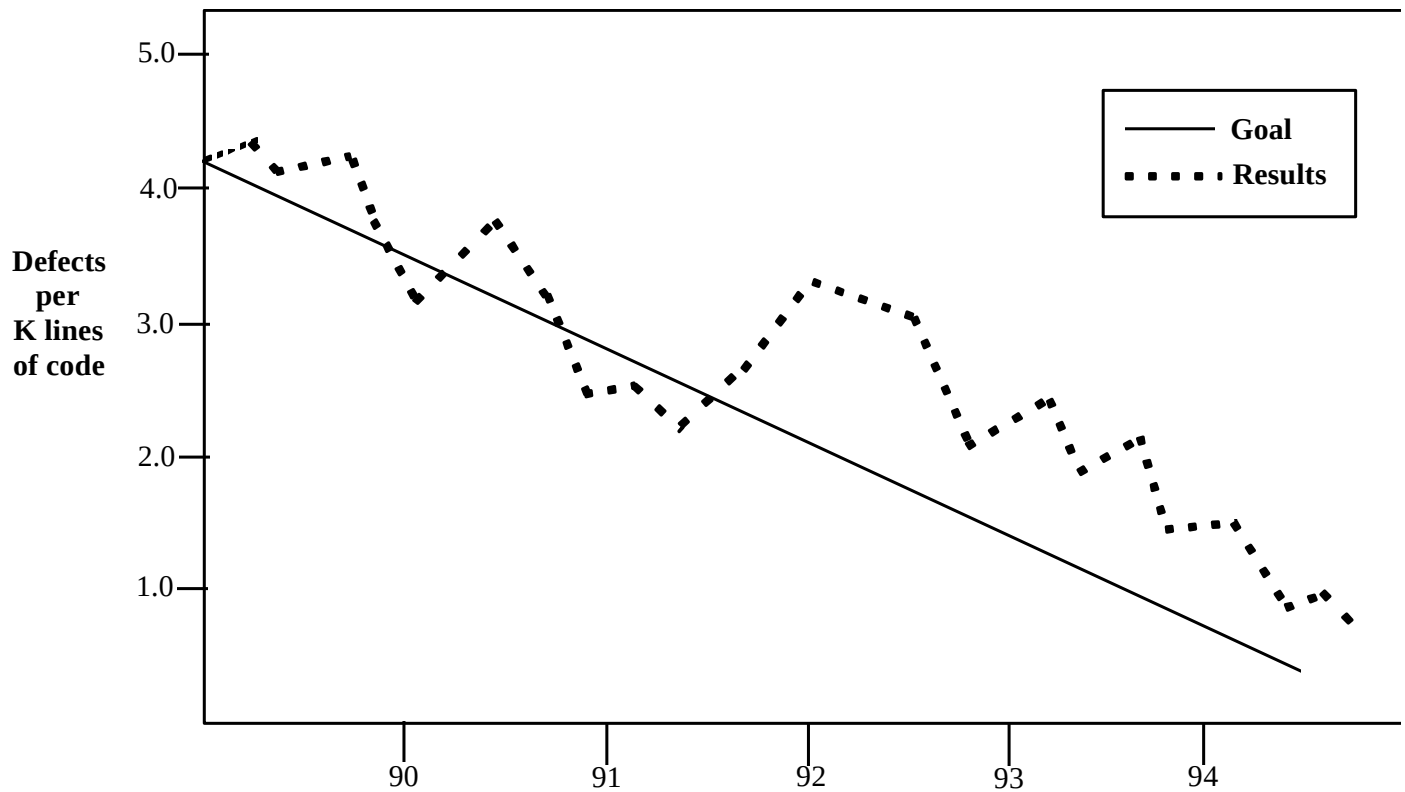
Defects & Failures per Unit Time Across the Life Cycle



The biggest challenge of software reliability today is to adequately correlate design for reliability parameters with operational reliability in a predictive manner.

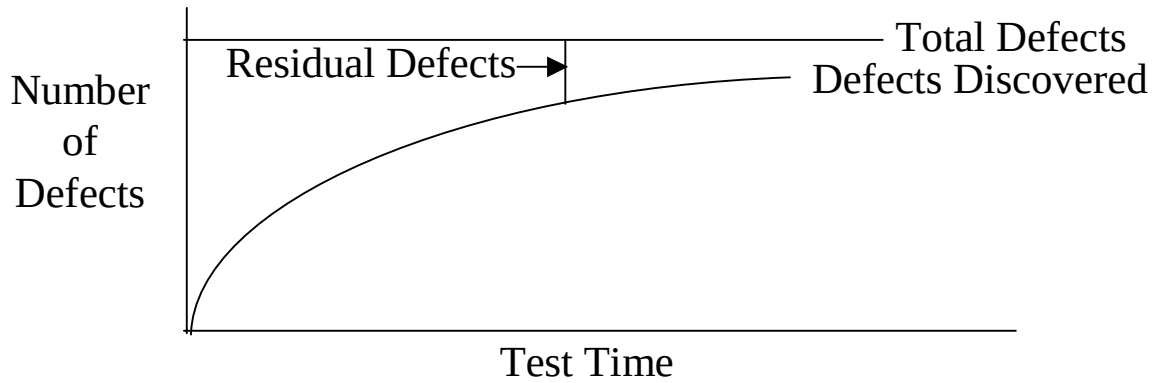
Defect Density by Release

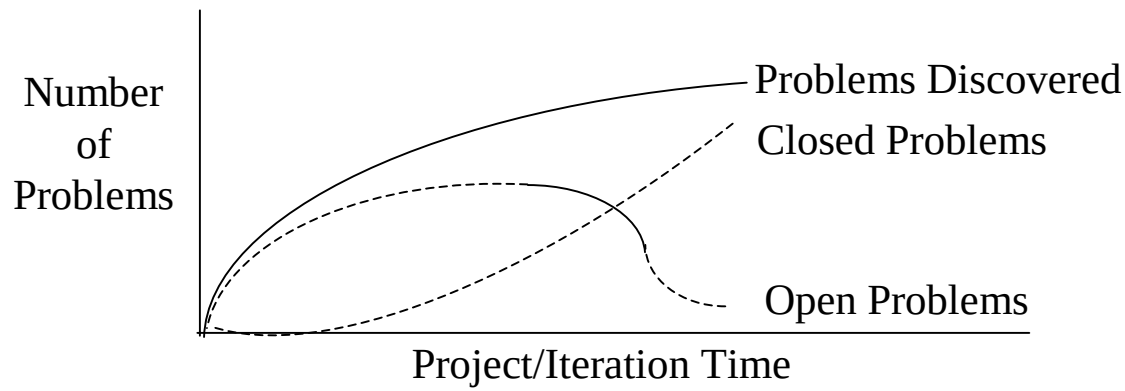
- Defects Over Time - Defect Tracking
  - Example: Tracking Defect Density by Release over Time



Tracking Defects and Problems

- Defects Over Time - Defect Tracking
  - Example: Cumulative Defects





Delivered Defects

Defects Over Time - Defect Tracking

- Example: Tracking Delivered Defects by Maturity Level [JONES]

SEL CMM Level	RTCA DO178B Level	Injected Defects/KSLOC	Delivered Defects/KSLOC
5	A	7.8	0.39
4	B	15.6	1.09
3	C	31.2	2.1
2	D	62.4	3.4
1	E	124.8	5.8

- Example: Defect Removal Efficiency by Life Cycle Activity

Defect Removal Activity	Defect Removal Efficiency
Informal design reviews	25% to 40%
Formal design inspections	45% to 65%
Informal code reviews	20% to 35%
Formal code inspections	45% to 70%
Unit test	15% to 50%
New function test	20% to 35%
Regression test	15% to 30%
Integration test	25% to 40%
System test	25% to 55%
Low-volume Beta test (< 10 clients)	25% to 40%
High-volume Beta test (> 1000 clients)	60% to 85%

All Phases Total:
Military Average: 0.96
System Software: 0.94
Commercial Software: 0.90
MIS: 0.73

SEI CMM Level
Defect & Effort Data Relationships

Maturity Level	Calendar Months	Effort in Person Months	Defects Found	Defects Shipped	Total $ Median Case
I	29.8	593.5	1348	61	$5,440K
II	18.5	143.0	328	12	$1,311K
III	15.2	79.5	182	7	$ 728K
IV	12.5	42.8	97	5	$ 392K
V	9.0	16.0	37	1	$ 146K

Krasner Consulting, 1991, typical 200 kncsloc project

Message: *the higher the maturity level, the more cost effective and reliable the delivered software product.*

NOTE: RTCA DO178B Levels of A, B, C, D, E correspond (for a project) roughly to the SEI maturity levels V, IV, III, II, I

Example Methods & Techniques*

I. Analysis Techniques
change impact analysis
common cause failure analysis
formal scenario analysis
FRACAS
Petri nets
reliability block diagrams
reliability estimation modeling
response time, memory, constraint analysis
FMECA
FTA
sneak circuit analysis

II. Design Techniques
block recovery
degraded mode operations
defensive programming
diversity
error detection/correction
fault tolerant design
information hiding
reliability allocation
design by contract

III. Verification Techniques
boundary value analysis
cleanroom
equivalence class partitioning
formal code inspections (Fagan)
functional testing
interface testing
peer reviews
performance testing
probabilistic testing
regression testing
reliability growth testing
root cause analysis
stress testing
testability analysis, fault injection, failure assertion
usability testing

* - JA1003, “Software Reliability Program Implementation Guide,” SAE G-11SW Draft, July 2002.

Software FMECA/FTA/FRACAS

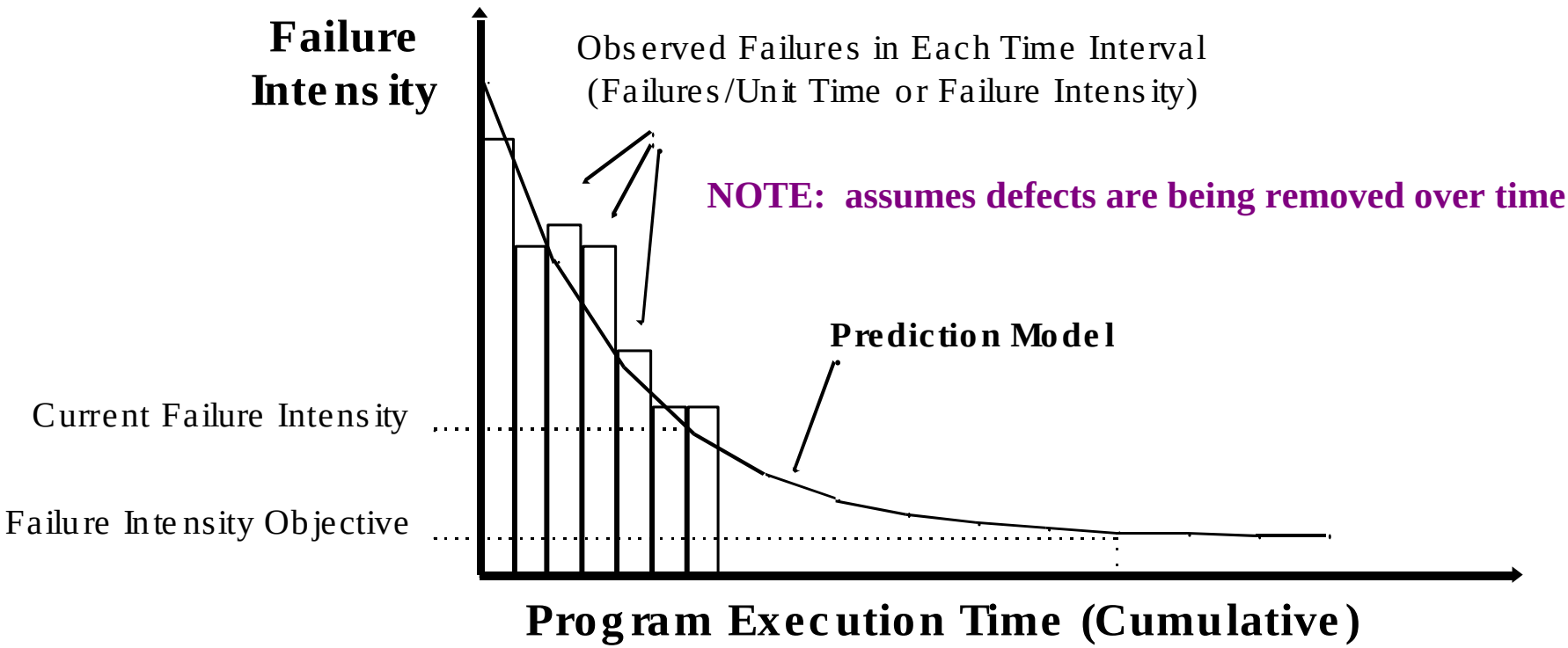
- FMECA
  - *Proactive approach* used for determining the *potential failure modes* of a system/equipment (including software), all likely ways in which a component or equipment can fail, causes for each failure mode, and effects/criticality of each failure mode.
- FTA
  - An extension of the FMECA activity in that the identified potential system failure modes are analyzed in terms of what potential faults (single point of failure) or multiple faults (multiple points of failure) might result in the potential system failure mode
- FRACAS
  - *Reactive* approach used for tracking failures and determining the *real failure modes* of the system/equipment through *root cause analysis*.

