

An Integrated Life Cycle-based Software Reliability Assurance Approach for NASA Projects

Presented by

Ying Shi

ManTech International

At

ASQ Baltimore Section 0502 Dinner Meeting

December 8, 2009

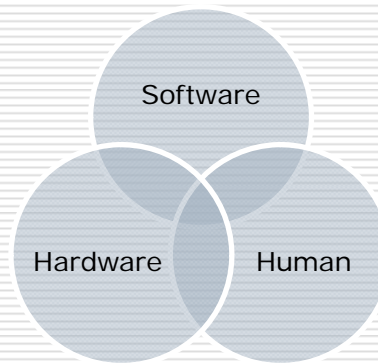
Outline

- Software Reliability (SWR) Introduction
 - *What* is Software Reliability?
 - *Why* do we care about Software Reliability?
 - *What* practices/approaches can we take to achieve optimal Software Reliability?
 - *When* shall we implement these practices/approaches?

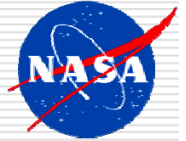
- An Integrated Life Cycle-based Software Reliability Assurance Approach
 - Review existing system reliability requirements and understand operational system dynamics
 - Identify techniques for software reliability improvement
 - Establish a process to guide requirements implementation

System and Software Reliability

- ❑ Reliability of complex systems is essentially determined by the reliability of the hardware systems, software and human reliability.



- ❑ Digital systems and software enable the successful execution of otherwise unachievable space missions. Mission success requires high confidence of success in entities:
 - High fidelity of flight hardware
 - High fidelity of software systems with multiple applications
 - Well understood human interfaces/interactions
 - Well understood hardware/software interactions



Software Reliability Definition*

- **Software Reliability** is the probability that software will not cause a failure for a specified time under specified conditions.

- Software error, fault and failure
 - Software **Error** -- Human action that results in software containing a fault.
 - Software **Fault** -- A defect in the code that can be the cause of one or more failures.
 - Software **Failure** -- A departure of program operation from program requirements

