ELISA
Enabling Linux in Safety Applications

WORKSHOP

NASA Goddard

Verification and Validation of the OS for NASA

Scott B. Tashakkor, P.E., NASA/MSFC
NASA Deputy Tech Fellow for Software

Outline

- Introduction
- NPR 7150.2D NASA’s Software Engineering Requirements
- Safety Critical Definition and Requirements
- Why NASA’s Requirements Apply
- What Else Is Needed

ELISA Enabling Linux in Safety Applications
WORKSHOP

Introduction

ELISA Enabling Linux in Safety Applications WORKSHOP

Scope

- NASA's Software Engineering Requirements at Agency Level
  - Centers/Programs/Projects can levy more requirements on their own
  - Technical Authorities and Risk Acceptance
- Focus on the Linux Kernel
  - Some GNU programs
- Criticality/Tools
- Specific needs information spread throughout discussion
  - No specific section
- Human Rating Requirements are not addressed

Software Is Not All the Same

flight software	≠	*Non-flight software*
engineering software	≠	*general purpose software*
safety critical software	≠	*non-safety critical software*

… and it shouldn't be treated the same!

NPR 7150.2 D
NASA Software Engineering Requirements

ELISA Enabling Linux in Safety Applications WORKSHOP

NASA-wide software classification structure

These definitions are based on:
(1) usage of the software with or within a NASA system,
(2) criticality of the system to NASA's major programs and projects,
(3) extent to which humans depend upon the system,
(4) developmental and operational complexity, and
(5) extent of the Agency's investment.

NASA-Wide Software Classifications

Class A	**Human-Rated Space Software Systems**
Class B	**Non-Human Space-Rated Software Systems or Large-Scale Aeronautics Vehicles**
Class C	**Mission Support Software or Aeronautic Vehicles, or Major Engineering/Research Facility Software**
Class D	**Basic Science/Engineering Design and Research and Technology Software**
Class E	**Design Concept, Research, Technology and General Purpose Software**
Class F	**General Purpose Computing, Business and IT Software**

Notes: It is not uncommon for a project to contain multiple systems and subsystems having different software classes.

Visual Overview of NPR 7150.2

30 "Institutional" Requirements (Chapter 2) Applicable to All Classifications

ISWE

OCE	SMA	Center Director/Delegate(s)		
Lead Software Engineering Initiave	Lead Software Assurance and Safety Initiative	Staff and advance software engineering capability	Measure for Improvement	Maintains contributor list
Benchmark Center's Capabilities against this NPR	Benchmark Center's SWA and SW Safety Capabilities	Establish and execute software processes	Establish and maintain software cost repo	Ensure Proper transfer of software
Benchmark Center Mapping Matrices	Review Center's Mapping Matrices	Comply with NPR per Classification in Appendix C	Contribute to Agency PAL (Process Asset Library)	Contract Officer: Ensure NPR is on contract
Authorize Compliance Appraisals	Authorize Appriasals against requirements	Report project status	Define content of SW documentation	Tech Authority: Assess against NPR
Provide Software Engineering Training	Provide Software Assurance Training	Maintain list of projects	Ensure Government rights to Software	OCE, SMA, OCIO: agree on tailoring
Maintain Process Asset Library (PAL)	Makes Decisions on Tailoring IVV Rqmt	Establish and maintain software Metrics	Ensure reuse software conforms to policies	Project Manager: Update plans per Classification