Operational Reliability Focus

- Improve Software Operational Performance
  - the probability that software will not cause the failure of a system for a specified time under specified environment conditions
- Implement Software Operational Reliability Engineering Program
  - Tailor program to application domain and organization
  - Reliability-based test architecture: operational profiles
  - Reliability-based measurement: defect collection, analysis during test, operation, and support; failure rates; confidence limits
  - Reliability-base risk decision system: when to ship, risks
- Apply Software Reliability Methods
  - System reliability allocation and prediction
  - Software reliability estimation (data) and prediction (models)

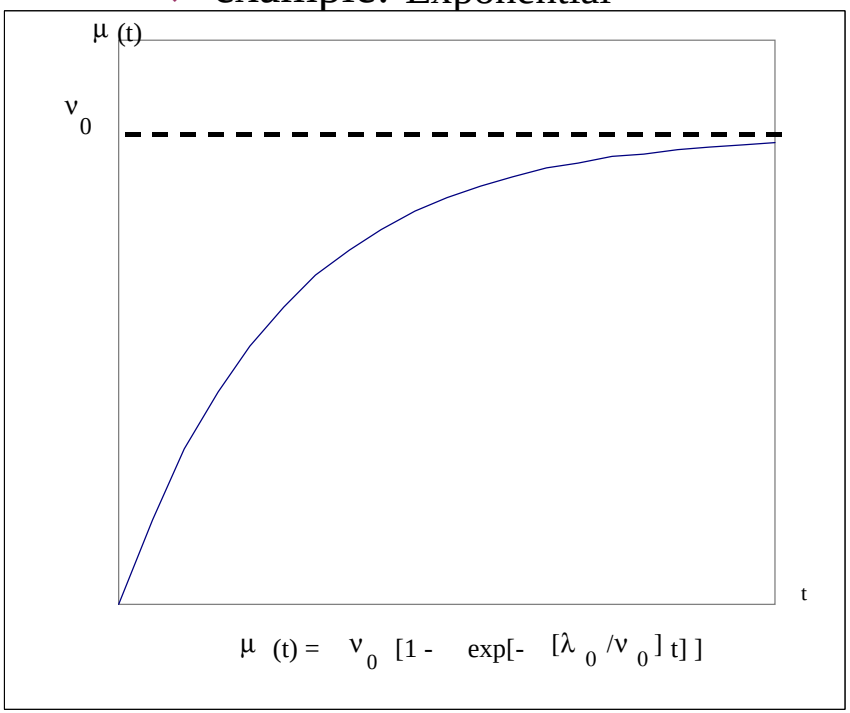
Failure Intensity

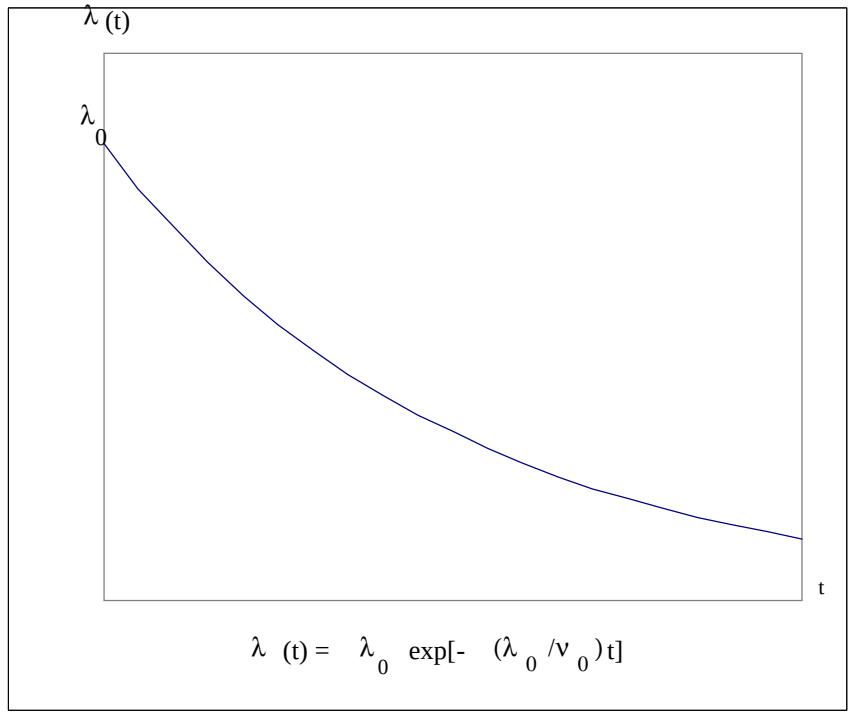
- Failure Intensity
  - The number of failures occurring in a given time period
  - Example: 1 failure per 1000 operational hours



Reliability Growth Model

- Reliability Growth Model
  - the probability of failure-free operation in a specified environment for a specified period of time [MUSA]
  - example: Exponential





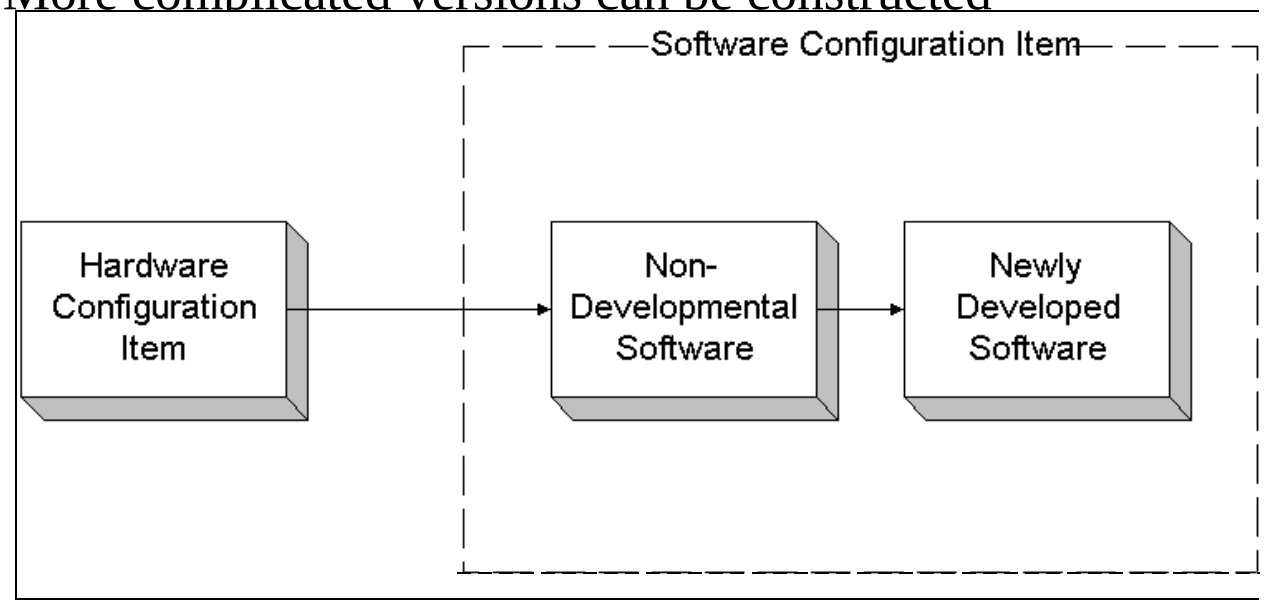
Reliability = $e^{-\lambda t}$

System/Software Reliability
Integrating SW and HW

- Step 1: Block Diagram
  - Divide system into block components for reliability analysis
- Step 2: Allocation
  - Allocate system reliability objectives to components by operational scenarios
  - Allocate to HW/SW components using parallel/series models
- Step 3: Predication
  - Conduct trade-offs with HW/SW to determine best approach to meeting reliability allocations
  - Select/develop HW/SW components to satisfy reliability allocations; use fitted data models for estimation and reliability growth models for prediction

Combining HW/SW Components to Compute Reliability

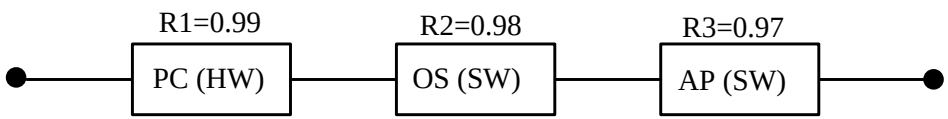
- Computational Component
  - Component with both hardware and software parts of which the software part may be composed of non-developmental and newly developed parts
  - Component parts are considered to be serial elements for reliability computation purposes; if any part fails, the component fails
  - More complicated versions can be constructed



Combining HW/SW Components to Compute Reliability: "AND" "OR"

- "AND" Configuration
  - "AND" part functions only when ALL components are functioning: SERIAL
  - R = R1*R2*R3*…*Rk
  - Example

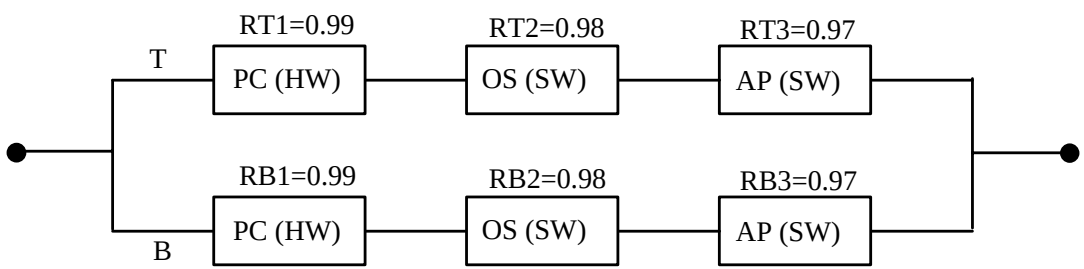
"AND" Configuration



R = R1*R2*R3 = 0.99*0.98*0.97 = 0.94

- "OR" Configuration
  - "OR" part functions when any of the components are functioning
  - R = 1-F = 1-F1*F2*F3*…*Fk
    = 1 - (1-R1)*(1-R2)…(1-Rk)
  - Example

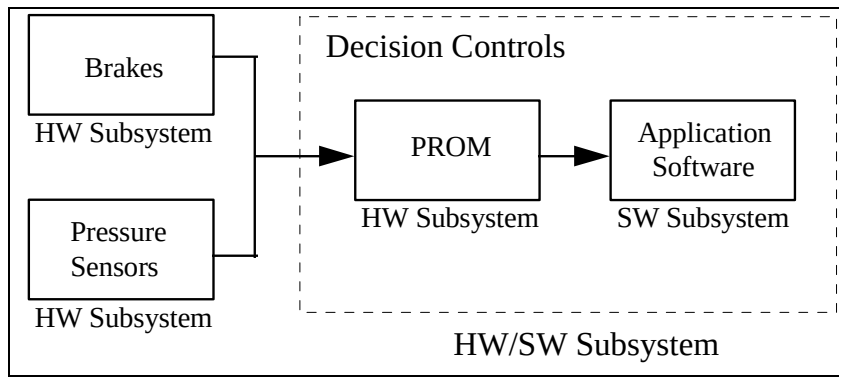
"OR" Configuration



R = [1- FT*FB] = [1 - (1-RT)(1-RB)] =[1-(1-.94)(1-.94)] = [1-0.06*0.06] = [1-0.0036] = 0.996

Simple Example/Exercise

- Given Block Diagram for Automobile Anti-lock Braking System (ABS)
- Suppose the following reliability values are known;
  - R (brakes) = R_b = 0.94
  - R (pressure sensors) = R_s = 0.96
  - R (PROM) = R_p = 0.91
  - R (application software) = R_a = 0.90
- Compute System R



$$R = [1 - (1-R_b)(1-R_s)] * R_p * R_a = [1-(1-0.94)(1-0.96)] * 0.91 * 0.90 = [0.9976]*0.91*0.90 = 0.8170344$$

Software Reliability Models

- History
  - 1970s: Jelinski-Morana, Shooman, Schick& Wolverton models
  - 1973: John Musa, Bell Laboratories, began his work
  - 1980s: Glory years of model research: Littlewood, Goel, etc
  - 1990s: The hard part - trying to apply models & get results
- Illustrate Musa's Models
  - Musa: Non-Homogeneous Poisson Process Exponential
    - Effective for System Test use
  - Musa: Non-Homogeneous Poisson Process Logarithmic
    - Effective for Field Operational use
- Modeling Cost [MUSA99]
  - 0.1-0.2% of project cost
  - Includes ALL SRE activity, not just Supportability-specific uses
  - Includes training, data collection, analysis

Model Types and Equations

	Exponential	Logarithmic
Failures Experienced (Expected)	$\mu(\tau) = \nu_0 (1 - e^{\frac{-\lambda_0}{\nu_0}\tau})$	$\mu(\tau) = \frac{1}{\theta} \ln(\lambda_0 \theta \tau + 1)$
Present Failure Intensity (Function of Failures)	$\lambda(\mu) = \lambda_0 (1 - \frac{\mu}{\nu_0})$	$\lambda(\mu) = \lambda_0 e^{-\theta\mu}$
Present Failure Intensity (Function of Time)	$\lambda(\tau) = \lambda_0 e^{\frac{-\lambda_0}{\nu_0}\tau}$	$\lambda(\tau) = \frac{\lambda_0}{\lambda_0 \theta \tau + 1}$
Additional Time to Failure Intensity Objective	$\Delta\tau = \frac{\nu_0}{\lambda_0} \ln(\frac{\lambda_P}{\lambda_F})$	$\Delta\tau = \frac{1}{\theta}(\frac{1}{\lambda_F} - \frac{1}{\lambda_P})$
Additional Failures to Intensity Objective	$\Delta\mu = \frac{\nu_0}{\lambda_0}(\lambda_P - \lambda_F)$	$\Delta\mu = \frac{1}{\theta}\lambda\nu(\frac{\lambda_P}{\lambda_F})$
Reliability (No Failures Corrected)	$R(\tau) = e^{-\lambda\tau}$	