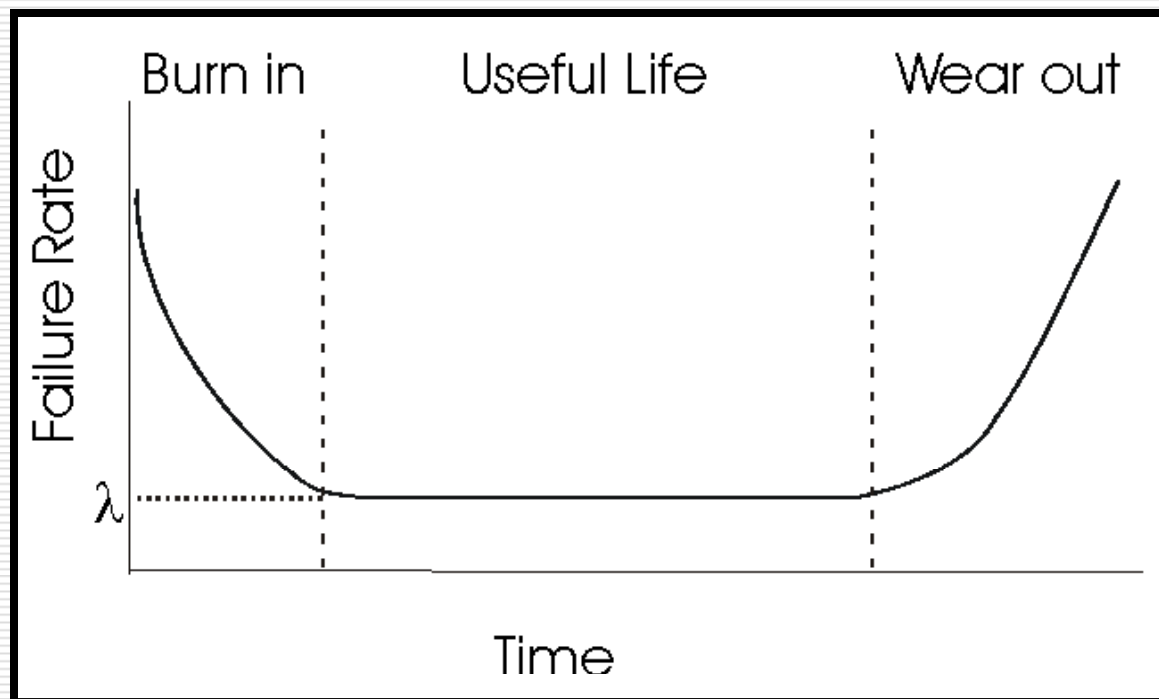
* IEEE Std 1633 – 2008

Why do we care about SWR?

- ❑ Systems are becoming software-intensive and software is becoming more and more complex
- ❑ More reliable software is required since software failures can lead to fatal consequences in safety-critical systems and business/financial systems
- ❑ Software development cost is increasing

Hardware Reliability

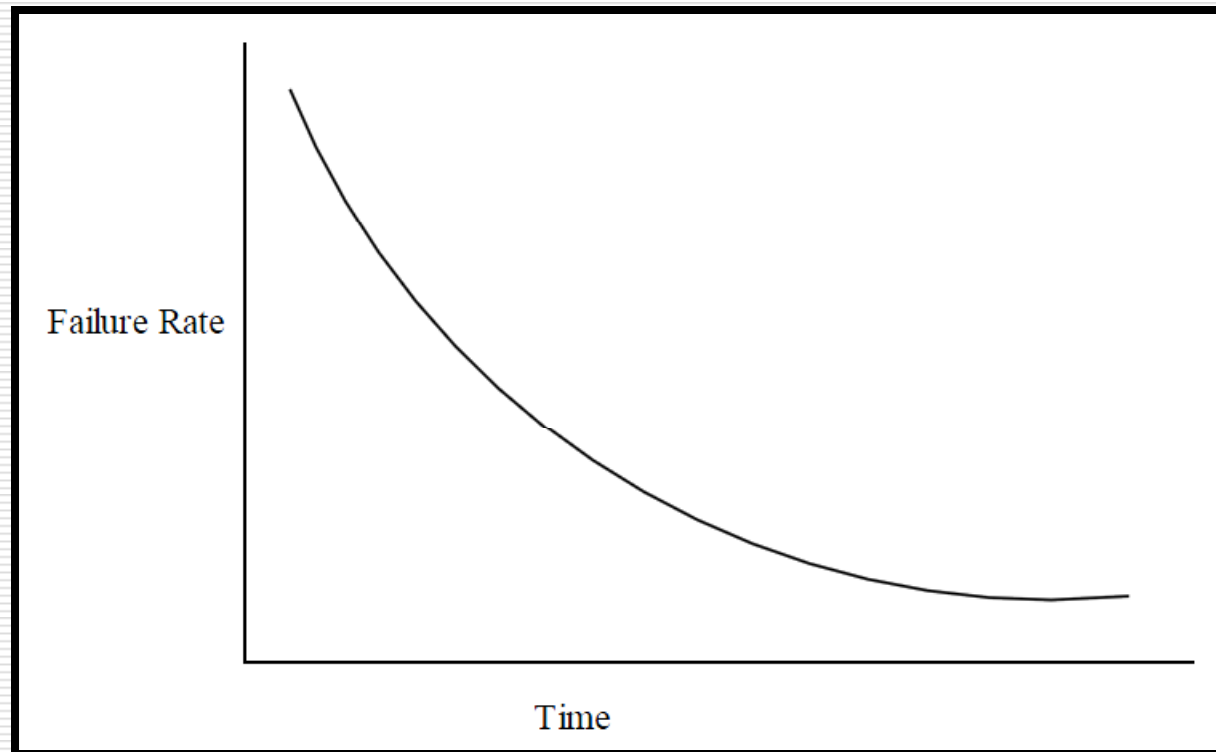
- The bathtub-shaped curve results from the combination of
 - “Infant Mortality” Failures
 - Constant Failures
 - Wear Out Failures



