Approaches on Assessing Safe Usage of Linux

Kate Stewart,
VP Dependable Embedded System
The Linux Foundation
kstewart@linuxfoundation.org

Who am I?

VP Dependable Embedded Systems, The Linux Foundation: 2015 →

- Real Time Linux: 2015 → 2024
- Zephyr Project: 2016 →
- CHAOSS: 2017 →
- ELISA Project: 2018 →

Volunteer:

- SPDX: 2009 →
- Linux Plumbers Committee: 2016 →
- NTIA SBOM Formats & Tooling co-lead 2018 → 2021
- DHS CISA SBOM Tooling & Implementation WG co-lead 2022 → 2024
- OpenSSF SBOM Everywhere SIG co-lead 2022 →

Standards:

- ISO/IEC 5962:2021 (SPDX)

Publications:

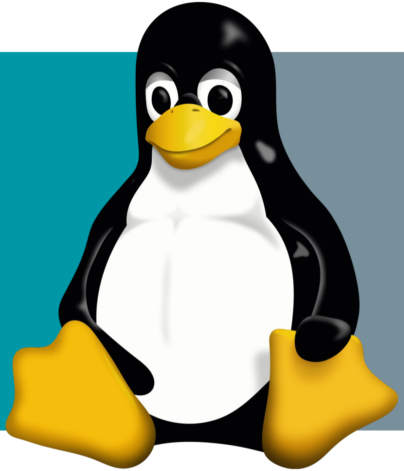
- cregit(Empirical Software Engineering, 2019)
- Linux Kernel History Report (Linux Foundation, 2020)
- SPDX and SBOM (Open Source Law, Policy & Practice, 2022)
- AI BOM (LF Research, 2024)

Other: amateur photographer & world traveler

Contact: kstewart@linuxfoundation.org

ELISA
Enabling Linux in Safety Applications

Linux is 35 years old!



The Linux kernel is the single most important open source project in the world.

It powers supercomputers, stock exchanges, nuclear submarines, Starlink, cars, and much, much, more!

ELISA
Enabling Linux in
Safety Applications

Where is Linux Being Used Today?

100%
Supercomputer Market

82%
Smartphone Market Share

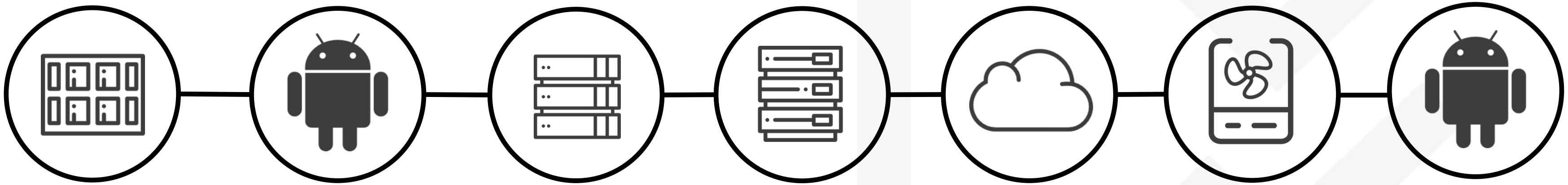
2nd
To Windows in Enterprise

90%
Mainframe Customers

90%
Public Cloud Workload

69%
Embedded Systems Market

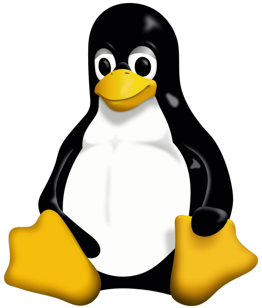
#1
Internet Client



Every market Linux has entered it eventually dominated

ELISA
Enabling Linux in Safety Applications

Focus on Most Used Embedded Components



69%

- Embedded Linux
- Ubuntu
- Android
- Debian (Linux)
- Red Hat (Linux)
- Wind River (Linux)

Source:
https://www.embedded.com/wp-content/uploads/2019/11/EETimes_Embedded_2019_Embedded_Markets_Study.pdf

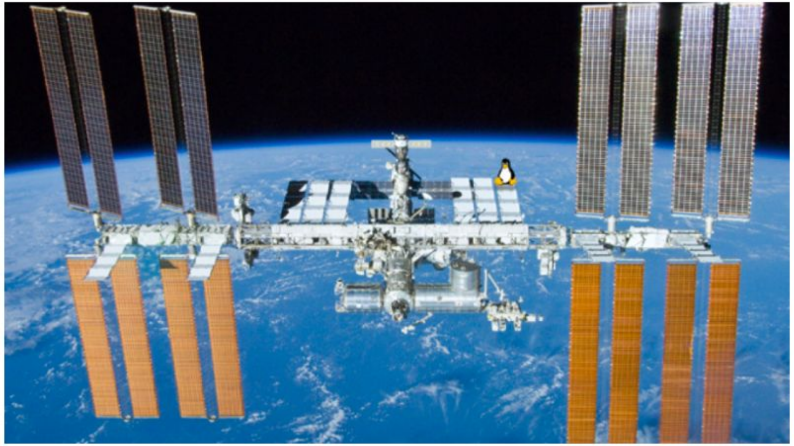
ELISA
Enabling Linux in Safety Applications