Basic Formula Terms

μ	=	failures experienced (expected)
$\Delta\mu$	=	additional failures (expected)
τ	=	execution time
$\Delta\tau$	=	additional execution time
λ_0	=	initial failure intensity - (must be estimated)
λ_P	=	present failure intensity
λ_F	=	failure intensity objective
ν_0	=	total failures (expected) - (must be estimated)
θ	=	failure intensity reduction (decay) rate - (must be estimated, nominal values = .02/failure - .05/failure)

Computational Exercise

Given that $\nu_0 = 500$ $\lambda_0 = 27/hr$ $\lambda_P = 15/hr$ $\lambda_F = 0.5/hr$ and $t = 10$

Compute (Exponential Model)

$\mu(t) = \nu_0[1 - \exp[-\lambda_0 t / \nu_0]]$ = expected number of current failures experienced at time t

$= 500[1-\exp[-27*10/500]] = 208.62587 \sim 209$ failures

$\lambda(t) = \lambda_0 [1 - \mu/\nu_0]$ = expected failure intensity/rate at the current time t

$= 27 [1 - 209/500] = 15.714$ failures per hour at t=10 hours

$\Delta\mu = [\nu_0/ \lambda_0] [\lambda_P - \lambda_F]$ = expected additional failures to reach failure intensity objective

$= [500/27][15 - 0.5] = 268.51852 \sim 269$ more failures to reach objective of 0.5/hr

$\Delta t = [\nu_0/ \lambda_0] \ln [\lambda_P/\lambda_F]$ = expected additional time to reach failure intensity objective

$= [500/27] \ln[15/0.15] = 62.985137 \sim 63$ hours more testing to reach objective of 0.5/hr

References

- [DEMARCO] DeMarco, T., Controlling Software Projects, Yourdon Inc., New York, NY, 1982.
- [IEEE87] IEEE Standard, *"A Standard Classification of Software Errors, Faults, and Failures,"* Technical Committee on Software Engineering, Standard P-1044/D3, December 1987.
- [JONES] Jones, C., *"Conflict and Litigation Between Software Clients and Developers,"* Version 1 -- March 4, 1996.
- [LEUNG] Leung, H., *"Improving Defect Removal Effectiveness for Software Development,"* Department of Computing, Hong Kong Polytechnic University.
- [LYU] Lyu, M., Software Reliability Engineering, IEEE Computer Society Press, McGraw-Hill, New York, NY, 1996.
- [MUSA] Musa, J., Software Reliability Engineering, McGraw-Hill, New Your, NY, 1999.
- [NEUFELDER] Neufelder Owner, A., N., *"The Facts About Predicting Software Defects and Reliability,"* Journal of the RAC, 2ndQ, 2002, pp 1-4.
- [NEUFELDER-LAKEY] Lakey, Neufelder, "System Software Reliability Assurance Guidebook," Rome Laboratory, 1995.

Appendix A

Glossary

A. Glossary of Terms

A.1 Primary Acronyms

AIAA	American Institute of Aeronautics and Astronautics
AIR	Aerospace Information Report
ANSI	American National Standards Institute
ARMP	Allied Reliability and Maintainability Publication
ASIC	Application Specific Integrated Circuit
BSI	British Standards Institute
CMMI	Capability Maturity Model Integrated
COTS	Commercial Off-The-Shelf
DACS	Data and Analysis Center for Software
DND	Department of National Defence (Canada)
DoD	Department of Defense
DSI	Delivered Source Instructions
EIA	Electronics Industries Alliance
FAA	Federal Aviation Administration
FIR	Formal In-Process Review
FMECA	Failure Modes, Effects and Criticality Analysis
FRACAS	Failure Reporting and Corrective Action System
FTA	Fault Tree Analysis
GQM	Goal, Question, Metric
HCI	Human Computer Interface
I4	Independence, Isolation, Interoperability, Inoperability
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronic Engineers
ISO	International Organization for Standardization
IVAN	Independent Vulnerability ANalysis
JA	Two character code for SAE ground vehicle (J) and aerospace (A) standards and guidelines
KSLOC	Thousands (K) of Source Lines of Code
MISRA	Motor Industry Software Reliability Association
MOD	Ministry Of Defence (United Kingdom)
NATO	North Atlantic Treaty Organization
NCSLOC	Non-Commented Source Lines of Code
NDI	Non-Developmental Item
NIST	National Institute of Standards and Technology
OTS	Off-The-Shelf
QA	Quality Assurance
QFD	Quality Function Deployment
R&M	Reliability and Maintainability
RAC	Reliability Analysis Center
RMSL	Reliability, Maintainability, Supportability, Logistics
SAE	Society of Automotive Engineers
SEI	Software Engineering Institute

SFMECA	Software FMECA
SFTA	Software FTA
SLOC	Source Lines of Code
SRE	Software Reliability Engineering
UK	United Kingdom
V&V	Verification and Validation

A.2 Primary Definitions

The following key terms are defined. Reference [IEEE610] is a generally applicable reference for software terms not defined in this section. Some terms apply only to software, but many terms apply more generally to a system of which software is a component. The specific source of the definitions in this section is referenced as follows:

[0] term defined by its use in the guide [JA1003]
[1] reference [AIAAR013]
[2] reference [JA1002]
[3] reference [JA1000-1]
[4] reference [ISO12207]
[5] reference [MUSA99]
[6] reference [JA1005]
[7] reference [DO178B]
[8] reference [MILSTD882D]

Acquirer[4]: An organization that procures a system, software product or software service from a supplier. NOTE-The acquirer could be one of the following: buyer, owner, user, purchaser.

Certification[7]: Legal recognition by the certification authority that a product, service, organization or person complies with the requirements. Such certification comprises the activity of technically checking the product, service, organization or person and the formal recognition of compliance with the applicable requirements by issue of a certificate, license, approval or other documents as required by national laws and procedures.

Certification Authority/Regulator [7]: The organization or person responsible within the state or country concerned with the certification of compliance with the requirements.

Contract[4]: A binding agreement between two parties, especially enforceable by law, or a similar internal agreement wholly within an organization, for the supply of software service or for the supply, development, production, operation, or maintenance of a software product.

Coverage[0]: The ratio of actual to possible software features/functions, requirements, statements, and/or branches/paths that are exercised during one or more test cases. Types of coverage can be categorized by the unit, e.g., feature coverage, requirements coverage, statement coverage, path coverage.

Customer[0]: see Acquirer.

Defect[0]: Any condition in a software artifact (e.g., specification, code, test) that if left unchanged could result in a software failure. Defect and fault are sometimes considered to be synonymous although fault is more strictly considered to be a defect in the code.

Dependability[0]: See Surety.

Design Reliability[0]: (1) The set of activities that focus on the prevention, detection, prediction, estimation, and/or mitigation of defects in software specifications (e.g., user guide, requirements, design, code, test plan/cases). (2) A measure of the remaining defects in software specifications at a specific reference point. (3) A measure of the predicted software reliability at a specific reference point.

Developer[4]: An organization that performs development activities (including requirements analysis, design, testing through acceptance) during the software life cycle process.

Error[1]: (1) A discrepancy between a computed, observed or measured value or condition and the true, specified or theoretically correct value or condition. (2) Human action that results in software containing a fault.

Failure[1]: (1) The inability of a system or system component to perform a required function within specified limits. A failure may be produced when a fault is encountered and a loss of the expected service to the user results. (2) The termination of the ability of a functional unit to perform its required function. (3) A departure of program operation from program requirements.

Failure Intensity[5]: see Failure Rate.

Failure Modes, Effects and Criticality Analysis [3]: A proactive approach used for determining the potential failure modes of a system/equipment (including software), all likely ways in which a component or equipment can fail, causes for each failure mode, and effects/criticality of each failure mode.

Failure Rate[1]: (1) The ratio of the number of failures of a given category or severity to a given period of time; for example failures per second of execution time, failures per month. Synonymous with failure intensity. (2) The ratio of the number of failures to a per unit of time, failures per number of transactions, failures per number of computer runs.

Failure Reporting and Corrective Action System [3]: A set of processes, procedures, and tools for reporting, reviewing, analyzing, correcting, and storing information about system/software failures.

Failure Severity[adapted from 1]: A rating system for the impact of every recognized credible failure mode.

Fault Tolerance[1]: The survival attribute of a system that allows it to deliver the required service after faults have manifested themselves within the system.

Fault Tree Analysis [3]: An analysis technique where identified potential system failure modes are analyzed in terms of what potential software faults (single point of failure) or multiple faults (multiple points of failure) might result in the potential failure mode.

I4- Independence, Isolation, Interoperability, Inoperability[0]:

Independence - multiple, independent subsystems and completely different sources of enabling stimuli for critical functions are incorporated within the system

Isolation - critical functions are encapsulated separate from any other functions that might cause undefined interactions with the critical functions

Interoperability - critical functions become predictably and irreversibly inoperable in credible abnormal operating environments before the isolation features are compromised

Inoperability - functional interfaces are constructed so that they are incompatible with functions (in particular safety critical functions) with which they are not intended to interface

Life Cycle Model[4]: A framework containing the processes, activities, and tasks involved in the development, operation, and maintenance of a software product, spanning the life of the system from the definition of its requirements to the termination of its use.

Non-Deliverable Item[4]: Hardware or software product that is not required to be delivered under the contract but may be employed in the development of a software product.

Off-The-Shelf Product[4]: Product that is already developed and available, usable either "as is" or with modification.

Operational Profile[5]: The complete set of operations (major system logical tasks) with their probabilities of occurrence.

Operational Reliability[0]: (1) The set of dynamic test activities that focus on the prevention, detection, prediction, estimation, and/or mitigation of defects in the operational software code through dynamic unit, integration, acceptance, certification testing). (2) A measure of the remaining faults in software code at a specific reference point. (3) A measure of the estimated software reliability at a specific reference point.

Process[4]: A set of interrelated activities, which transform inputs into outputs. NOTE-The term "activities" covers use of resources.

Qualification[4]: The process of demonstrating whether an entity is capable of fulfilling specified requirements.

Quality Assurance[4]: All the planned and systematic activities implemented within the quality system, and demonstrated as needed, to provide adequate confidence that an entity will fulfill requirements for quality. NOTES – 1) There are both internal and external purposes for quality assurance: a) Internal quality assurance: within an organization, quality assurance provides confidence to management; b) External quality assurance: in contractual situations, quality assurance provides confidence to the customer or others. 2) Some quality control and quality assurance actions are interrelated. 3) Unless requirements for quality fully reflect the needs of the user, quality assurance may not provide adequate confidence.

Safety[8]: Freedom from those conditions that can cause death, injury, occupational illness, damage to or loss of equipment or property, or damage to the environment.

Security[0]: Features and procedures of a system that ensure its requirements are met for timely access to authenticated services and for protection from denial of authenticated services.

Software Failure[2]: The inability of a software component to perform its required functions within specified performance requirements.

Software Fault[1]: (1) A defect in the code that can be the cause of one or more failures. (2) An accidental condition that causes a functional unit to fail to perform its required function. Synonymous with bug.

Software Fault[2]: An accidental condition that causes a software functional unit to fail to perform its required function.

Software Fault Density[0]: The ratio of code faults to a unit of size, such as function points, modules, source lines of code at a specific reference point of time, such as at the start of system test or operational use.

Software Maintainability[6]: The ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment. Also, a set of attributes that bear on the effort needed to make specified modifications.

Software Maintenance[6]: The process of modifying a software system or component after delivery to correct faults, improve performance or other attributes, or adapt to a changed environment.[1]

Software Modification Support[6]: The software support activities of change analysis, implementation, test and release of software products. Changes may be termed corrective, perfective and adaptive, and may also embrace modifications that are designed to prevent foreseeable future software operating problems.

Software Product[1]: The set of computer programs, procedures, and possibly associated documentation and data.

Software Reliability[1]: (1) The probability that software will not cause the failure of a system for a specified time under specified conditions. The probability is a function of the inputs to and use of the system, as well as a function of the existence of faults in the software. The inputs to the system determine whether existing faults, if any, are encountered. (2) The ability of a program to perform a required function under stated conditions for a stated period of time.

Software Reliability[2]: (1) The probability of failure-free operation of a software program for a specified time under specified conditions. (2) A set of attributes that bear on the capability of software to maintain its level of performance under stated conditions for a stated period of time.

Software Reliability Case[0]: The evidence presented throughout the project that software reliability requirements are consistent with system level requirements, are achievable, are understood by the development organization, and that ambiguities have been resolved.

Software Reliability Engineering[1]: 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.

---

[1] Software maintenance as defined above is essentially the same as software modification support, but is only part of the software support activities.

Software Reliability Estimation[1]: The application of statistical techniques to observed failure data collected during system testing and operation to assess the reliability of the software.

Software Reliability Management[0]: The process of optimizing the reliability of software across the complete software life cycle by emphasizing human error prevention, fault detection and removal, use of measurements to improve reliability, and balancing the level of reliability consistent with project constraints such as resources, schedule, and performance.

Software Reliability Model[1]: A mathematical expression that specifies the general form of the software failure process as a function of factors such as fault introduction, fault removal and the operational environment.