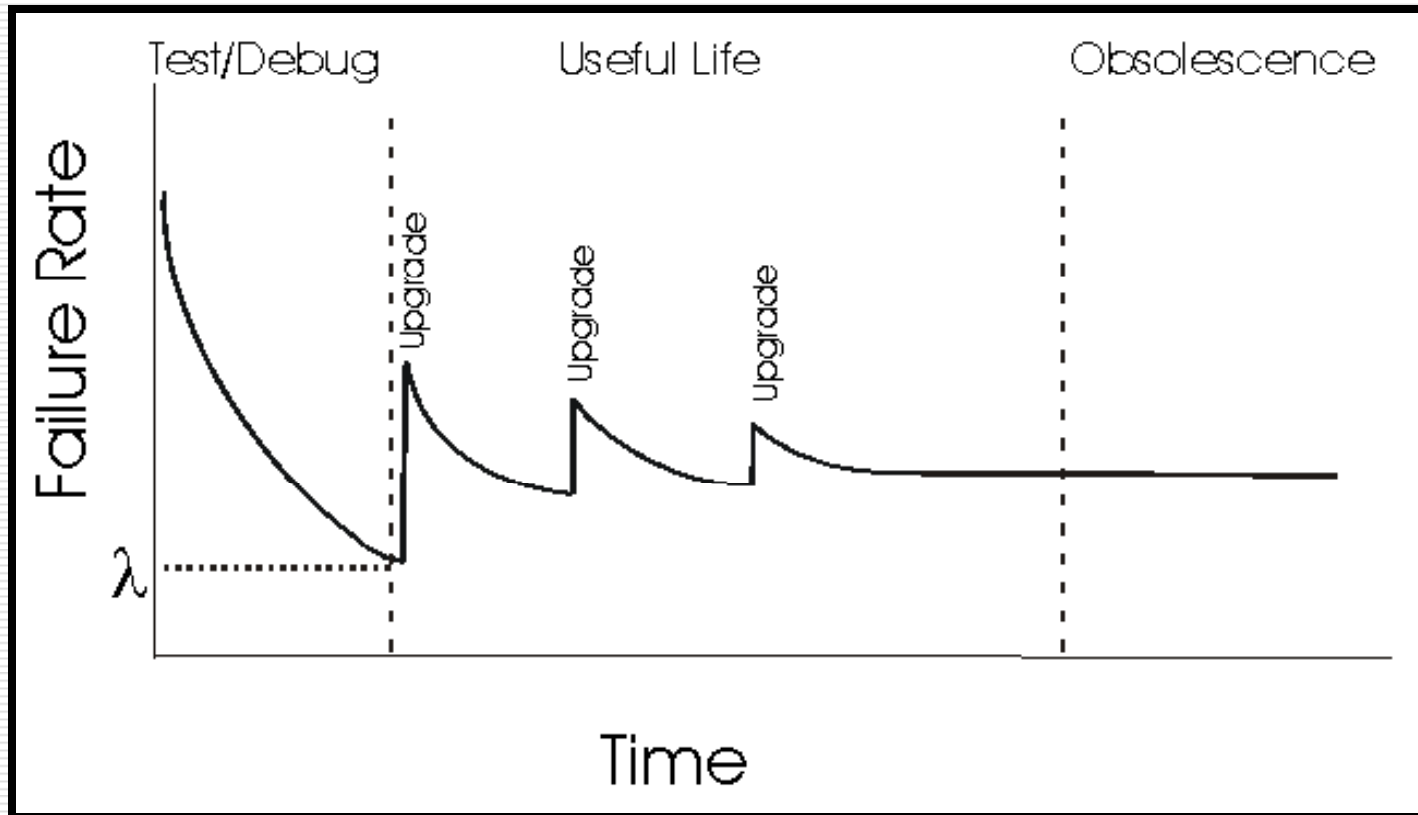
Software VS Hardware

- ❑ Software does not wear out
- ❑ Software may be more complex than hardware
- ❑ Failure mechanisms for hardware and software are different
- ❑ Redundancy and fault tolerance for hardware are common practices; these concepts are only beginning to be practiced in software
- ❑ Changes to hardware require a series of important and time-consuming steps; changes to software is frequently more feasible
- ❑ Repair generally restores hardware to its previous state; software repair always changes the software to a new state and could introduce new defects to software
- ❑ Hardware reliability is expressed in calendar time; software reliability may be expressed in execution or calendar time

Software Failure Rate



Software Failure Rate (cont.)



Quantitative SWR Approach

Procedures

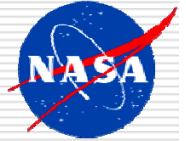
1. Develop a software reliability allocation plan and a software reliability growth plan from system's perspective for critical software functions;
2. Document, monitor, analyze and track software defects assessed during testing/operational performance for each stage of development and across development and operational phases;
3. Assess the reliability of software products produced by each process of the life cycle through software reliability measurements or software reliability models;
4. Conduct periodic verifications (e.g. at each NASA project key decision point) of whether the reliability growth target has been met;
5. Provide corrective actions for software subsystems/modules which could not achieve the reliability growth target.

Understand Software Reliability

- From Quantitative Assessment to Qualitative Management
 - **Software Reliability (SWR)** is a subset of SWRM and is (quantitatively) defined as the probability that software will not cause the failure of a system for a specified time under specified conditions.
 - **Software Reliability Management (SWRM)** is (qualitatively and quantitatively) the process of optimizing the reliability of software through a program that emphasizes software error prevention, fault detection and removal, and the use of measurements to maximize reliability (software reliability growth) in light of project constraints such as resources, schedule and performance.