Home > Extreme

International Space Station switches from Windows to Linux, for improved reliability

The United Space Alliance, which manages the computers aboard the International Space Station in association with NASA, has announced that the Windows XP computers aboard the ISS have been switched to Linux. "We migrated key functions from Windows to Linux because we needed an operating system that was stable and reliable."

By Sebastian Anthony May 9, 2013

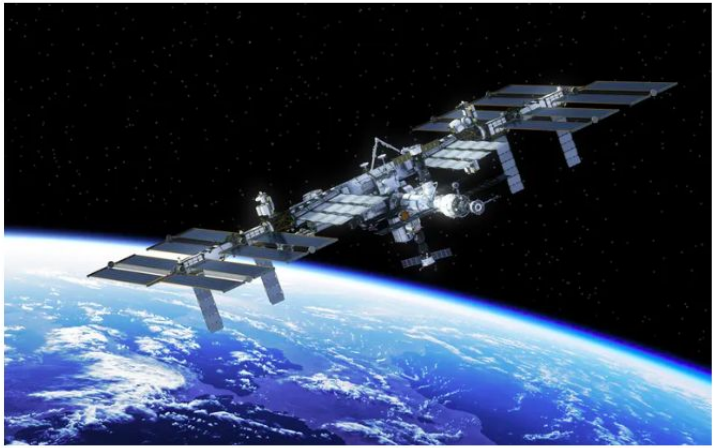


Source:https://www.extremetech.com/extreme/155392-international-space-station-switches-from-windows-to-linux-for-improved-reliability

The ISS just got its own Linux supercomputer

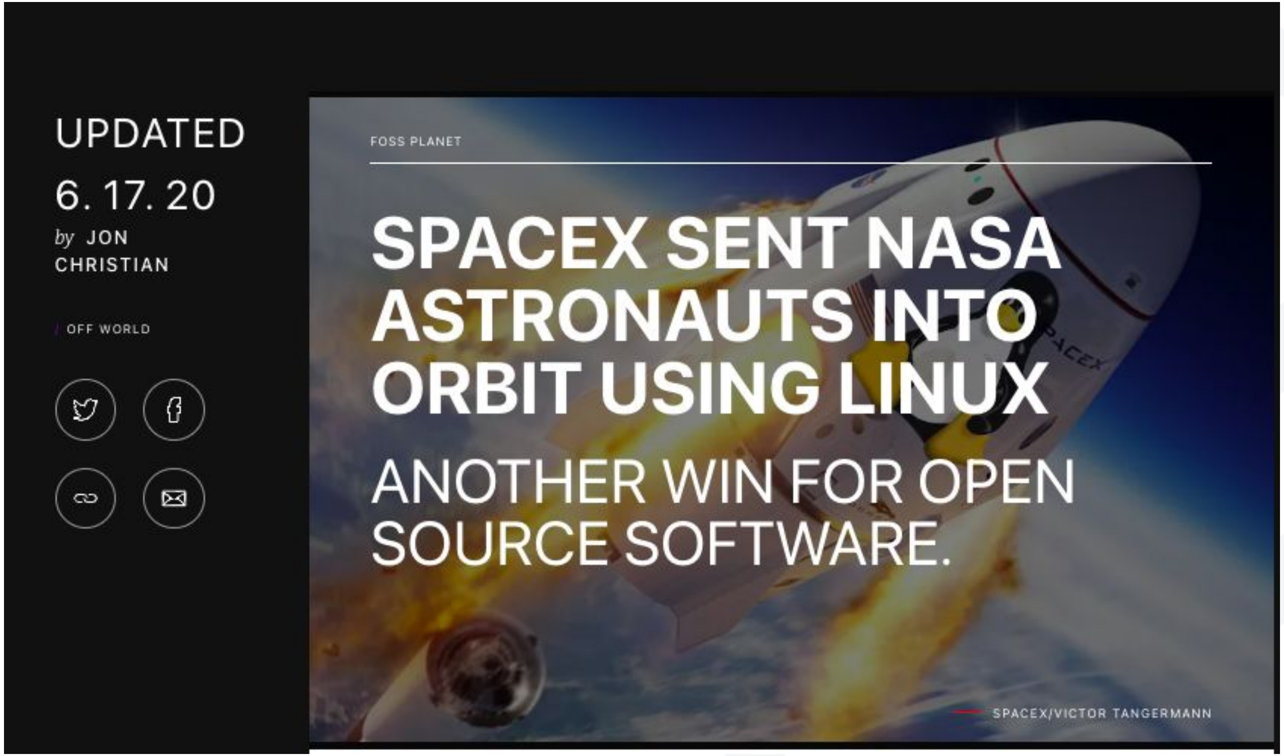
The International Space Station will host a test of computer hardware that's vital for any future missions to Mars.