Software Reliability Plan[0]: A description of the set of activities that will be performed throughout a project to ensure that requirements for software reliability have been defined through negotiations with the customer, analyses have been identified and conducted that ensure customer reliability requirements are met, and demonstrated evidence is provided that the customer reliability requirements have been achieved.

Software Reliability Prediction[1]: A forecast of the reliability of the software based on parameters associated with the software product and its development environment.

Software Reliability Program[0]: The management infrastructure and activities necessary to adequately integrate software reliability within a system reliability program and provide adequate evidence that the software reliability requirements have been determined, met, and demonstrated. The two key components of the management infrastructure are the Software Reliability Plan and Software Reliability Case.

Software Reliability Program [adapted from 3]: The organizational processes and practices that are intended to: (1) Ensure the delivery of a software product that has been adequately designed to achieve its performance specifications within its system application context; and (2) Ensure there is adequate evidence that the performance specification for the delivered software product has been achieved and continues to be met during operational use.

Software Safety[0]: Features and procedures which ensure that a software product performs predictably under normal and abnormal conditions, thereby minimizing the likelihood of an unplanned event occurring, controlling and containing its consequences, and preventing accidental injury, death, destruction of property and/or damage to the environment, whether intentional or unintentional.

Security[0]: Features and procedures of a system that ensure that its requirements for timely access to authenticated services and for protection from denial of authenticated services.

Supplier[4]: An organization that enters into a contract with the customer for the supply of a system, software product or software service under the terms of the contract. NOTES- 1) The term "supplier" is synonymous with contractor, producer, seller, or vendor. 2) The customer may designate a part of its organization as supplier.

Surety[0]: Attributes of and activities associated with achieving and assessing system safety, security, and reliability.

System[1]: (1) A collection of people, machines and methods organized to accomplish a set of specific functions. (2) An integrated whole that is composed of diverse, interacting, specialized structures and subfunctions. (3) A group or subsystem united by some interaction or interdependence, performing many duties but functioning as a single unit.

System Reliability[3]: The ability of a system to perform a stated function under stated conditions, for a stated period of time.

System Safety[8]: The application of engineering and management principles, criteria, and techniques to achieve acceptable mishap risk, within the constraints of operational effectiveness and suitability, time, and cost, throughout all phases of the system life cycle.

Time[1]: There are several categories of time that may be of interest for determining when failures occur and the impact of the frequency of the failures. These categories include: (1) Calendar Time: chronological time, including time during which a computer may not be running. (2) Clock Time: elapsed wall clock time

from the start of program execution to the end of program execution. (3) Execution Time: the amount of actual processor time used in executing a program.

Validation[4]: Confirmation by examination and provision of objective evidence that the particular requirements for a specific intended use are fulfilled. NOTES – 1) In design and development, validation concerns the process of examining a product to determine conformity with user needs. 2) Validation is normally performed on the final product under defined operating conditions. It may be necessary in earlier stages. 3) "Validated" is used to designate the corresponding status. 4) Multiple validations may be carried out if there are different intended uses.

Verification[4]: Confirmation by examination and provision of objective evidence that specified requirements have been fulfilled. NOTES – 1) In design and development, verification concerns the process of examining the result of a given activity to determine conformity with the stated requirement for that activity. 2) "Verified" is used to designate the corresponding status.

Appendix B

References

B. References

The indicated references and web links were the current known version as of the publication of this guide. There are numerous other publications relevant to software reliability.

B.1 SAE PUBLICATIONS

SAE publications can be obtained from:

http://www.sae.org/
SAE World Headquarters
400 Commonwealth Drive
Warrendale, PA 15096-0001 USA

[AIR5022] SAE Aerospace Information Report AIR5022, "Reliability and Safety Process Integration," Society of Automotive Engineers, July 1996.

[ARP5580] SAE Aerospace Recommended Practice ARP 5580, "Recommended Failure Modes and Effects Analysis (Fmea) Practices for Non-Automobile Applications," Society of Automotive Engineers, July 2001.

[J1739] SAE J Standard 1739, " Potential Failure Mode and Effects Analysis in Design (Design FMEA) and Potential Failure Mode and Effects Analysis in Manufacturing and Assembly Processes (Process FMEA) and Effects Analysis for Machinery (Machinery FMEA)," Society of Automotive Engineers, August 2002.

[JA1000] SAE JA Standard 1000, “Reliability Program Standard,” Society of Automotive Engineers, 1998.

[JA1000-1] SAE JA Guideline1000-1, “Reliability Program Implementation Guide,” Society of Automotive Engineers, 2000.

[JA1002] SAE Surface Vehicle/Aerospace (JA) Standard 1002, “Software Reliability Program Standard,” Society of Automotive Engineers, 1998.

[JA1003] SAE Surface Vehicle/Aerospace (JA) Guideline 1003 (Draft v0.93), “Software Reliability Program Standard Implementation Guide,” Society of Automotive Engineers, Draft, Publication Scheduled for November 2003.

[JA1004] SAE Surface Vehicle/Aerospace (JA) Standard 1004, “Software Supportability Program Standard,” Society of Automotive Engineers, 1998.

[JA1005] SAE Surface Vehicle/Aerospace (JA) Standard 1004, “Software Supportability Program Implementation Guidelines,” Society of Automotive Engineers, 2001.

[JA1006] SAE Surface Vehicle/Aerospace (JA) Standard 1004, “Software Support Concept,” Society of Automotive Engineers, 1999.

B.2 RELATED STANDARDS

A world-wide search capability for reliability standards and standards developing organizations is available from:

http://rac.iitri.org/rac/jsp/standards/standard.jsp
IIT Research Institute / Reliability Analysis Center
201 Mill Street, Rome, NY 13440-6916

AIAA documents can be obtained from:

http://www.aiaa.org/
American Institute of Aeronautics and Astronautics (AIAA)
1801 Alexander Bell Drive, Suite 500
Reston, VA 20191-4344

British Standards Institute documents can be obtained from:

http://www.bsi-global.com/index.xalter
British Standards Institute (BSI)
Linford Wood Milton Keyes
MK14 6LE UK

DoD documents can be obtained from:

http://dodssp.daps.mil/

Chief, Bibliographic Systems
U.S. Government Printing Office
Sales Management Division (SSMB)
Washington, DC 20402

IEC documents can be obtained from:
http://www.techstreet.com/info/iec.html
International Electrotechnical Commission
1327 Jones Dr.
Ann Arbor, MI, 48105 USA

IEEE documents can be obtained from:
http://www.computer.org/
IEEE Computer Society
Publications Office
10662 Los Vaqueros Circle
P. O. Box 3014
Los Alamitos, CA 90720-1264 USA

ISO documents can be obtained from:
http://www.ili-info.com/us/
Europe: ILI, Index House, Ascot, Berkshire, SL5 7EU, UK
USA: ILI, 610 Winters Avenue, Paramus, NJ 07652, USA
Germany: ILI, Dietlindenstraße 15, D-80802, Munich, Deutschland
Italy: ILI, Via Guido D'Arezzo, 4 - 20145 Milano
France: ILI, 25 rue de Ponthieu, 75008 Paris, France

MISRA documents can be obtained from:
http://www.misra.org.uk/index.htm
Motor Industry Software Reliability Association (MISRA)
Electrical Group, MIRA Ltd
Watling Street
Nuneaton, Warwickshire CV10 0TU UK

NATO documents can be obtained from:
http://www.nato.int/docu/standard.htm
Directorate of Standardization
Stan 2
Kentigern House
65 Brown Street
GLASGOW G2 8EX

NIST documents can be obtained from:
http://csrc.nist.gov/publications/nistpubs/index.html
National Institute of Standards and Technology
100 Bureau Drive, Stop 3460
Gaithersburg, MD 20899-3460 USA

RTCA documents can be obtained from:
http://www.rtca.org/
RTCA, Inc.
1828 L Street, NW
Suite 805
Washington, DC 20036 USA

Software Engineering Institute documents can be obtained from:
http://www.sei.cmu.edu/
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213-3890 USA

UK Ministry of Defence documents can be obtained from:
http://www.dstan.mod.uk/home.htm

UK Defence Standardization
Room 1138
Kentigern House
65 Brown Street
GLASGOW G2 8EX

[AIAAR013] ANSI/AIAA R-013-1992, “AIAA Recommended Practice for Software Reliability,” February 1993.

[ARMP1] ARMP-1, Edition 3, "NATO Requirements for Reliability and Maintainability," June 2002.

[ARMP4] ARMP-4, Edition 2, "Guidance on Writing NATO R&M Requirements Documents," October 2001.

[ARMP6] ARMP-6, Edition ?, "Monitoring and Managing In-Service R&M,"

[ARMP7] NATO R&M Terminology in ARMPs

[BS5760-P8] BS 5760, “Reliability of Systems, Equipment and Components,” Part 8: “Guide to Assessment of Reliability of Systems Containing Software,” British Standards Institute, Draft for Approval for Publication, July 7, 1997.

[CMMI2000] CMMI-SE/SW-Continuous, V1.02, CMMI for Systems Engineering/Software Engineering, Version 1.02, Continuous Representation, CMU/SEI-2000-TR-019, November 2000.

CMMI-SE/SW-Staged, V1.02, CMMI for Systems Engineering/Software Engineering, Version 1.02, Staged Representation, CMU/SEI-2000-TR-018, November 2000.

[DEFSTAN0042] Defence Standard 00-42 (PART 2)/Issue 1, “Reliability And Maintainability Assurance Guides, Part 2: Software,” United Kingdom Ministry of Defence, September 1997.

[DEFSTAN0055] Defence Standard 00-55 Issue 2, “Requirements for Safety Related Software in Defence Equipment," Part 1: Requirements, Part 2: Guidance," United Kingdom Ministry of Defence, August 1997.

[DEFSTAN0060] Defence Standard 00-60, “Integrated Logistic Support”, Issue 2, “Logistic Support Analysis Application to Software Aspects of Systems”, Part 3, United Kingdom Ministry of Defence, March 1998.

[DO178B] RCTA/DO-178B/ED-12B, “Software Considerations in Airborne Systems and Equipment,” Federal Aviation Administration software standard, RTCA Inc., December 1992.

[DO248B] RCTA/DO-128, Final Report for Clarification of DO-178B, " Software Considerations in Airborne Systems and Equipment,” Prepared by SC-190, October 12, 2001.

[IEC61508] ISO/IEC 61508, Edition 1.0: "Functional safety of electrical/electronic/programmable electronic safety-related systems," Multi-part standard, International Electrotechnical Commission, 1998.

[IEC61713] ISO/IEC 61713, Edition 1.0: "Software dependability through the software life-cycle processes - Application guide," International Electrotechnical Commission, June 30, 2000.

[IEC61719] ISO/IEC 61719 (Draft): "Guide to measures to be used for the quantitative dependability assessment of software," ISO/IEC/TC56/SC7/WG10/N111, Draft February 11, 2000.

[IEEE12207-0] IEEE/EIA Std 12207.0-1996, “Software life cycle processes,” IEEE Computer Society, March 1998.

[IEEE12207-1] IEEE/EIA Std 12207.1-1997, “Software life cycle processes - Life cycle data,” IEEE Computer Society, April 1998.

[IEEE12207-2] IEEE/EIA Std 12207.2-1997, “Software life cycle processes – Implementation considerations,” IEEE Computer Society, April 1998.

[IEEE610] IEEE Std-610.12-1990, “IEEE Standard Glossary of Software Engineering Terminology,” IEEE Computer Society, September 1990.

[IEEE982-1] IEEE Std-982.1-1988, "IEEE Standard Dictionary of Measures to Produce Reliable Software," IEEE Computer Society, June 1988.

[IEEE982-2] IEEE Std-982.2-1988, "IEEE Guide for the use of Standard Dictionary of Measures to Produce Reliable Software," IEEE Computer Society, September 1988.

[IEEE1228] IEEE Std-1228-1994,"IEEE Standard for Software Safety Plans," ," IEEE Computer Society, March 1994.

[ISO12207] ISO/IEC 12207, "Software Life Cycle Processes," August 1, 1995.

[MILSTD882D] MIL-STD-882D, " Department of Defense Standard Practice for System Safety," Department of Defense, February 10, 2000.

[MISRA-VBS] ISO/TR 15497, "Development Guidelines for Vehicle Based Software, the Motor Industry," Motor Industry Software Reliability Association, ISBN 0 9524156 0 7, November 1994.

[NATO96] NATO (Draft), "COTS Software Acquisition Guidelines and COTS Policy Issues – 1st Revision," NATO Communications and Information Systems Agency, January 12, 1996.

[NATO97] NATO (Draft), "NATO Guidelines for the Integration of Off-The-Shelf Software," Working Paper AC/322(SC/5)WP/4, NATO C3 Board Information Systems Sub-Committee, June 30, 1997.

[NIST800-14] NIST 800-14, "Generally Accepted Principles and Practices for Securing Information Technology Systems," National Institute for Standards and Technology, 1996.

[NIST800-26] NIST 800-26, "Security Self-Assessment Guide for Information Technology Systems," National Institute for Standards and Technology, 2001.

[NIST800-27] NIST 800-27, "Engineering Principles for Information Technology Security (A Baseline for Achieving Security)," National Institute for Standards and Technology, 2001.

[NUREG6421] NUREG/CR-6421, "A Proposed Acceptance Process for Commercial Off-the-Shelf (COTS) Software in Reactor Applications," Office of Nuclear Reactor Regulation, US Regulatory Commission, March 1996.