100 NPR Requirements* - *Applicable Based on Classification*

ISWE

Software Management (Chapter 3) | Lifecycle (Chapter 4) | Lifecycle Support-Ch5

Make/Buy	Tailor	Classify	Perform MC/DC	Verify Cyber Protection	Validate	Accredit Tools	Regression Test	Track Changes	Record Peer Review Results
Plan	Mapping to this NPR	Maintain Classification Records	Track Cyclomatic Complexity **	Use Secure Coding	Architect	Plan, Report Tests	Test Safety Rqmts	Identify CM Items	Measure Software
Track Actual vs. Expected Plan	Establish and Acquire OTS	Plan SA & IVV	Plan Auto-Gen lifecycle	Use Cyber Static Analysis	Review Architecture	Test	Develop, Test Data Upload Procedures	Establish CM Procedures	Analyze Software Measurements
Determine Acceptance Criteria	Establish Cost	Ensure IVV	Receive Auto-Gen Supplier Inputs	Record Adversarial Actions	Design	Manage Configuration	Test Reuse/COTS Equally	Maintain CM Records	House Measurement Data
Determine Deliverables	Include Specific Cost Items	Ensure IVV Project Exec Plan (IPEP) if IVV	Perform and Certify as CMMI	Perform Bi-Directional Traceability	Implement, Code	Evaluate Test Results	Plan Ops, Maintenance, Retirement	Perform CM Audits	Compare Measured vs. Expected
Define Milestones	Store Cost in Repo	Provide IVV Artifacts	Identify Reuse Rqmts	Establish Rqmts	Adhere to Coding Standards	Use Accredited Tools	Deliver Products	Develop Release Procedures	Measure Software Volatility
Developer Report Status	Develop Schedule	Respond to IVV Findings	Evaluate Reusability	Map to System Rqmts	Perform Static Code Analysis	Update Plans	Complete Verification	Participate in Audits	Track Defects
Dev'er Provide Product & Metrics	Regularly Review with Stakeholders	Determine Safety Criticality	Assess Cyber	Include Safety Rqmts	Unit Test	Validate in High-fidelity	Maintain	Determine, Manage Risk	Determine Severity Levels
Developer to Provide Access to Source Code	Dev'ers Report Schedule	Adhere to 8739.8 SWA & SW Safety Std	Identify Cyber Risks	Track Rqmt Changes	Repeat Unit Test	Track Code Coverage Metrics	Archive	Peer Review Rqmts, Plans, Code, Test	Assess reuse, COTS defects
Comply with this NPR	Train	Do Safety-Crit items: SWE-134	Implement Cyber Protection	Track Corrective Actions	Develop VDD	Validate Metrics in Test	Plan CM	Follow Basic Peer Review Process	Assess Process Defects

*Note SWE-220 Cyclomatic Complexity has 2 shalls, counted as 1 here (**), Safety Critical in Red

Class A&B (All 101) C (93) D (65) E (12) Requirements (sc)

ISWE

Make/Buy	Tailor	Classify	Perform MC/DC	Verify Cyber Protection	Validate	Accredit Tool	Regression Test	Track Changes	Record Peer Review Results
Plan	Mapping to this NPR	Maintain Classification Records	Track Cyclomatic Complexity **	Use Secure Coding	Architect	Plan, Report Test	Test Safety Rqmts	Identify CM Items	Measure Software
Track Actual vs. Expected Plan	Establish and Acquire OTS	Plan SA & IVV	Plan Auto-Gen lifecycle	Use Cyber Static Analysis	Review Architecture	Test	Develop, Test Data Upload Procedures	Establish CM Procedures	Analyze Software Measurements
Determine Acceptance Criteria	Establish Cost	Ensure IVV	Receive Auto-Gen Supplier Inputs	Record Adversarial Actions	Design	Manage Configuration	Test Reuse/COTS Equally	Maintain CM Records	House Measurement Data
Determine Deliverables	Include Specific Cost Items	Ensure IVV Project Exec Plan (IPEP) if IVV	Perform and Certify to CMMI	Perform Bi-Directional Traceability	Implement, Code	Evaluate Test Results	Plan Ops Maintenance, Retirement	Perform CM Audits	Compare Measured vs. Expected
Define Milestones	Store Cost in Repo	Provide IVV Artifacts	Identify Reuse Rqmts	Establish Rqmts	Adhere to Coding Standards	Use Accredited Tools	Deliver Products	Develop Release Procedures	Measure Software Volatility
Developer Report Status	Develop Schedule	Respond to IVV Findings	Evaluate Reusability	Map to System Rqmts	Perform Static Code Analysis	Update Plans	Complete Verification	Participate in Audits	Track Defects
Developer Provide Product & Metrics	Regularly Review with Stakeholders	Determine Safety Criticality	Assess Cyber	Include Safety Rqmts	Unit Test	Validate in High-fidelity	Maintain	Determine Manage Risk	Determine Severity Levels
Developer to Provide Access to Source Code	Developers Report Schedule	Adhere to 8739.8 SWA & SW Safety Std	Identify Cyber Risks	Track Rqmt Changes	Repeat Unit Test	Track Code Coverage Metric	Archive	Peer Review Rqmts, Plans, Code, Test	Assess reuse, COTS Defects
Comply with this NPR	Train	Do Safety-Critical items: SWE-134	Implement Cyber Protection	Track Corrective Actions	Develop VDD	Validate Metrics in Test	Plan CM	Follow Basic Peer Review Process	Assess Process Defects