Written by **Steve Ranger,** Global News Director on Sept. 21, 2017



Source:https://www.extremetech.com/extreme/155392-international-space-station-switches-from-windows-to-linux-for-improved-reliability

ELISA
Enabling Linux in Safety Applications



Source: https://futurism.com/the-byte/spacex-nasa-astronauts-linux

ELISA
Enabling Linux in Safety Applications

PC #BestTechforSchool #AllAboutAI Best Products Reviews How-To News Deals Newsletters

Home > News > Drones

Linux Is Now on Mars, Thanks to NASA's Perseverance Rover

Previous NASA Mars rovers mostly used an operating system from Wind River Systems. But this time, the space agency chose Linux for Perseverance's Ingenuity helicopter drone.

By Michael Kan February 19, 2021

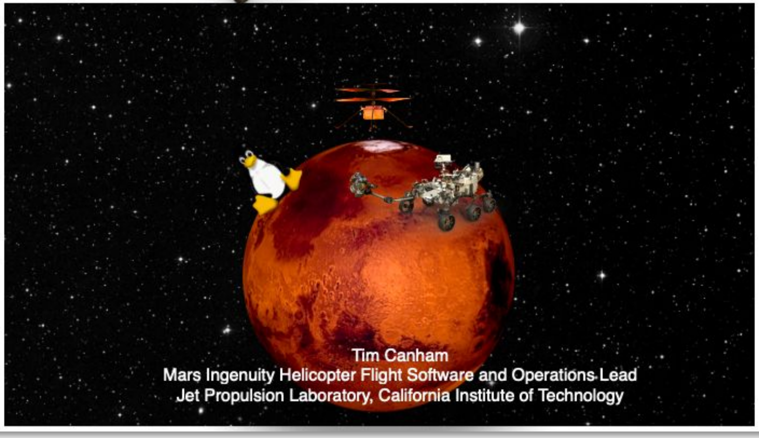
(Credit: NASA)

Source:
https://www.pcmag.com/news/linux-is-now-on-mars-thanks-to-nasas-perseverance-rover

ELISA
Enabling Linux in Safety Applications

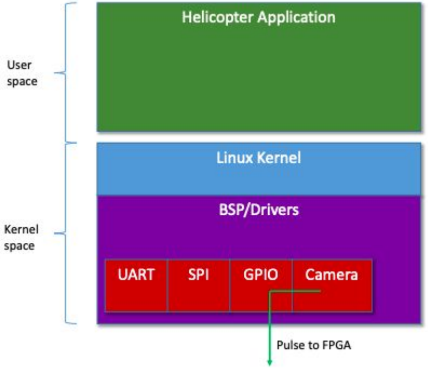
Jet Propulsion Laboratory
California Institute of Technology

Linux on Mars



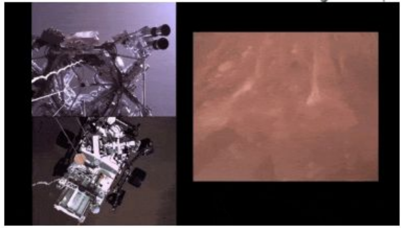
Mars Helicopter Operating System and Software

- Linux
  - Linaro 3.4.0
  - Linux/Android hybrid
  - PREEMPT patch (No RT patch!)
  - BSP provided by Qualcomm/Intrinsyc
  - Camera drivers included with BSP
    - Modified to "pulse" camera interface with FPGA to time-stamp images
  - Linux kernel driver interface for I/O in BSP
  - Helicopter application is fully userspace
    - Runs as root



Perseverance Rover EDL Cameras

- Perseverance Rover also had Linux-based landing camera system
  - Not involved in guidance, just recorded landing
- Ruggedized Intel Atom PC
  - More like a conventional PC
- USB cameras
  - USB cabling with hubs throughout vehicle
  - FTDI to rover interface UART
- Linux x86 kernel 4.15.7
- Used much open-source including ffmpeg and Python



Computer

USB Hub

USB Cameras

Conclusions

- Linux boosted our ability to develop quickly
  - We had standard I/O drivers
  - Manufacturer BSP was available
  - Shell/adb interface made testing much easier
  - COTS facilities like Wi-Fi, USB and standard I/O made test support equipment *much* easier
  - Allowed early prototyping on other platforms like Raspberry Pi
- Linux did very well, as long as you were aware of its limitations
  - Not real time, so built in robustness to slips
    - RT patch probably would have been better, but not available on our kernel
  - Avoid file I/O during performance critical times
  - Build in file-system level protections (ex. read-only partitions for software/Linux executables)
- Future of Linux in space exploration is rosy!