[SPICE98] ISO/IEC TR 15504:1998: "Software Process Improvement Capability Determination (SPICE) - Software Process Assessment," ISO/IEC/JTC1/SC7/WG10/N111, ISO 1998.

B.3 PUBLICATIONS

[BASILI02] Basili, Vic, Boehm, Barry, and others, "What We Have Learned About Fighting Defects," Proceedings of the Eighth IEEE Symposium on Software Metrics (METRICS™02), IEEE Computer Society, 2002. http://www.CeBASE.org

[DACS02] DACS CD, "Software Reliability Source Book," Data and Analysis Center for Software, Rome, NY, 2002. http://iac.dtic.mil/dacs/

[FALLA96] Falla, Mike, "Results and Achievements from the DTI/EPSRC R&D Programme in Safety Critical Systems," Compiled and Edited by Mike Falla, Motor Industry Software Reliability Association, November 1996. http://www.comp.lancs.ac.uk/computing/resources/scs/

[HELAN98] Helander, M., Shao, M., and Ohlsson, N. "Planning Models for Software Reliability and Cost," IEEE Transactions on Software Engineering, Vol 24, Number 6, June 1998, pp 420-434.

[HERRM99] Herrmann, D., Software Safety and Reliability: Techniques, Approaches, and Standards of Key Industrial Sectors, IEEE Computer Society, Los Alamitos, CA, 1999.

[LEVESON95] Leveson, Nancy G., Safeware: System Safety and Computers, Addison Wesley Publishing Company, 1995.

[LITTLEWD01] Littlewood, Bev, "Software Reliability and Dependability: A Roadmap," Centre for Software Reliability, City University, Northampton Square, London, UK, 2001.

[LYU96] Lyu, Michael, Handbook of Software Reliability Engineering, McGraw Hill / IEEE CS Press, 1996.

[MUSA92] Musa, John D. "Operational Profiles in Software Reliability Engineering," IEEE Software, March 1993, pages 14-32.

[MUSA99] Musa, John D., Software Reliability Engineering, McGraw-Hill Book Company, NY, 1999

[NEUF02] Neufelder-Owner, A., N., “The Facts About Predicting Software Defects and Reliability,” Journal of the RAC, 2ndQ, 2002, pp 1-4.

[PRIM97] PRIM-97, "Worldwide Reliability & Maintainability Standards," Reliability Analysis Center, IIT Research Institute / Reliability Analysis Center, Rome, NY, 1997.

[ROME97] Lakey, Peter and Neufelder, Ann Marie, “System and Software Reliability Assurance Notebook,” Rome Laboratory Report, Griffiss Air Force Base, Rome NY, 1997. http://www.cs.colostate.edu/~cs530/rh/master01.pdf

[SCHN97] Schneidewind, N., “Reliability Modeling for Safety-Critical Software,” IEEE Transactions on Reliability, Vol 46, Number 1, March 1997, pp 88-98.

[SSSHDBK99] Joint Software System Safety Committee and EIA G-46 Committee, Software System Safety Handbook, Joint Services Computer Resources Management Group, U.S. Navy, U.S. Army, U.S. Air Force, December 1999.

[XTALK03] CrossTalk, "Programming Languages," Journal of Defense Software Engineering, Vol. 16 No. 2, February 2003.

B.4 Interesting Web Sites

[G11SW] http://www.sae.org/TECHCMTE/g11soft.htm , G-11SW Committee

[IEEECS] http://www.ieee.computer.org/ , IEEE Computer Society

[ILS0060] http://www.dstan.mod.uk/dsmain.htm , UK MOD ILS

[ISO] http://www.iso.ch/ , International Standards Organization (ISO)

[LOGSA] http://www.logpars.army.mil/alc/logEngr.htm , Acquisition Logistics Center part of the US Army Materiel Command Logistic Support Activity (LOGSA), Mil-PRF & MIL-HDBK documents

[RAC] http://iitri.com/RAC/ , Reliability Analysis Center

[SEI] http://www.sei.cmu.edu/ , Software Engineering Institute

[SOLE] http://www.sole.org/ , International Society of Logistics (SOLE)

[SPMN] http://www.spmn.com/index.html , Software Program Manager's Network

[STSC] http://www.stsc.hill.af.mil/ , Software Technology Support Center

[USC] http://sunset.usc.edu/COCOMOII/cocomo.html , COCOMO Project

[MUSA] http://members.aol.com/JohnDMusa/ , John Musa

[CSR] http://www.csr.ncl.ac.uk:80/ , Centre for Software Reliability, Newcastle University, UK

[TRIVEDI] http://www.ee.duke.edu/~kst/ , Dr Kishor Trivedi, Duke University

[FAA-ACS] http://av-info.faa.gov/software/ , FAA Aircraft Certification Service Software

References to User Experiences with Software Reliability Engineering

JOHN D. MUSA
Website http://members.aol.com/JohnDMusa/
Updated August 25, 2000

If you have written or know of a published article not on this list, please send it to me in the citation format shown. Please send only references to articles written about the actual use by project personnel of SRE on real projects (data collection and analysis is not sufficient). To qualify as SRE, the project must have developed and used operational profiles, set and applied failure intensity objectives, or measured failure intensity and used it in managing the project. The reason for imposing these requirements is to limit the list to articles that new users can learn from and apply.

Send E-mail to: mailto:j.musa@ieee.org

Alam, M., W. Chen, W. Ehrlich, M. Engel, D. Kropfl, P. Verma. 1997. Assessing software reliability performance under highly critical but infrequent event occurrences. Proceedings 8th International Symposium on Software Reliability Engineering, Albuquerque, NM, November 1997, pp. 294-307.

Beck, A. 1998. "ESSI process improvement experiment 23843 - USST usage specification and statistical testing." Proceedings 8th International Symposium on Software Reliability Engineering: Case Studies, Albuquerque, NM, November 1997, pp. 95-100.

Bennett, J., Denoncourt, M., and Healy, J. D. 1992. "Software Reliability Prediction for Telecommunication Systems," Proc. 2nd Bellcore/Purdue Symposium on Issues in Software Reliability Estimation, Oct. 1992, pp. 85-102.

Bentz, R. W. and C. D. Smith. 1996. Experience report for the software reliability program on a military system acquisition and development. Proceedings 7th International Symposium on Software Reliability Engineering - Industrial Track, White Plains NY, October 30-November 2, 1996, pp. 59-65.

Bergen, L. A. 1989. "A Practical Application of Software Reliability to a Large Scale Switching System," IEEE International Workshop: Measurement of Quality During the Life Cycle, Val David, Quebec, Canada, April 25-27, 1989.

Carman, D. W., Dolinsky, A. A., Lyu, M. R., and Yu, J. S. 1995. "Software Reliability Engineering Study of a Large-Scale Telecommunications Software System," Proc. 1995 International Symposium on Software Reliability Engineering, Toulouse, France, Oct. 1995, pp. 350-.

Carnes, P. 1997. "Software reliability in weapon systems." Proceedings 8th International Symposium on Software Reliability Engineering: Case Studies, Albuquerque, NM, November 1997, pp. 95-100.

Carnes, P. 1998. "Software reliability in weapon systems." Proceedings 9th International Symposium on Software Reliability Engineering: Industrial Practices, Paderborn, Germany, November 1998, pp.272-279.

Christenson, D. A. 1988. "Using Software Reliability Models to Predict Field Failure Rates in Electronic Switching Systems," Proc. 4th Annual National Joint Conference on Software Quality and Productivity, Washington, DC.

Chruscielski, K. and J. Tian. 1997. An operational profile for the cartridge support software. Proceedings 8th International Symposium on Software Reliability Engineering, Albuquerque, NM, November 1997, pp. 203-212.

Cramp, R., Vouk, M. A., and Jones, W. 1992. "On Operational Availability of a Large Software-Based Telecommunications System," Proc. 3rd International Symposium on Software Reliability Engineering, Research Triangle Park, NC, Oct. 7-10, 1992, pp. 358-366.

Cusick, J. and M. Fine. 1997. Guiding reengineering with the operational profile. Proceedings 8th International Symposium on Software Reliability Engineering: Case Studies, Albuquerque, NM, November 1997, pp. 15-25.

Derriennic, H. and G. Le Gall. 1995. Use of failure-intensity models in the software-validation phase for telecommunications. IEEE Transactions on Reliability 44(4):658-665.

Dixit, P., M. A. Vouk, D. L. Bitzer, and C. Alix. 1996. Reliability and availability of a wide area network-based education system. Proceedings 7th International Symposium on Software Reliability Engineering, White Plains NY, October 30-November 2, 1996, pp. 213-218.

Dixit, P., M. A. Vouk, and D. L. Bitzer. 1997. Reliability behavior of a large network based education system. Proceedings 8th International Symposium on Software Reliability Engineering: Case Studies, Albuquerque, NM, November 1997, pp. 43-56.

Donnelly, M., Everett, B., Musa, J., and Wilson, G. 1996. "Best Current Practice of SRE," Lyu, M. R. (ed.), Handbook of Software Reliability Engineering, McGraw-Hill, 1996, pp. 219-254.

Drake, H. D. and D. E. Wolting. 1987. Reliability theory applied to software testing. Hewlett-Packard Journal 38(4):35-39.

Ehrlich, W. K., R. Chan, W. J. Donnelly, H. H. Park, M. B. Saltzman, and P Verma. 1996. Validating software architectures for high reliability. Proceedings 7th International Symposium on Software Reliability Engineering, White Plains NY, October 30-November 2, 1996, pp. 196-206.

Ehrlich, W. K., Lee, K., and Molisani, R. H. 1990. "Applying Reliability Measurements: A Case Study," IEEE Software, March 1990.

Ehrlich, W. K., Prasanna, B., Stampfel, J. P., and Wu, J. R. 1993.

"Determining the Cost of a Stop-Test Decision," IEEE Software, March 1993, pp. 33-42.

Ehrlich, W. K., Stampfel, J. P., and Wu, J. R. 1990. "Application of Software Reliability Modeling to Product Quality and Test Process," Proc. 12th International Conference on Software Engineering, Nice, France, March 1990.

Elentukh, A. 1994. "System Reliability Policy at Motorola Codex," Proc. 5th International Symposium on Software Reliability Engineering, Monterey, CA, Nov. 6-9, 1994, pp. 289-293.

Everett, W. W. and J. M. Gobat. 1996. DQS's experience with SRE. Proceedings 7th International Symposium on Software Reliability Engineering, White Plains NY, October 30-November 2, 1996, pp. 219-224.

Fuoco, G., Irving, N., Juhlin B., Kropfl, D., and Musa, J. 1996. "The Operational Profile," Lyu, M. R. (ed.), Handbook of Software Reliability Engineering, McGraw-Hill, 1996, pp. 167-216 (includes three project applications).

Hamilton, P. A. and Musa, J. D. 1978. "Measuring Reliability of Computation Center Software," Proc. 3rd International Conference on Software Engineering, pp. 29-36.

Hill, S. W. and F. S. Kmetz. 1997. Application of software reliability engineered testing (SRET) to project accounting application (PAA). Proceedings 8th International Symposium on Software Reliability Engineering: Case Studies, Albuquerque, NM, November 1997, pp. 59-68.

Hudepohl, J. P. Measurement of software service quality for large telecommunications systems. IEEE Journal on Selected Areas in Communications 8(2):210-218.

Hudepohl, J. P., W. Snipes, T. Hollack, and W. Jones. A methodology to improve switching system software service quality and reliability. Proceedings IEEE Global Communications Conference, pp. 1671-1678.

Iannino, A and Musa, J. D. 1991. "Software Reliability Engineering at AT&T," Apostolakis, G. (ed.) Probability Safety Assessment and Management - Vol. 1, Elsevier, New York.

Jenson, B. D. 1995. "A Software Reliability Engineering Success Story: AT&T's Definity(PBX," Proc. 1995 International Symposium on Software Reliability Engineering, Toulouse, France, Oct. 1995, pp. 338-343.

Jones, W. D. 1991. "Reliability Models for Very Large Software Systems in Industry," Proc. 1991 International Symposium on Software Reliability Engineering, Austin, TX, May 17-18, 1991, pp. 35-42.

Jones,W.D. 1998. "A Brief History of SRE in PCN," Proc. 9th Annual SRE Workshop 7/14-15/98, Ottawa, Ontario, Canada.

Juhlin, B. D. 1992. "Implementing Operational Profiles to Measure System Reliability," Proc. 3rd International Symposium on Software Reliability Engineering, Research Triangle Park, NC, Oct. 7-10, 1992, pp. 286-295.

Juhlin, B. D. 1992. "Applying Software Reliability Engineering to International PBX Testing," Proc. 9th International Conference on Testing Computer Software, Washington, DC, June 16-18, 1992, pp. 165-176.

Juhlin, B. D. 1993. "Software Reliability Engineering in the System Test Process," Proc. 10th International Conference on Testing Computer Software, Washington, DC, June 14-17, 1993, pp. 97-115.

Kaâniche, M. and K. Kanoun. 1996. Reliability of a commercial telecommunications system. Proceedings 7th International Symposium on Software Reliability Engineering, White Plains NY, October 30-November 2, 1996, pp. 207-212.

Kanoun, K and Sabourin, T. 1987. "Software Dependability of a Telephone Switching System," Proc. 17th IEEE International Symposium on Fault-Tolerant Computing, Pittsburgh, June 1987, pp. 236-241.