Qualitative SWR Approach

1. Conduct software reliability trade-off studies when comparing different system/subsystem/module design architectures;
2. Perform software hazard analysis to ensure the success of software-hardware interaction or software-human interaction;
3. Perform software failure modes and effects (SFMEA) analysis starting with safety-critical functions;
4. Incorporate other critical factors to system-level risk identification. Critical factors include known concerns or weaknesses from re-use of software elements, fault tolerance structures and, hardware operational conditions;
5. Address the level and manner of fault and failure detection, isolation, fault tolerance, and recovery expected to be fulfilled by the software, as part of the overall system.
6. Track the compliance with development standards, e.g. standard code development, walk through, modularity etc.



Software FMEA

□ Background

- Software FMEA was introduced in the literature as early as 1983
- Software FMEA has been applied to safety critical real-time control systems embedded in military and automotive products over the last decade

□ Approach

- Inductive (“bottom up”) technique for identifying how each component could fail and its impact on subsystem/system operations.
- Identify software faults that can lead to system/subsystem failure.

SFMEA Procedure

- A Software FMEA uses the methods of a hardware FMEA, substituting software components for hardware components.
- A widely used FMEA procedure is MIL-STD-1629, which is based on the following steps:
 1. Define the system to be analyzed.
 2. Construct functional block diagrams.
 3. Identify all potential item and interface failure modes.
 4. Evaluate each failure mode in terms of the worst potential consequences.
 5. Identify failure detection methods and compensating provisions.
 6. Identify corrective design or other actions to eliminate / control failure.
 7. Identify impacts of the corrective change.
 8. Document the analysis and summarize the problems which could not be corrected.