Source: Embedded Linux Conference 2021 https://elinux.org/images/5/5a/1._TIMOTHY_CANHAM.pdf

BEYOND GLOBAL DOMINANCE: OPEN SOURCE IN ORBIT AND BEYOND

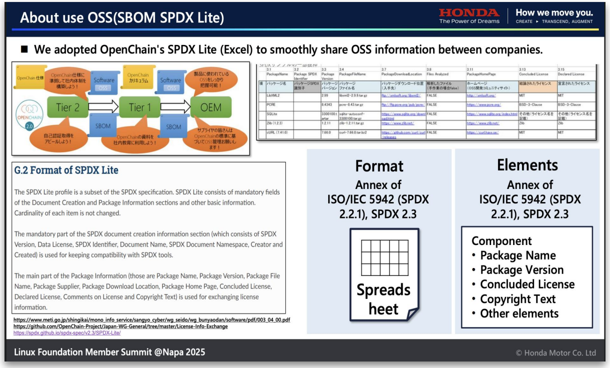
David VomLehn

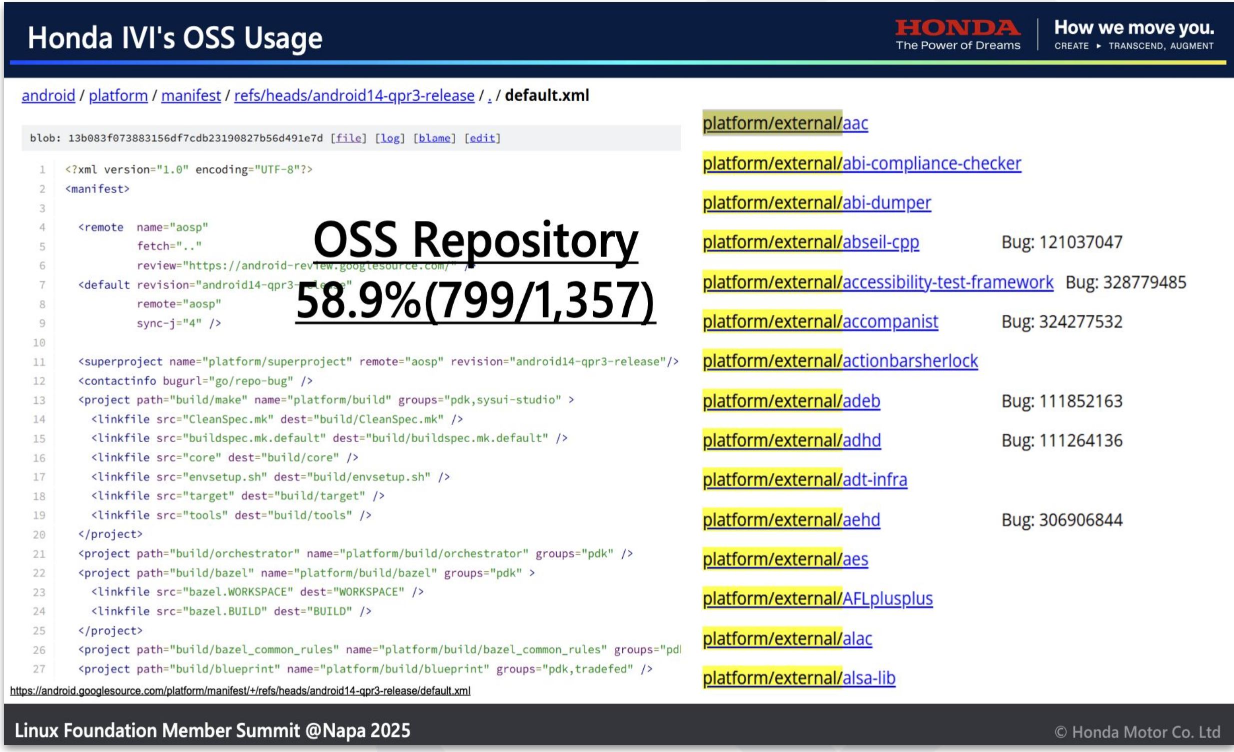
2023 June 28

Embedded Linux Conference 2023
Video:https://www.youtube.com/watch?v=I_LbbvA0NiU&list=PLbzoR-pLrL6qMzIhKVe9lMFnYvFZqrp03&index=5

Controls and Infotainment System Change over 30 Years!