Kanoun, K., Bastos Martini, M., and Moreira de Souza, J. 1991. "A Method for Software Reliability Analysis and Prediction-Application to the TROPICO-R Switching System," IEEE Trans. Software Engineering, April 1991, pp. 334-344.

Keller, T. and N. Schneidewind. 1997. Successful application of software reliability engineering for the NASA space shuttle. Proceedings 8th International Symposium on Software Reliability Engineering: Case Studies, Albuquerque, NM, November 1997, pp. 71-82.

Kropfl, D. and Ehrlich, W. 1995. "Telecommunications Network Operating Systems: Experiences in Software Reliability Engineering," Proc. 1995 International Symposium on Software Reliability Engineering, Toulouse, France, Oct. 1995, pp. 344-349.

Kruger, G. A. 1988. Project management using software reliability growth models. Hewlett-Packard Journal 39(6):30-35.

Kruger, G. A. 1989. Validation and further application of software reliability growth models. Hewlett-Packard Journal 40(4):75-79.

Lakey, Peter B. 1998. "How does any software organization proceed in incorporating SRE?" (Crusader self-propelled howitzer project) Proc. 9th Annual SRE Workshop 7/14-15/98, Ottawa, Ontario, Canada.

Lee, I. and R. K. Iyer. 1995. Software dependability in the Tandem GUARDIAN system. IEEE Transactions on Software Engineering 21(5):455-467.

Levendel, Y. 1989. "Defects and Reliability Analysis of Large Software Systems: Field Experience," Proc. 19th IEEE International Symposium on Fault-Tolerant Computing, Chicago, June 1989, pp. 238-244.

Levendel, Y. 1990. "Reliability Analysis of Large Software Systems: Defect Data Modeling," IEEE Trans. Software Engineering, vol. SE-16, no. 2, February 1990, pp. 141-152.

Levendel, Y. 1995. "The Cost Effectiveness of Telecommunication Service Dependability," Lyu, M. R. (ed.), Software Fault Tolerance, Wiley and Sons, pp. 279-314.

Martini, M. R., Kanoun, K. and de Souza, J. M. 1990. "Software Reliability Evaluation of the TROPICO-R Switching System, IEEE Trans. Reliability, vol. 33, no. 3, pp. 369-379.

Mendiratta, Veena B. 1998. "Reliability Analysis of Clustered Architectures," Proc. 9th Annual SRE Workshop 7/14-15/98, Ottawa, Ontario, Canada.

Musa, J. D., G. Fuoco, N. Irving, B. Juhlin, and D. Kropfl. 1996. The operational profile. In Handbook of Software Reliability Engineering, ed. M. R. Lyu, McGraw-Hill, 1996, pp. 167-216 (includes three project applications).

Nikora, A. P. and Lyu, M. R. 1996. "Software Reliability Measurement Experiences," Lyu, M. R. (ed.), Handbook of Software Reliability Engineering, McGraw-Hill, pp. 255-301.

Oshana, R. and F. P. Coyle. 1997. Improving a system regression test with usage models developed using field collected data. Proceedings Software Quality Week 1997.

Pemler, S. and Stahl, N. 1994. "An Automated Environment for Software Testing and Reliability Estimation," Proc. 5th International Symposium on Software Reliability Engineering, Monterey, CA, Nov. 6-9, 1994, pp. 312-317.

Rapp, B. 1990. "Application of Software Reliability Models in Medical Imaging Systems," Proc. 1990 International Symposium on Software Reliability Engineering, Washington, DC, April 1990.

Sandfoss, R. V. and S. A. Meyer. 1997. Input requirements needed to produce an operational profile for a new telecommunications system. Proceedings 8th International Symposium on Software Reliability Engineering: Case Studies, Albuquerque, NM, November 1997, pp. 29-39.

Schneidewind, N. F. and Keller, T. W. 1992. "Application of Reliability Models to the Space Shuttle," IEEE Software, July 1992, pp. 28-33.

Teresinski, J. A. 1996. Software reliability: getting started. Proceedings 7th International Symposium on Software Reliability Engineering - Industrial Track, White Plains NY, October 30-November 2, 1996, pp. 39-47.

Tian, J. , P. Lu, and J. Palma. 1995. Test-execution based reliability measurement and modeling for large commercial software. IEEE Transactions on Software Engineering 21(5):405-414.

Tierney, J. 1996. Putting aspects of software reliability engineering to use. Proceedings 7th International Symposium on Software Reliability Engineering - Industrial Track, White Plains NY, October 30-November 2, 1996, pp. 89-92.

Weinberg, T. 1996. SoothSayer: a tool for measuring the reliability of Windows NT services. Proceedings 7th International Symposium on Software Reliability Engineering - Industrial Track, White Plains, NY, October 30-Nivember 2, 1996, pp. 49-56.

Appendix C

Example Software Failures

1. Therac 25 Accidents (6), June 1985 - January 1987

Leveson, Nancy, "An Investigation of the Therac-25 Accidents," IEEE Computer, July 1993, pp 18-41.

Between June 1985 and January 1987, six known accidents involved massive overdoses by the Therac-25 (radiation treatment equipment), with resultant deaths and serious injuries. They have been described as the worst series of radiation accidents in the 35-year history of medical accelerators.

The mistakes that were made are not unique to this manufacturer but are, unfortunately, fairly common in other safety-critical systems.

"A significant amount of software for life-critical systems comes from small firms, especially in the medical device industry; firms that fit the profile of those resistant to or uninformed of the principles of either system safety or software engineering."

Most accidents are SYSTEM accidents; stem from complex interactions between various components and activities. It would be a serious mistake to attribute a single cause to an accident.

However, the facts in these cases are that software was a major contributor to each of the six accidents. Major changes in procedures, hardware interlocks, and software code were requested by the FDA and were made.

1. Airbus A320-211, September 14, 1993

Summary on pg 130 in Herrmann, Debra, Software Reliability and Safety, Computer Society Press, IEEE Inc, Piscataway, NJ, 1999.

Airbus A320-211 crashed in Warsaw, September 14, 1993 killing two people. A variety of factors contributed to the hazardous consequences, including a software requirements specification defect. Weather conditions caused the aircraft to hydroplane which caused a delay in the air-to-ground transition trigger from the computer because both wheels did not reach the "required speed and exhibit the required landing gear compression" per software specification.

1. Ariane 501 Disaster, June 4, 1996

From the official report as contained at: http://www.esrin.esa.it/htdocs/tidc/Press/Press96/ariane5rep.html

- 10 years, $7 billion, Ariane giant rocket for launching 3-ton satellites - commercial space business for Europe
- 39 seconds after launch, altitude of 2 1/2 miles, self-destruct mechanism destroyed Ariane 5 & its payload of 4 scientific satellite; aerodynamic forces were ripping the boosters from the rocket.
- Spacecraft swerved off course under pressure three powerful nozzles in its boosters and main engine; rocket made unneeded abrupt course correction, compensating for a wrong turn not taken.
- Steering controlled on-board computer, thought the rocket needed a course change because of numbers from the inertial guidance system. Numbers were actually diagnostic error message. The guidance system had shut down (@36.7 sec) when the guidance system's computer tried to convert one piece of data, the sideways velocity of the rocket, from a 64-bit format to a 16-bit format. The number was too big, an overflow error resulted, the guidance system shut down and passed control to an identical, redundant unit, there to provide backup in case of just such a failure. The second unit had failed in the identical manner a few milliseconds before. It was running the same software.
- Decision was made that this particular velocity figure would never be large enough to cause trouble. Unluckily, Ariane 5 was a faster rocket than Ariane 4. The calculation containing the bug served no purpose once the rocket was in the air - only function was to align the system before launch; should have been turned off but engineers chose long ago, in an earlier version of the Ariane, to leave this function running for the first 40 secs of flight -- to make it easy to restart the system in the event of a brief hold in the countdown.
- CAUSE OF THE FAILURE

The failure of the Ariane 501 was caused by the complete loss of guidance and altitude information 37 seconds after start of the main engine ignition sequence (30 seconds after lift- off). This loss of information was due to specification and design errors in the software of the inertial reference system. The extensive reviews and tests carried out during the Ariane 5 Development Programme did not include adequate analysis and testing of the inertial reference system or of the complete flight control system, which could have detected the potential failure.

1. Friendly Fire Deaths, March 2002

Jamie McCarthy <jamie@mccarthy.vg>

Tue, 26 Mar 2002 10:47:52 -0500

In one of the more horrifying incidents I've read about, U.S. soldiers and allies were killed in December 2001 because of a stunningly poor design of a GPS receiver, plus "human error."

http://www.washingtonpost.com/wp-dyn/articles/A8853-2002Mar23.html

A U.S. Special Forces air controller was calling in GPS positioning from some sort of battery-powered device. He "had used the GPS receiver to calculate the latitude and longitude of the Taliban position in minutes and seconds for an airstrike by a Navy F/A-18."

According to the *Post* story, the bomber crew "required" a "second calculation in 'degree decimals'" -- why the crew did not have equipment to perform the minutes-seconds conversion themselves is not explained.

The air controller had recorded the correct value in the GPS receiver when the battery died. Upon replacing the battery, he called in the degree-decimal position the unit was showing -- without realizing that the unit is set up to reset to its *own* position when the battery is replaced.

The 2,000-pound bomb landed on his position, killing three Special Forces soldiers and injuring 20 others.

If the information in this story is accurate, the RISKS involve replacing memory settings with an apparently-valid default value instead of blinking 0 or some other obviously-wrong display; not having a backup battery to hold
values in memory during battery replacement; not equipping users to translate one coordinate system to another (reminiscent of the Mars Climate Orbiter slamming into the planet when ground crews confused English with metric); and using a device with such flaws in a combat situation.

1. Air-traffic control software reliability, May 2002

"Peter B. Ladkin" <ladkin@rvs.uni-bielefeld.de>

Wed, 15 May 2002 10:03:39 +0200

An article in *Aviation Week and Space Technology*, "Why Controllers Are Skeptics Regarding New Technology", by Bruce Nordwall, 6 May 2002, pp.50-51, tells the following tale recounted recently at an air-traffic controllers' conference by Philippe Domogola, supervisor at the Maastricht Upper Area Control Center.

"Some years ago," a new European ATC center installed software specified as "99.99% reliable", which apparently meant 99.99% availability in each calendar year, or a maximum of roughly 52 minutes down-time per year. The software "failed" a couple of months after installation, and suffered 20 hours down-time. "The manufacturer's conclusion was: human error that will not happen again" (come to think of it, any specific software bug can be put down to "human error that will not happen again").

Someone had forgotten about leap years. It failed at 23:59 on February 28.

Some controllers suggested that since the software was "99.99% reliable" and it had already been unavailable for 20 hours, it follows there were going to be no more failures for the next 25 years.

They were right. It does follow.

Peter B. Ladkin, University of Bielefeld, Germany
http://www.rvs.uni-bielefeld.de

1. Impact of inadequate software testing on US economy, June 2002

Rick Kuhn <kuhn@nist.gov>

Wed, 05 Jun 2002 14:53:35 -0400

http://www.nist.gov/director/prog-ofc/report02-3.pdf

NIST has released a new study conducted by the Research Triangle Institute that should be of interest to readers: "The Economic Impacts of Inadequate Infrastructure for Software Testing". From the summary:

NIST engaged the Research Triangle Institute (RTI) to assess the cost to the U.S. economy of inadequate software testing infrastructure. Inadequate testing is defined as failure to identify and remove software bugs in real time. Over half of software bugs are currently not found until downstream in the development process leading to significant economic costs. RTI identified a set of quality attributes and used them to construct metrics for estimating the cost of an inadequate testing infrastructure. Two in depth case studies were conducted. In the manufacturing sector, transportation equipment industries were analyzed. Data were collected from software developers (CAD/CAM/CAE and product data management vendors) and from users (primarily automotive and aerospace companies). In the service sector, financial services were analyzed with data collected again from
software developers (routers and switches, financial electronic data interchange, and clearinghouse) and from users (banks and credit unions). ...the annual cost to these two major industry groups from inadequate software infrastructure is estimated to be $5.85 billion. Similarities across industries with respect to software development and use and, in particular, software testing labor costs allowed a projection of the cost to the entire U.S. economy. Using the per-employee impacts for the two case studies, an extrapolation to other manufacturing and service industries yields an approximate estimate of $59.5 billion as the annual cost to the nation of inadequate software testing infrastructure.

1. Army Training Accident, June 2002

Steve Bellovin <smb@research.att.com>

Thu, 13 Jun 2002 09:38:10 -0400

According to a U.S. Army report, a software problem contributed to the deaths of two soldiers in a training accident at Fort Drum. They were firing artillery shells, and were relying on the output of the Advanced Field Artillery Tactical Data System. But if you forget to enter the target's altitude, the system assumes a default of 0. (A Web site I found indicates that (part of) Ft. Drum is at 679 feet above sea level.) The report goes on to warn that soldiers should not depend exclusively on this one system, and should use other computers or manual calculations.

Other factors in the incident include the state of training of some of the personnel doing the firing. [Source: AP]

1. Questions About New Air-Traffic Computer System, June 2002

Ian Macky <ian.macky@oracle.com>

Wed, 5 Jun 2002 14:10:15 -0700 (PDT)

There are some highly scary quotes in this article regarding the new STARS (Standard Terminal Automation Replacement System) which is supposed to replace the hodge-podge of old air-traffic control systems:

http://www.cnn.com/2002/TRAVEL/NEWS/06/05/faa.airtraffic.ap/index.html

Players are the FAA Union (representing the flight controllers), the FAA technicians who are trying to roll out the new system, the equipment builder, Raytheon Co., and the DOT (Department of Transportation). [...]