Class F Requirement Applicability (OCIO Authority)

ISWE

Make/Buy	Tailor	Classify	~~Perform MC/DC~~	Verify Cyber Protection	Validate	~~Accredit Tools~~	Regression Test	Track Changes	Record Peer Review Results
Plan	Mapping to this NPR	Maintain Classification Records	~~Track Cyclomatic Complexity **~~	~~Use Secure Coding~~	~~Architect~~	Plan, Report Tests	~~Test Safety Rqmts~~	Identify CM Items	~~Measure Software~~
Track Actual vs. Expected Plan	Establish and Acquire OTS	~~Plan SA & IV&V~~	Plan Auto-Gen lifecycle	Use Cyber Static Analysis	~~Review Architecture~~	Test	Develop, Test Data Upload Procedures	Establish CM Procedures	~~Analyze Software Measurements~~
Determine Acceptance Criteria	Establish Cost	~~Ensure IV&V~~	Receive Auto-Gen Supplier Inputs	Record Adversarial Actions	~~Design~~	Manage Configuration	~~Test Reuse/COTS Equally~~	Maintain CM Records	~~House Measurement Data~~
Determine Deliverables	Include Specific Cost Items	~~Ensure IV&V Project Exec Plan (IPEP) if IVV~~	~~Perform and Certify to CMMI~~	Perform Bi-Directional Traceability	Implement, Code	Evaluate Test Results	Plan Ops, Maintenance, Retirement	Perform CM Audits	~~Compare Measured vs. Expected~~
Define Milestones	~~Store Cost in Repo~~	~~Provide IV&V Artifacts~~	Identify Reuse Rqmts	Establish Rqmts	Adhere to Coding Standards	~~Use Accredited Tools~~	Deliver Products	Develop Release Procedures	~~Measure Software Volatility~~
Developer Report Status	Develop Schedule	~~Respond to IV&V Findings~~	Evaluate Reusability	~~Map to System Rqmts~~	Perform Static Code Analysis	Update Plans	Complete Verification	Participate in Audits	Track Defects
Dev'er Provide Product & Metrics	Regularly Review with Stakeholders	~~Determine Safety Criticality~~	Assess Cyber	~~Include Safety Rqmts~~	Unit Test	~~Validate in High fidelity~~	Maintain	Determine, Manage Risk	Determine Severity Levels
Developer to Provide Access to Source Code	Dev'ers Report Schedule	~~Adhere to 8739.8 SWA & SW Safety Std~~	~~Identify Cyber Risks~~	Track Rqmt Changes	Repeat Unit Test	~~Track Code Coverage Metrics~~	Archive	Peer Review Rqmts, Plans, Code, Test	~~Assess reuse, COTS defects~~
Comply with this NPR	Train	~~Do Safety Critical items SWE-134~~	~~Implement Cyber Protection~~	Track Corrective Actions	Develop VDD	~~Validate Metrics in Test~~	Plan CM	~~Follow Basic Peer Review Process~~	~~Assess Process Defects~~

Safety Criticality

What is NASA's Safety Critical Software?