Supply Chain behind a modern car …





Source: https://static.sched.com/hosted_files/lfms25/e0/250312_UnderstandingandManagingSBOMsinModernAutomotiveVehicles_LFMemberSummit2025.r2.pdf

Linux Rate of Change

6.13 Linux Kernel Statistics*

2,001
Contributors From
210 Organizations

6,850
Lines of Code
Added Daily

1,917
Lines of Code
Modified Daily

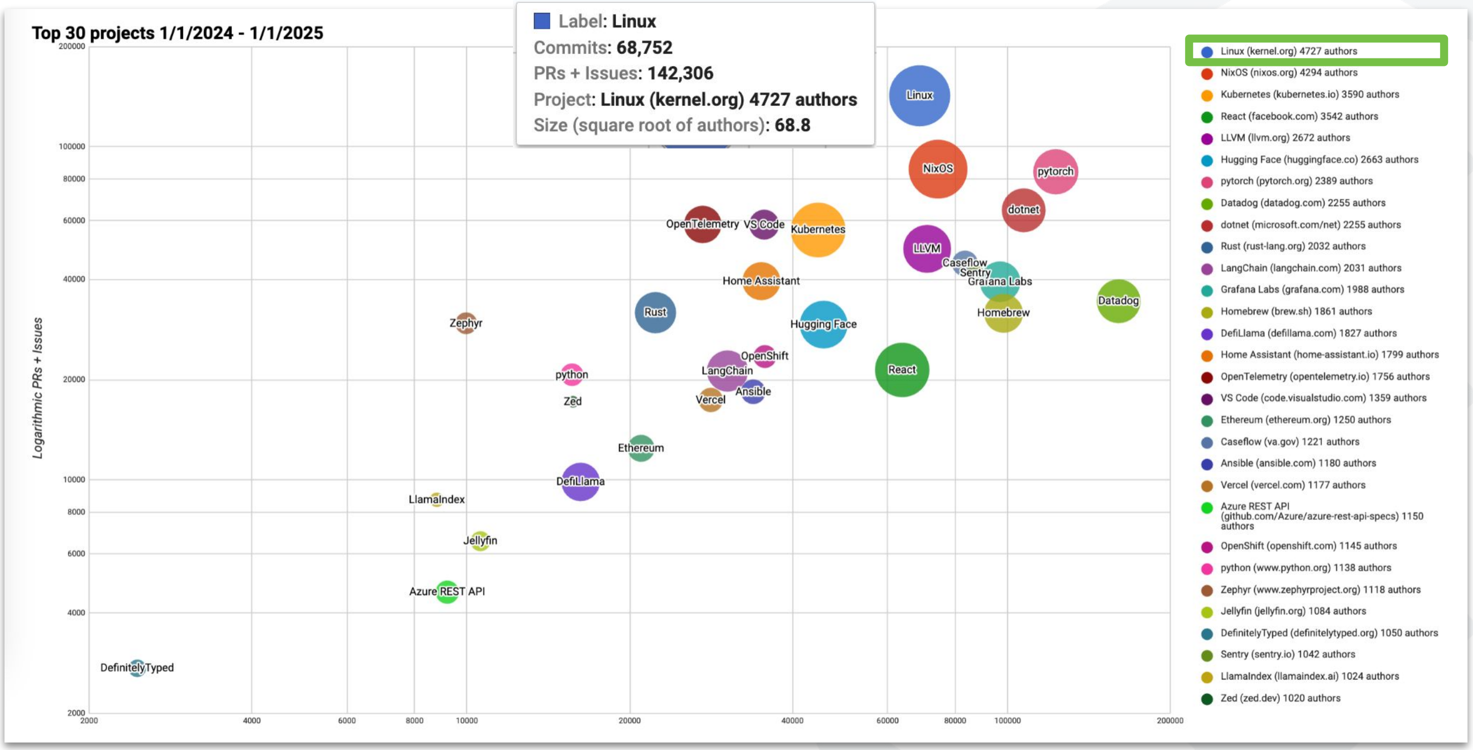
3,786
Lines of Code
Removed Daily

8.6
Changes Per
Hour

ELISA
Enabling Linux in
Safety Applications

Time period for 6.13: 2024/11/18-2025/1/19=63 days
Source: https://github.com/gregkh/kernel-history/blob/master/kernel_stats.ods from 6.13