"DOT Inspector General Kenneth Mead ... said there were 71 specific software problems that could prevent the system from operating as designed, or could threaten safety or security. " "Mead said controllers in El Paso had to track airplanes manually because the computer system didn't properly display the flights."

Union vice president Tom Brantley: "They don't believe it's operationally suitable," Brantley said. "It's failing. It has a lot of errors. They can't verify that it works because it fails a lot of the tests."

FAA spokesman Scott Brenner said the only problems are the normal bugs (!) that accompany any new technology. [Ship it!] "When the [FAA] technicians refused to certify the system in Syracuse, New York, the FAA invoked a never-before-used [emergency] clause in its contract with its employees and ordered them to approve the equipment.
The Syracuse system was turned on Monday night." Brantley: "The emergency clause was never intended for something like this. That was intended if there were an actual emergency."

Blanche Necessary (!), a spokeswoman for the equipment builder, Raytheon Co., said the system was working well in El Paso and Syracuse. etc., etc.

The RISKS are painfully familiar. Feel safer flying?

1. Software "glitch" Changes the Colour of the Universe, March 2002

Pete Mellor <pm@csr.city.ac.uk>

Wed, 13 Mar 2002 00:35:43 +0000 (GMT)

As reported on the "Broadcasting House" programme on BBC Radio 4, Sunday 10th March:-

Scientists at John Hopkins University have spent several years calculating the weighted average of the electromagnetic frequency of emissions from all galaxies in the observable universe. They concluded their research by announcing last month that, on average, the universe is turquoise.

Last week, they announced that, due to a software "glitch", they had miscalculated, and that the universe is, in fact, beige.

Broadcasting House are threatening legal action, claiming that they have just had their studio painted turquoise in order to be in harmony with the rest of the universe.

Peter Mellor, Centre for Software Reliability, City University,
Northampton Square, London EC1V 0HB UK NEW Tel.: +44 (0)20 7040 8422

From the IT department at Franciscan

Computer Zen

In Japan, they have replaced the impersonal and unhelpful Microsoft error messages with Haiku poetry messages.

Haiku poetry has strict construction rules. Each poem has only three lines, 17 syllables: five syllables in the first line, seven in the second, five in the third.

Haikus are used to communicate a timeless message often achieving a wistful, yearning and powerful insight through extreme brevity....the essence of Zen!

Here are some examples:

Your file was so big.
It might be very useful.
But now it is gone.

The Web site you seek
Cannot be located, but
Countless more exist.

Chaos reigns within.
Reflect, repent, and reboot.
Order shall return.

Program aborting:
Close all that you have worked on.
You ask far too much.

Windows NT crashed.
I am the Blue Screen of Death.
No one hears your screams.

Yesterday it worked.
Today it is not working.
Windows is like that.

First snow, then silence.
This thousand-dollar screen dies
So beautifully.

With searching comes loss
And the presence of absence:
"My Novel" not found.

Stay the patient course.
Of little worth is your ire.
The network is down.

A crash reduces
Your expensive computer
To a simple stone.

Three things are certain:
Death, taxes and lost data.
Guess which has occurred.

You step in the stream,
But the water has moved on.
This page is not here.

Having been erased,
The document you're seeking
Must now be retyped.

Serious error.
All shortcuts have disappeared.
Screen. Mind. Both are blank.

Appendix D

Case Study Materials

Software Reliability Plan Thematic Outline

1. MANAGING THE SOFTWARE RELIABILITY PROGRAM ACTIVITIES
   - 1.1 Define purpose, scope of plan and program, reliability goals and objectives
   - 1.2 Nomenclature and project references
   - 1.3 Program management functions: responsibility, authority, interaction between system and software reliability programs; customer interaction/involvement; risk management
   - 1.4 Resources needed, including personnel and equipment
   - 1.5 Schedule
   - 1.6 Training
   - 1.7 Subcontract Management
   - 1.8 Plan approval and maintenance
2. PERFORMING SOFTWARE RELIABILITY PROGRAM ACTIVITIES
   - 2.1 Determine Customer Requirements
     - 2.1.1 Establish supplier-customer dialogue
     - 2.1.2 Identify operational conditions of use
     - 2.1.3 Define in-service conditions of support
     - 2.1.4 Establish metrics: goals, assumptions and claims, and expected evidence
     - 2.1.5 Develop plan
     - 2.1.6 Document pre-development case evidence
   - 2.2 Meet Customer Requirements
     - 2.2.1 Define lifecycle model, methodology, interaction with system engineering
     - 2.2.2 Identify specific static and dynamic analyses to be performed throughout lifecycle, and associated progress reporting approach
     - 2.2.3 Perform design, implementation, test activities
     - 2.2.4 Document development case evidence
   - 2.3 Demonstrate Customer Requirements
     - 2.3.1 Qualify the product and process
     - 2.3.2 Establish process controls
     - 2.2.3 Transition to operational environment
     - 2.2.4 Training end-users, operations and support staff
     - 2.3.5 Pursue continuous improvement
     - 2.3.6 Establish data collection and reporting
     - 2.3.7 Document post-development case evidence
3. DOCUMENTING SOFTWARE RELIABILITY PROGRAM ACTIVITIES
   - 3.1 Lifecycle practices
   - 3.2 Software reliability case file of evidence

Software Reliability Case Thematic Outline

1. SOFTWARE RELIABILITY GOALS AND OBJECTIVES
   - 1.1 What they are, overall and for partitions
   - 1.2 How were they derived, apportioned to software and partitions
   - 1.3 Relation to system reliability goals
   - 1.4 Regulatory and/or contractual requirements
2. ASSUMPTIONS AND CLAIMS
   - 2.1 Assumptions: agreed upon constraints and basis for claims
   - 2.2 Claims: agreed upon validation and certification criteria
3. EVIDENCE

   For each of the phases: Pre-development/Development/Post-development/In-Service
   - 3.1 Process activities that demonstrate achievement of software reliability goals and objectives
   - 3.2 Product characteristics that demonstrate achievement of software reliability goals and objectives
   - 3.3 Qualifications of people and resources that demonstrate achievement of software reliability goals and objectives
4. CONCLUSION/RECOMMENDATION
   - 4.1 Summary of reliability goals, claims, and actual evidence provided
   - 4.2 Recommendations related to warranty, certification, qualification
5. CERTIFICATION RECORDS
   - 5.1 Record of all acceptance warranty, certification, qualification activities and results

Software Reliability Case Evidence Guidelines

1. SOFTWARE RELIABILITY GOALS AND OBJECTIVES

The information in the software reliability case file must correlate with the specified software reliability goals. Hence, the software reliability goals and objectives should be stated first, for the system and individual partitions, as appropriate. The process by which the reliability goals were derived and apportioned to software should be described. The relationship between the system and software reliability goals should be explained. Any regulatory and/or contractual reliability requirements should be highlighted. In addition, the agreed upon validation and certification criteria should be noted.

2. ASSUMPTIONS AND CLAIMS

All assumptions, such as citing existing systems or research, and claims made relative to achievement and assessment of the software reliability goals and objectives should be clearly stated and justified.

3. EVIDENCE

Three categories of evidence should be supplied in the software reliability case file: process activities, product characteristics, and qualifications of people and resources that demonstrate achievement of software reliability goals. As an introduction to the evidence, this information can be summarized as shown in the Table below.

System:_____

Intended Use/Environment:_____

Phase/Date:_____

Table. Summary of Software Reliability Case Evidence

Reliability Control Measure	Product Evidence/ Safeguards	Process Evidence/ Safeguards	Resource Evidence/ Safeguards
Fault Elimination	- - - - -	- - -	- - -
Failure containment	- - - - -	- - -	- - -
Failure rate estimation	- - - - -	- - -	- - -

3.1 Process activities that demonstrate achievement of software reliability goals and objectives

A description of the selected lifecycle model and development methodology should be provided, including an explanation of how this model and methodology contribute(d) to the attainment and assessment of reliability goals throughout the lifecycle phases. Specific lifecycle activities that were used to assess software reliability should be called out, such as performing iterative risk analyses or using of static analysis techniques. An assessment should be made of the:

- software reliability design analysis,
- software reliability code analysis,
- software reliability change analysis, and
- effectiveness of validation and verification activities.

Suspected or confirmed reliability problems should be documented, along with the current status of their resolution. Results from analyzing and interpreting process metrics should also be discussed.

3.2 Product characteristics that demonstrate achievement of software reliability goals and objectives

A description of the design features which contribute to enhanced reliability should be provided, such as: partitioning, diversity, block recovery, independence, information hiding, and system/software fault tolerance. This description should explain how the likelihood of common cause failures has been eliminated or reduced. In addition, a discussion of whether the product: 1) operates in a demand-mode or continuous-mode environment; 2) was designed to fail safe or fail operational; and 3) contains any monitoring and/or error detection and correction features should be included. The results of static and dynamic analyses should be recorded, along with an analysis of the effectiveness of the reliability control measures. Results from analyzing and interpreting product metrics should also be discussed.

3.3 Qualifications of people and resources that demonstrate achievement of software reliability goals and objectives

An explanation of why the education, experience, and certification of the professional staff is appropriate for a project of this reliability level should be provided. Likewise, a justification of why the hardware and software platforms, including automated tools, are appropriate for this project should be provided. Results from analyzing and interpreting people/resource metrics should also be discussed.

4. CONCLUSION/RECOMMENDATION

The conclusion should summarize the information presented in sections 4.2.2 and 4.2.3 to demonstrate whether the software reliability goals and objectives have been met and make a recommendation regarding certification.

5. CERTIFICATION RECORDS

An accurate and complete chronological history of all certifications attempted should be maintained in the software reliability case file.

Contract for Acquisition
of
Commercial Aircraft Equipment with Software Product

1. INTRODUCTION

1.1. Purpose and Scope

1.2. Roles and Responsibilities

- customer
- supplier
- certification authority

1.3. Expectations

- objectives
- software process
- specialty engineering (reliability/safety)

2. REQUIREMENTS

2.1. Graded Formality

- software product level based on application and supplier survey of capabilities

2.2. Certification Requirements

- per DO178B with caveats per CAST-1 guidance/system safety/reliability level
- appropriate for the graded formality

2.3. Key Performance Parameters

- include software reliability case as part of verification plan/case/results evidence
- include software safety case as part of verification plan/case/results evidence
- appropriate for the graded formality

2.4. Supplier Survey

- include requirement to conduct supplier survey for final product acceptance

3. CONDITIONS AND CONSTRAINTS

3.1. Schedule

3.2. Budget

3.3. Deliverables

A. Appendix A – Supplier Survey

(a) RCTA/DO178B Certification Evidence Review

- minimal: Plan for Software Aspects of Certification
  Software Configuration Index
  Software Accomplishment Summary
  Results of certification authority review of associated equipment

(b) Key Performance Parameters Evidence Review

- additional: verification plan/case/results evidence for software reliability
  verification plan/case/results evidence for software safety

(c) Supplier Capabilities Checklist (FAA N 8110.87)

- Top Ten Practices Recommended by Neufelder
- Score: 0 – none; 0.5 – partial; 1.0 – complete

Table 1. Best Practices for Reliability

Practice	Supplier Score	Customer Score	Observations
All requirements are mapped to system tests			
Requirements are reviewed before designing or coding			
System test beds are used			
Test plan started at least one phase of the life cycle before testing begins			
Testers use a FRACAS (defect tracking system) to determine what to test/retest			
All upgrades after a system test are regression tested			
Correction action releases per year <= 4			
All modifications made after a system test are regression tested			
FRACAS used for tracking all corrective actions			
Walk-thrus are performed for all phases of life cycle			

TABLE 2. OTHER RELEVANT CRITERIA

	CRITERIA	Scale MIN.		MAX.	Score
1.	**Applicant/Developer Software Certification Experience**				
1.1	Experience with civil aircraft and systems certification.	Scale: 0 # projects: 0	5 3-5	10 6+	
1.2	Experience with DO-178B.	Scale: 0 # projects: 0	5 2-4	10 5+	
1.3	Experience with DO-178 or DO-178A.	Scale: 0 # projects: 0	3 4-6	5 7+	
1.4	Experience with other software standards (other than DO-178 [ ])	Scale: 0 # projects: 0	2 4-6	4 7+	
2.	**Applicant/Developer Demonstrated Software Development Capability**				
2.1	Ability to consistently produce DO-178B software products.	Scale: 0 Ability: Low	5 Med	10 High	
2.2	Cooperation, openness and resource commitments	Scale: 0 Ability: Low	5 Med	10 High	
2.3	Ability to manage software development and sub-contractors	Scale: 0 Ability: Low	5 Med	10 High	
2.4	Capability assessments (e.g., SEI CMM, ISO 9001-3, IEC)	Scale: 0 Ability: Low	2 Med	4 High	
2.5	Development team average relevant experience	Scale: 0 Ability: < 2 yrs	5 2-4 yrs	10 > 4 yrs	
3.	**Applicant/Developer Software Service History**				
3.1	Incidents of software-related problems. (as a percentage of affected products)	Scale: 0 Incidents: > 25%	5 > 10%	10 None	
3.2	Company management and support of designees	Scale: 0 Quality: Low	5 Med	10 High	
3.3	Company software quality assurance organization and configuration management process	Scale: 0 Quality: Low	5 Med	10 High	
3.4	Company stability and commitment	Scale: 0 Stability: Low	3 Med	6 High	
3.5	Success of past company certification efforts	Scale: 0 Success: None	3 >50%	6 All	
4.	**The Current System and Software Application**				
4.1	Complexity of the system architecture, functions and interfaces	Scale: 0 Complex: High	5 Med	10 Low	
4.2	Complexity & size of the software and safety features	Scale: 0 Complex: High	5 Med	10 Low	
4.3	Novelty of design and use of new technology	Scale: 0 Newness: Much	5 Some	10 None	
4.4	Software development and verification environment	Scale: 0 Environ: None	3 Older	6 Modern	
4.5	Use of alternative methods or additional considerations	Scale: 0 Standard: Much	3 Little	6 None	
5.	**Designee Capabilities**				
5.1	Experience of designees with DO-178B.	Scale: 0 Projects: <5	5 5-10	10 >10	
5.2	Designee authority, autonomy and independence.	Scale: 0 Autonomy: None	5 Self-starter	10 Outgoing	
5.3	Designee cooperation, openness and issue resolution effectiveness.	Scale: 0 Effectiveness: Non-Responsive	5 Responsive	10 Cooperative & Outgoing	
5.4	Relatedness of assigned designee's experience.	Scale: 0 Related: None	5 Somewhat	10 Exact	
5.5	Designees current workload on project and other projects.	Scale: 0 Workload: High	5 Medium	10 Low	
5.6	Experience of designees with other software standards (other than DO-178[ ]).	Scale: 0 Projects: <5	3 5-10	5 >10	