- **NASA definition from NASA-STD-8739.8 B:**
  - Software is classified as safety-critical if it meets at least one of the following criteria:
    - a. Causes or contributes to a system hazardous condition/event,
    - b. Provides control or mitigation for a system hazardous condition/event,
    - c. Controls safety-critical functions,
    - d. Mitigates damage if a hazardous condition/event occurs,
    - e. Detects, reports, and takes corrective action if the system reaches a potentially hazardous state.
  - *Note: Software is classified as safety-critical if the software is determined by and traceable to hazard analysis. See appendix A for guidelines associated with addressing software in hazard definitions. See SWE-205. Consideration for other independent means of protection (software, hardware, barriers, or administrative) should be a part of the system hazard definition process.*

ELISA Enabling Linux in Safety Applications

WORKSHOP

Safety Standard

- 3.7.2 If a project has safety-critical software, the project manager shall implement the safety-critical software requirements contained in NASA-STD-8739.8. [SWE-023]

1. **Confirm that the NPR 7150.2 requirement (SWE-134) items "a" through "l" are documented in the detailed software requirements.**
2. **Assessment that the source code satisfies the conditions in the NPR 7150.2 requirement (SWE-134) "a" through "l" for safety-critical software.**

Safety Critical Software Design (1/2)

- 3.7.3 If a project has safety-critical software or mission-critical software, the project manager shall implement the following items in the software: [SWE-134]

  a. The software is initialized, at first start and restarts, to a known safe state.

  b. The software safely transitions between all predefined known states.

  c. Termination performed by software functions is performed to a known safe state.

  d. Operator overrides of software functions require at least two independent actions by an operator.

  e. Software rejects commands received out of sequence when execution of those commands out of sequence can cause a hazard.

  f. The software detects inadvertent memory modification and recovers to a known safe state.

(SWE-134 continued) …

ELISA Enabling Linux in Safety Applications
WORKSHOP

Safety Critical Software Design (2/2)

(SWE-134 continued)

g. The software performs integrity checks on inputs and outputs to/from the software system.

h. The software performs prerequisite checks prior to the execution of safety-critical software commands.

i. No single software event or action is allowed to initiate an identified hazard.

j. The software responds to an off-nominal condition within the time needed to prevent a hazardous event.

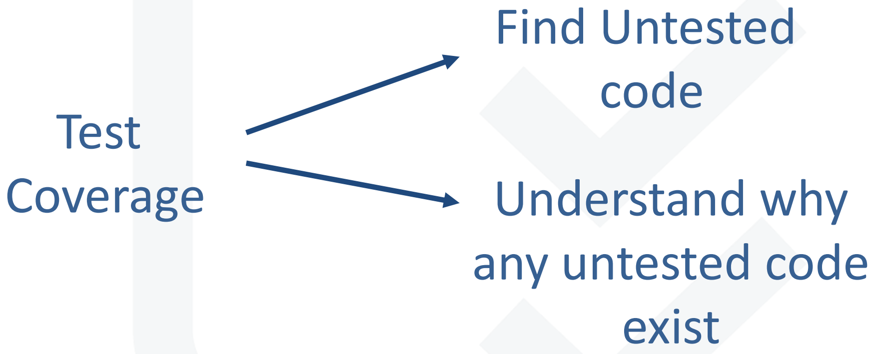
k. The software provides error handling.

l. The software can place the system into a safe state.

Modified Condition/Decision Coverage

- 3.7.4 If a project has safety-critical software, the project manager shall ensure that there is 100 percent code test coverage using the Modified Condition/Decision Coverage (MC/DC) criterion for all identified safety-critical software components. [SWE-219]
    - *Note: In MC/DC coverage, every condition in a decision is tested independently to reach full coverage. Each condition will be executed twice, once with the results true and once with the results of false, but with no difference in the truth values of all other conditions in the decision. In addition, it will be shown that each condition independently affects the decision. Any deviations from 100 percent should be reviewed and waived with rationale by the TAs approval. It is recommended that someone independent of the developer of the code under test design and perform this testing to ensure requirement interpretation or incorrect assumptions do not escape this testing.*

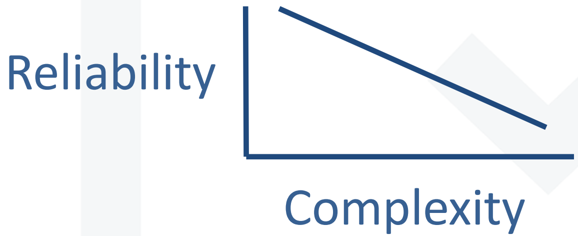
$$\text{Test Coverage} = \frac{\text{Number of Lines of Code Called by Test Suite}}{\text{Total Number of Relevant Lines of Code}} \times 100\%$$