Top Open Source Projects Velocity



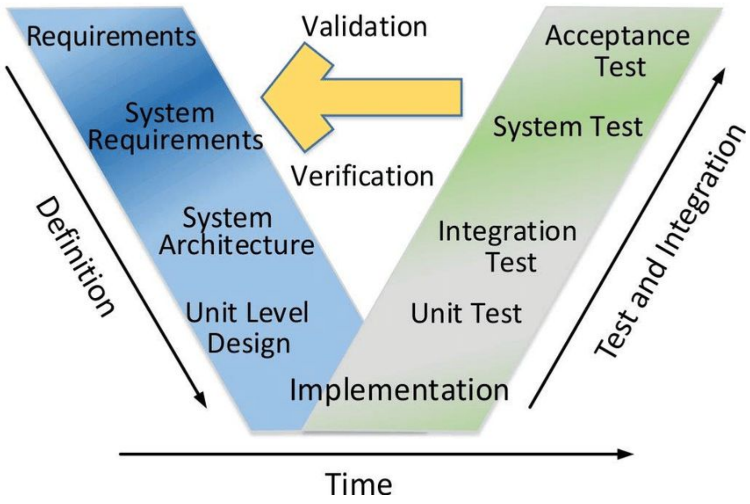
Source: https://github.com/cncf/velocity

ELISA
Enabling Linux in Safety Applications

Safety Standards have emerged from 70+ years of analysis to mitigate systemic risk.

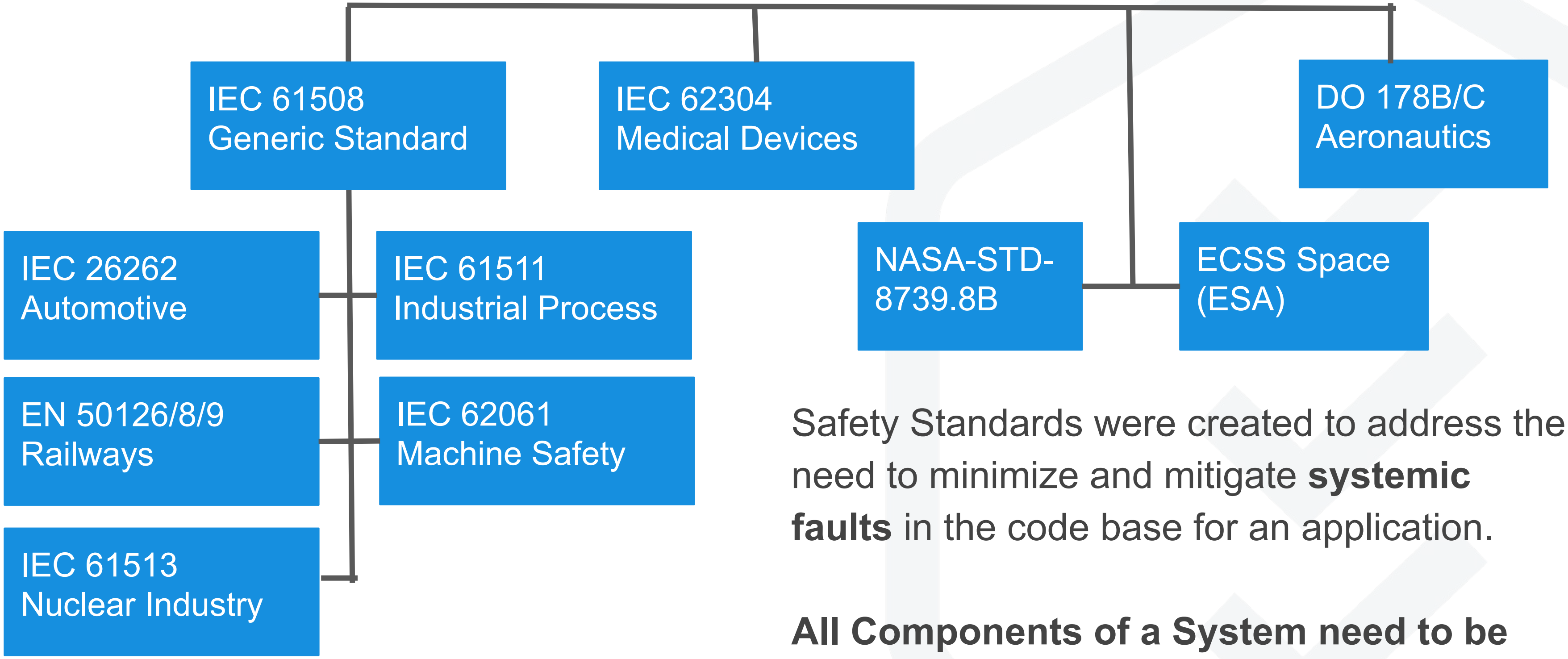
Safety Engineering 101: the "V-Model"