Total Score Result (TSR): _____

IMA Inc Level of FAA Involvement Assessment

1. IMA Inc COMPANY OVERVIEW

Applicant company IMA, Inc is applying for approval of their Integrated Modular Avionics (IMA) product that is usually approved for Technical Standard Order (TSO) projects and then the installation by Supplemental Type Certificate (STC) and installed on new or in-service aircraft. The equipment provides standard capability required by airlines, and is regularly upgraded for improved capabilities. IMA Inc has prior TSO approvals on a number of aircraft and recently upgraded the software aspects of their Load Control software product to RTCA DO-178B Level A criteria. In past programs, they have consistently demonstrated their willingness to commit the necessary resources and change their processes to utilize new technologies while maintaining a quality product and satisfying certification requirements.

IMA Inc's product service history indicates almost no in-service difficulties with their products and their technology and system architecture are fairly stable. Replacement of obsolete parts is being planned and seemingly being well managed. They appear to have a stable in-house process for managing changes, even though almost every different aircraft installation requires some changes to the software. The development and verification environment is state-of-the-practice and new tools are introduced when economically advantageous. The company contracts through job placement agencies for low-level software testers.

IMA Inc has 3 company designees on-site, 2 with software authority and 1 with electrical system authority, and the company occasionally contracts with a consultant Designated Engineering Representative (DER) for system approvals. One of the software DERs is very experienced and the other has been a DER for less than a year. The experienced software DER also is the manager for the software verification group, part of the engineering organization, and the less experienced software DER is in the company's SQA organization, which is independent of the engineering organization and has highly qualified and experienced personnel.

2. IMA Inc ASSESSMENT

An experienced software Aviation Safety Engineer (ASE) involved with several previous projects for the company, and having previously conducted 2 on-site reviews, assesses IMA Inc on the new project to deliver the IMA units, in particular the Load Control software. The results of the assessment:

Criteria 1: Application/Developer Software Certification Experience
Score: 20 out of possible 29
Criteria 2: Application/Developer Demonstrated Software Development Capability
Score: 21 out of possible 44
Criteria 3: Application/Developer Software Service History
Score: 36 out of possible 42
Criteria 4: The Current System and Software Application

Score: 26 out of possible 42

Criteria 5: Designee Capabilities

Score: 38 out of possible 55

Total Score Results (TSR): 141 out of possible 212

3. LEVEL OF FAA INVOLVEMENT FOR IMA Inc

Using Table below with a Level A software assessment and TSR of 141 indicates that the Level of FAA Involvement (LOFI) should be MEDIUM. There would be some need for National Resource Specialist (NRS) or Technical Standard (TS) support since a new authentication technology (digital signatures) is being used. For this project, the Aircraft Certification Office (ACO) may elect to perform one on-site review and some desk reviews, depending on their workload. Much of the data approval could be delegated. However, because it is a level A software project in the system, approval of the software accomplishment summary should be reserved by the ACO.

Total Score Result	SW Level A	Software Level B	Software Level C	Software Level D
TSR <= 80	HIGH	HIGH	MEDIUM	LOW
80 < TSR <= 130	HIGH	MEDIUM	MEDIUM	LOW
130 < TSR	MEDIUM	MEDIUM	LOW	LOW

Acronyms:

ACO Aircraft Certification Office
ASE Aviation Safety Engineer
DER Designated Engineering Representative
FAA Federal Aviation Administration
IMA Integrated Modular Avionics
LOFI Level Of FAA Involvement
NRS National Resource Specialist
SQA Software Quality Assurance
STC Supplemental Type Certificates
TS Technical Standard
TSO Technical Standard Order
TSOA Technical Standard Order Authorization
TSR Total Score Result

4. FAA MEDIUM LEVEL INVOLVEMENT FOR IMA Inc

The following activities constitute the agreement between FAA and IMA, Inc for certification of the new IMA Load Control Software for use with certified IMA Units.

Level of FAA Involvement	**Example of Typical Program Decisions**
MEDIUM	• DER has approval authority of SCI, SDP, SQAP, SCMP • FAA/TSO involvement for planning, reliability demonstration, final compliance meetings; approval authority of PSAC, Verification Plan/Case and SAS • FAA/TSO conducts on-site review as part of initial PSAC and final compliance meetings • FAA/TSO conducts desk reviews of PSAC, Verification Plan/Case, SCI, and SAS • FAA/TSO requires submittal of PSAC, SCI, Verification Plan/Case, SAS

IMA Inc
Plan for Software Aspects of Certification (PSAC)
Life Cycle Activity Matrix

Life Cycle Activity	Verification Activity	Reliability Activity	Claim	Evidence	Rationale	References
Requirements	Formal Inspection	-DRE Major-Severity Level 1,2,3,4 Minor-Severity Level 5 -Traceability Analysis	-DRE will be at least 70% based on defects found and root cause sources throughout life cycle -Defects/page<= 0.5	-#Defects -Defects Source -DRE	-Formal Inspection is recognized as a "best practice"	-Formal Inspection Reports
Design	Formal Inspection	-DRE Major-Severity Level 1,2,3,4 Minor-Severity Level 5 -Traceability Analysis	-DRE will be at least 70% based on defects found and root cause sources throughout life cycle -Defects/page<= 0.5	-#Defects -Defects Source -DRE	Formal Inspection is recognized as a "best practice"	-Formal Inspection Reports
Design	Analysis of potential failure modes	-Software FMEA and FTA	-State chart design and code analysis ensures no software defects exist that could cause Major Failures	-Failure Modes -Fault Trees -Mitigation Approach	SW FMEA and SW FTA support System Analyses	-Detailed FMEA and FTA analyses
Code	Formal Inspection	-DRE Major-Severity Level 1,2,3,4 Minor-Severity Level 5 -Traceability Analysis	-DRE will be at least 70% based on defects found and root cause sources throughout life cycle -Defects/KSLOC<=7	-#Defects -Defects Source -DRE	Formal Inspection is recognized as a "best practice"	-Formal Inspection Reports
Unit Test	Test to low level requirements	-Requirement Coverage -Feature Coverage -Path Coverage -Statement Coverage	-100% -100% -100% (logic path) -100%	-#Defects -Actual Coverage Metrics -Unit test results	-Coverage testing is considered a 'Top Ten best practice for software reliability'	-Development Folders
System/ Integration Test Plan	Formal Inspection	-DRE Major-Severity Level 1,2,3,4 Minor-Severity Level 5 -Traceability Analysis	-DRE will be at least 70% based on defects found and root cause sources throughout life cycle -Defect density = 0.5	-#Defects -Defects Source -DRE	-Formal Inspection is recognized as a 'Top Ten best practice for software reliability'	-Formal Inspection Reports
System/ Integration Testing	Test to High Level Requirements	-Requirement Coverage -Feature Coverage -Operational Profiles	-100% -100% -Defect pKSLOC <= 1.0	-Coverage Metrics -Defect data -Reliability model	-System testing and regression testing when changes are made is	-Formal product definition -

Life Cycle Activity	Verification Activity	Reliability Activity	Claim	Evidence	Rationale	References
		-Reliability Growth	-Reliability >= 0.99 ph h=ex hour ~ 2500 IMU op hrs	results -Test Results	considered a 'Top Ten best practice for software reliability'	Verification Report
General FRACAS	Failure reporting and corrective action system	-Track defects throughout life cycle; root cause analyses; defect density; failure rate	-FRACAS system will reduce rework and provide for process improvement	-Defect identification and corrective action documentation	-System testing and regression testing when changes are made is considered a 'Top Ten best practice for software reliability'	-FRACAS data base
FAA Certification	-FAA Initial Review -FAA Verification Review -FAA Final Compliance Review	-Provide reliability evidence at each review, including reliability demonstration/acceptance at Final Compliance Review acceptance metrics	-All review actions will be resolved prior to delivery of Load Control Software to customer	-Review action items -Final Compliance Review -acceptance metrics -PSAC -Verification Plan/Case -SCI -SAS	-FAA LOFI determined by applying FAA guidance	-FAA Certification Data
NSIA Air Acquisition	-FAA Certification Evidence Review -Key Performance Parameters Evidence Review -Supplier Capabilities Checklist Review	-Same as for FAA Certification -Customer assessment of supplier capabilities/KPP	-Customer independent reviews and KPP assessment provides additional assurance that reliability goals are met.	-Same as for FAA Certification -KPP values -Supplier checklist score	-Independent verification is a best practice and provides customer with specific capability to ensure adequate goals are set and met by the supplier	-FAA Certification Information -IMA project configuration management system -IMA FRACAS

Appendix E

Abstract and Biography

The Procurement of Software Dependent Systems
Making Systems Reliable through Software Reliability Engineering Techniques
Dr David E Peercy, Sandia National Laboratories

This presentation provides an introduction to software reliability with a case study example. The presentation illustrates how one might establish a software reliability program as part of the procurement of software dependent systems. Recently developed Society of Automotive Engineers (SAE) standards are the primary source for the software reliability program concepts. The case study is specific to FAA Aerospace product certification and illustrates hypothetical interactions of customer, supplier, and certification authority with a focus on example software reliability requirements and results.

A software reliability program includes activities across the full system/software life cycle that provide a level of confidence the software will not fail during its operational mission. The goals and objectives of a software reliability program and the key principles of determining, meeting, and demonstrating customer requirements will be discussed. The context for software reliability includes integration with system reliability, design for and operational measurement of reliability, and use of a management planning and case evidence framework.

Dr David E Peercy
Sandia National Laboratories
Biography

Dr. Peercy is a Distinguished Member of the Technical Staff at Sandia National Laboratories responsible for quality engineering of critical software systems. He is lead quality engineer for Use Control software applications, including over 20 Mark Quality software products delivered to DoD customers over the past four years. He is lead quality engineer for the W80-3 Crypto Coded Switch, principal investigator for the ASCI V&V program, core Sandia representative to the Nuclear Weapons Complex Software Quality Assurance Subcommittee, and Chair of the SAE G-11 RMSL Software Committee developing International software supportability and reliability standards. Dr Peercy has software publications in reliability, maintenance, supportability, and process improvement. Dr. Peercy received his Ph.D in Mathematics from New Mexico State University in 1971, is a Certified Software Quality Engineer, and a member of several professional societies: ASQ, IEEE, ACM, AIAA, and SOLE.

Dr David E Peercy
Office: 505-844-7965
Email: depeerc@sandia.gov
Sandia National Laboratories
P.O. Box 5800, MS-0638
Albuquerque, NM 87185-0638

Biography

Dr. Peercy is a Distinguished Technical Staff Member at Sandia National Laboratories (SNL) responsible for quality engineering of critical software systems. He has been the lead quality engineer for Use Control software and Use Control applications that have been developed over the past 10 years. The most recent effort is called the Code Management System (CMS) project that will produce a common family of products and applications over the period 1997 through 2005 that will replace all Use Control equipment in the field. Dr. Peercy is also the lead quality engineer for the Crypto Coded Switch component of the new W80-3 Stockpile Life Extension program. Dr Peercy also provides V&V program consultation for the Advanced Simulation and Computing Program (ASCI) and is the core representative for SNL on the Nuclear Weapons Complex Software Quality Assurance Subcommittee (SQAS).

Dr. Peercy is the chairman of the Society of Automotive Engineers (SAE) G-11SW Software Committee developing International standards and guidelines for software supportability and software reliability. He has taught short courses / tutorials on a variety of software subjects and has numerous publications in software engineering areas such as software reliability, software maintenance, software supportability, and software process improvement.

Dr. Peercy has been a reviewer for numerous standards including the Pascal and ADA language standards, the National Computer Security Center Trusted Network Evaluation Criteria and various IEEE standards. Dr. Peercy received his Ph.D. and Masters in mathematics from New Mexico State University and his Bachelors in Applied Mathematics from the University of Colorado.

Professional Organization Affiliations

- American Society for Quality (ASQ), Certified Software Quality Engineer (CSQE)
- IEEE Computer Society
- Association for Computing Machinery (ACM)
- American Institute of Aeronautics and Astronautics (AIAA)
- International Society of Logistics (SOLE)