ELISA Enabling Linux in Safety Applications WORKSHOP

Cyclomatic Complexity

- 3.7.5 If a project has safety-critical software, the project manager shall ensure all identified safety-critical software components have a cyclomatic complexity value of 15 or lower. Any exceedance shall be reviewed and waived with rationale by the project manager or technical approval authority. [SWE-220]
    - *Note: Cyclomatic complexity is a metric used to measure the complexity of a software program. This metric measures independent paths through the source code. The point of the requirement is to minimize risk, minimize testing, and increase reliability associated with safety-critical software code components, thus reducing the chance of software failure during a hazardous event.*



ELISA Enabling Linux in Safety Applications

WORKSHOP

Why does this apply?

Requirement on Off-the-shelf and Open-Source Software

- 4.5.14 The project manager shall ***test embedded COTS, GOTS, MOTS, OSS, or reused software components to the same level required to accept a custom developed software component for its intended use.*** [SWE-211]

- Objective evidence is needed to show that the software works for **this** application
  - Ariane 5 is an example of this not being done
  - Users do not know all the assumptions and failure methods of the software
- This requirement does **NOT** mean that users must write unit tests on their own.
  - Package creators may have unit and functional tests

Establishing Requirements

- 3.11.2 The project manager **shall perform a software cybersecurity assessment on the software components per the Agency security policies and the project requirements, including risks posed by the use of COTS, GOTS, MOTS, OSS, or reused software components. [SWE-156]**
- 4.1.2 The project manager shall **establish, capture, record, approve, and maintain software requirements**, including requirements for COTS, GOTS, MOTS, OSS or reused software components, as part of the technical specification. [SWE-050]
- 4.5.3 The project manager shall **test the software against its requirements*.*** [SWE-066]
  - Note: A best practice for Class A, B, and C software projects is to have formal software testing planned, conducted, witnessed, and approved by an independent organization outside of the development team.
- 4.5.6 The project manager shall ***use validated and accredited software models, simulations, and analysis tools*** required to perform qualification of flight software or flight equipment. [SWE-070]
  - Note: Information regarding specific verification, validation and credibility techniques and the analysis of models and simulations can be found in NASA-STD-7009 and NASA-HDBK-7009.

Testing Requirements

- 4.5.10 The project manager shall ***verify code coverage is measured*** by analysis of the results of the execution of tests. [SWE-190]
    - Note: If it can be justified that the required percentage cannot be achieved by test execution, the analysis, inspection or review of design can be applied to the non-covered code. The goal of the complementary analysis is to assess that the non-covered code behavior is as expected.
- 4.5.11 The project manager shall plan and ***conduct software regression testing*** to demonstrate that defects have not been introduced into previously integrated or tested software and have not produced a security vulnerability. [SWE-191]
- 4.5.12 The project manager shall ***verify through test the software requirements that trace to a hazardous event, cause or mitigation technique.*** [SWE-192]
- 4.5.13 The project manager shall ***develop acceptance tests for loaded or uplinked data, rules, and code that affects software and software system behavior.*** [SWE-193]
    - Note: These acceptance tests should validate and verify the data, rules, and code for nominal and off-nominal scenarios.

What else is needed?

Determinism and Protections

- Determinism will be critical for safety applications
  - Linux is an operating system that runs on unknown hardware
  - Linux must provide information or a way for users to determine a guaranteed response time
    - Required response time is defined by user
- Memory, scheduler, and resource protections
  - Linux needs to provide isolation and protections between systems
  - Prevent inadvertent memory modifications or cache protections
    - Examples: mprotect, cpuset, irqaffinity
- Robustness

Support

- For safety applications teams often need/want support with defined response times
  - Teams will often pay for this type of support model
    - Example, if there is a bug or feature needed, how is this added
  - Models from some vendors are already in place to support this
  - Trade off with cost, some teams want the no-cost solution (or work with what they know already)

ELISA Enabling Linux in Safety Applications WORKSHOP

Questions?