- The V-model for functional safety development originated from systems engineering practices to structure the process of designing and validating complex systems.
- It was later adapted and widely adopted in the automotive industry and other safety-critical sectors as a **framework for ensuring the systematic integration of safety requirements and processes throughout each stage of the development lifecycle.**
  - **ISO 26262** in the automotive industry
  - **IEC 61508** for industrial systems
  - **DO-178C** in aerospace



Source Image image provided under CC-4.0
https://www.researchgate.net/figure/The-functional-safety-development-via-V-model-14_fig4_362572593

Sample Standards for Safety Critical Systems



Safety Standards were created to address the need to minimize and mitigate **systemic faults** in the code base for an application.

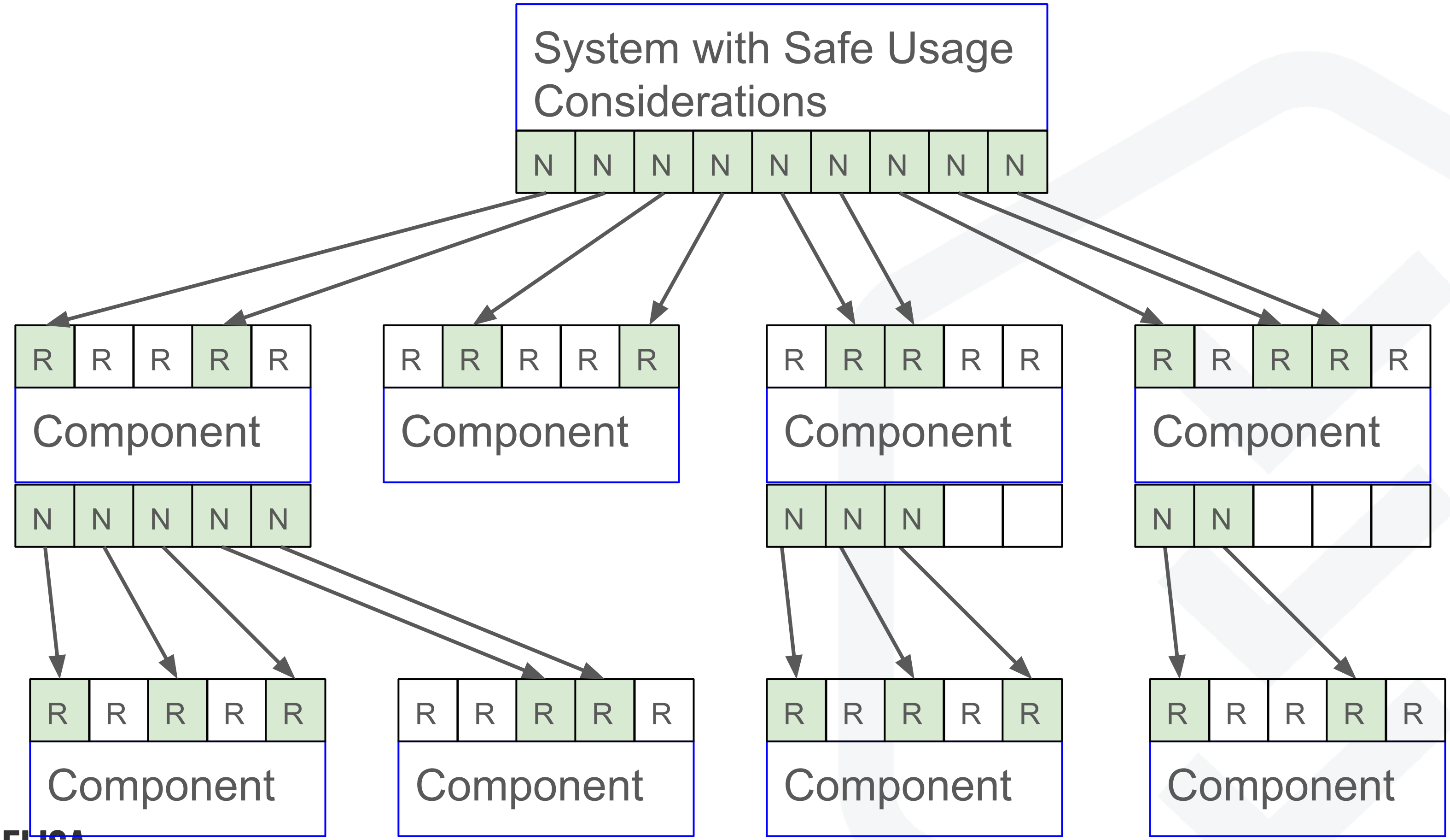
All Components of a System need to be known, tested and managed.