Levels of SFMEA

- High Level (System Level) SFMEA
 - Assess the ability of the software architecture to provide protection from the effects of software and hardware failures
 - Software elements are treated as black boxes
 - Possible failure modes:
 - Fails to execute
 - Executes incompletely
 - Incorrect Output
 - Incorrect timing (too early, too late etc)

Levels of SFMEA (Cont.)

- Detailed Level (Component Level) SFMEA
 - Used to validate that software design achieves the requirements
 - Is similar to component level hardware FMEA
 - Possible Failure Modes:
 - Component:
 - Missing data
 - Incorrect data
 - Timing data
 - Extra data
 - Process:
 - Missing event
 - Incorrect logic/algorithm
 - Abnormal logic
 - Timing issue

PROs and CONs

□ PROs

- Help find hidden failure modes, system interactions, and dependencies
- Help identify inconsistencies between the requirements and the design

□ CONs

- Time consuming
- Expensive
- Manual approach
- Need expertise

Identify Safety-Critical Software

- Safety-critical software includes hazardous software (which can directly contribute to, or control a hazard). It also includes all software that can influence that hazardous software.

- In summary, software is safety-critical if it performs any of the following:
 - Controls hazardous or safety-critical hardware or software.
 - Monitors safety-critical hardware or software as part of a hazard control.
 - Provides information upon which a safety-related decision is made.
 - Performs analysis that impacts automatic or manual hazardous operations.
 - Verifies hardware or software hazard controls.
 - Can prevent safety-critical hardware or software from functioning properly

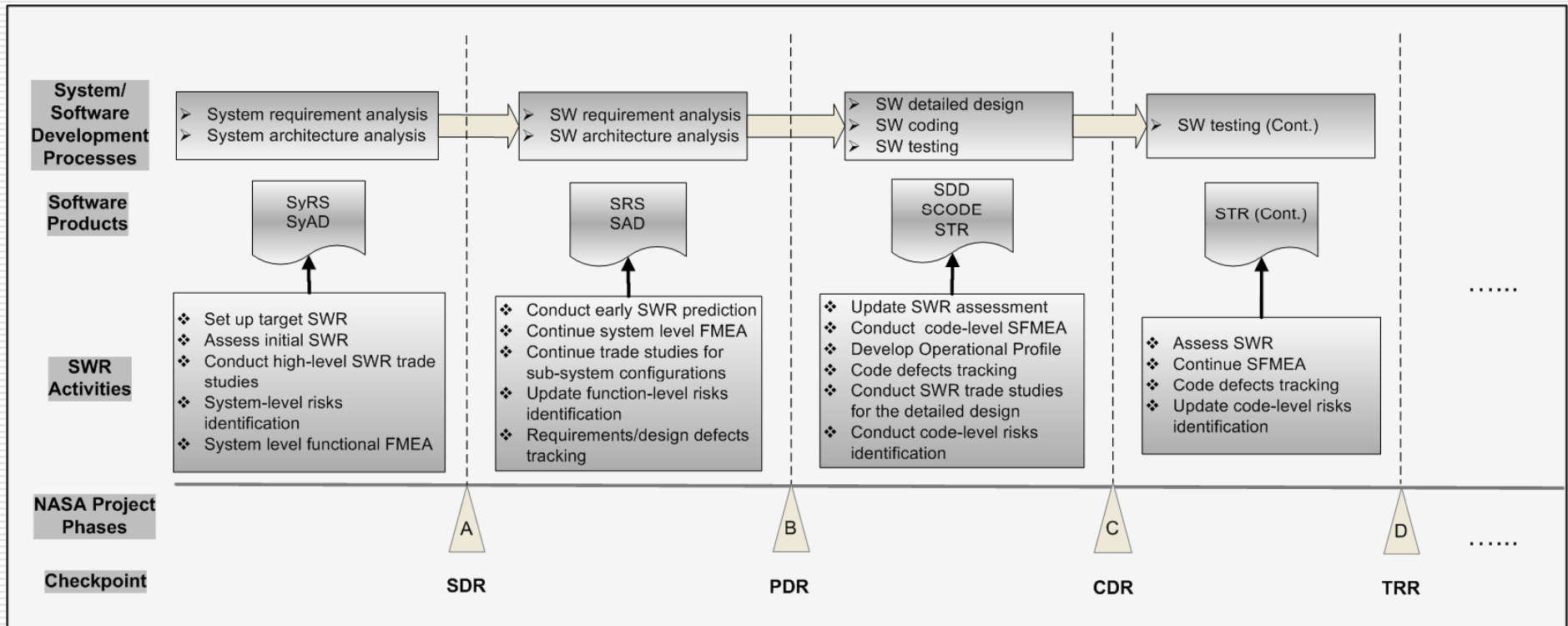
Risk Score Card

- Risk Score Card ---- A 4C evaluation system:
 - Classification
 - Complex-electronics
 - Composition
 - Characteristics

□ Example:

CSCI	Description	Classification	Complex-Electronics	Composition					Characteristics		
				% custom	% COTS	% heritage	% legacy	% reuse	Lines of Code (Function point)	Defect Density	Cyclomatic Complexity
Common Services and Framework (CMS)	Common Services and Framework is safety critical. It processes safety-critical commands and data, and resides on processors that have safety-critical software. It also processes data that leads to safety decisions.	Class A	☐	☐	☐	☐	☐	☐	☐	☐	☐
Remote Distribution Services (ADVS)	Remote Distribution Services are not safety-critical. They do not reside on a safety-critical system, do not process data used in safety decisions, nor do they provide any verification or validation of safety-critical systems.	Class C	☐	☐	☐	☐	☐	☐	☐	☐	☐

An overview



Phase A - Concept and Technology Development

- Phase A: Mission concepts and program requirements on the project are established; functions and requirements are allocated to particular items of hardware, software and personnel. (System requirements analysis and system architecture design)

- Typical software products delivered at SDR include system requirements document and system architecture document.

- SWR Activities:
 - Software reliability allocation plan
 - Initial software reliability assessment
 - System level trade studies for different system configurations
 - System level software functional FMEA starting with critical software functions
 - System level risk identification

Phase B - Preliminary Design and Technology Completion

- Phase B: establish a functionally complete preliminary design solution that meets mission goals and objectives. (software requirements analysis and software architecture design phase.)

- Typical software products delivered at PDR include software requirements specifications and software architecture design

- SWR Activities:
 - Update software reliability assessment
 - Continue system level Software FMEA based on SRS, SDD and/or UML model
 - Continue trade studies for different software sub-system configurations
 - Update risk identification
 - Requirements/design defects detection

Phase C - Final Design and Fabrication

- Phase C: establish a complete design, fabricate or produce hardware, and develop the software code in preparation for integration. (software detailed design, software coding and software testing (unit test) phase.)

- Typical software products delivered at CDR include software detailed design, software code and software unit test results.

- SWR activities:
 - Continue updating software reliability
 - Conduct code level SFMEA
 - Develop Operational Profile based on operation scenarios
 - Code defects tracking
 - Conduct SWR trade studies for the detailed design
 - Conduct code-level risks identification

Phase D - System Assembly, Integration, Test & Launch

- Phase D: activities are performed to assemble, integrate, test, and launch the system. (software testing phase in the software development process.)

- The typical software product delivered at TRR is software testing results based on functional testing.

- SWR activities:
 - Assess SWR using actual testing failure data
 - Continue SFMEA
 - Code defects tracking
 - Update code-level risks identification

Summary & Future Work

- ❑ The proposed process will help proactively integrate collaborative arrangement with design engineering, FDIR (Diagnostics & Prognostics) and software assurance.

- ❑ The proposed life-cycle based approach will help identify key processes in each major milestone.

- ❑ More focused efforts on key risk drivers that could inhibit the mission success and resolving them.

- ❑ Future work will focus on the application of the proposed approach to ongoing NASA projects.