Source: https://www.tuvsud.com/en-us/services/functional-safety/about

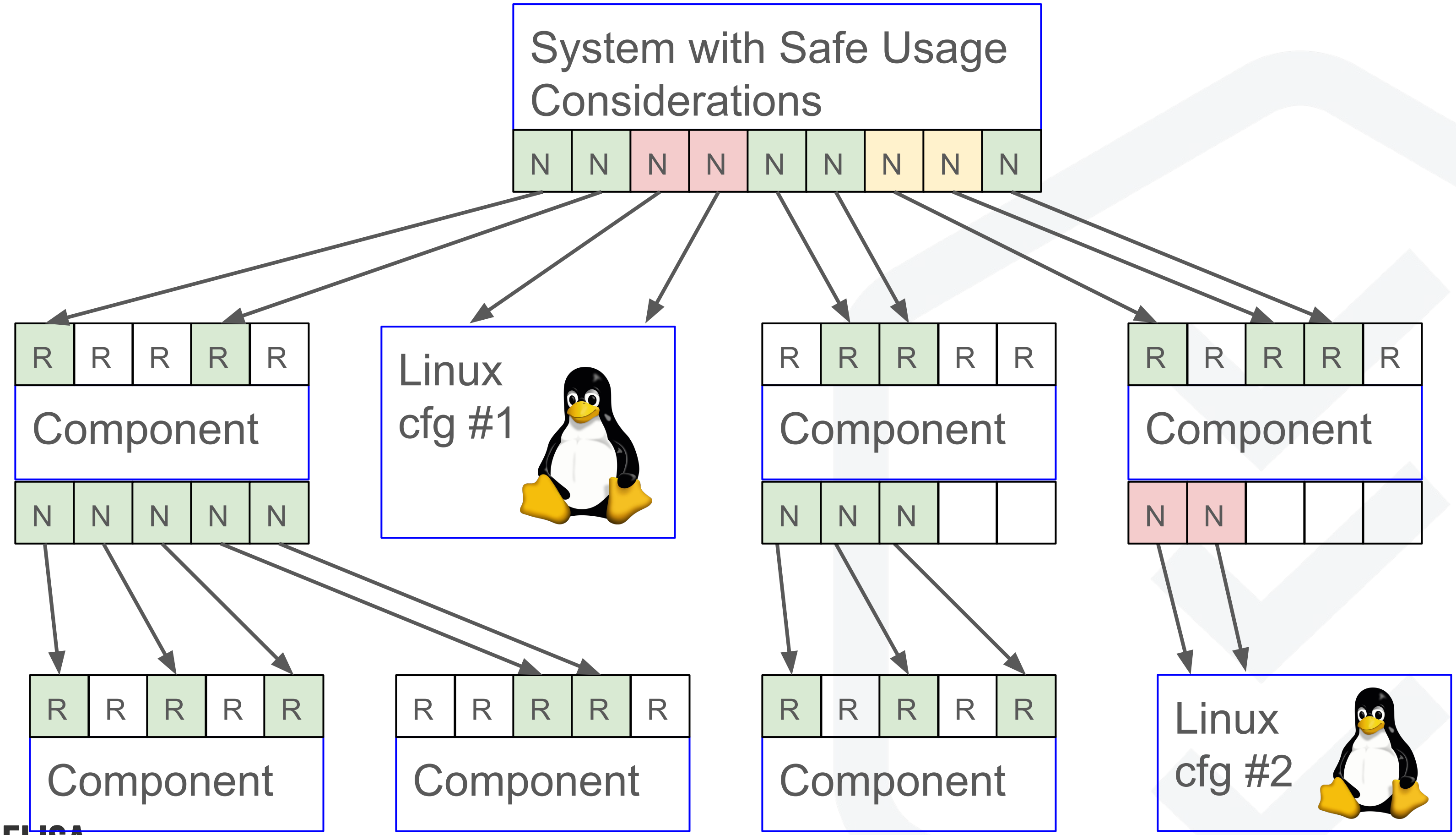
Approximate cross-domain mapping of ASIL

Domain	Domain-Specific Safety Levels						
Automotive (ISO 26262)	QM	ASIL A		ASIL B	ASIL C	ASIL D	-
General (IEC 61508)	-	SIL-1		SIL-2		SIL-3	SIL-4
Railway (CENELEC 50126/128/129)	-	SIL-1		SIL-2		SIL-3	SIL-4
Space (ECSS-Q-ST-80)	Category E	Category D		Category C		Category B	Category A
Aviation: airborne (ED-12/DO-178/DO-254)	DAL-E	DAL-D		DAL-C		DAL-B	DAL-A
Aviation: ground (ED-109/DO-278)	AL6	AL5		AL4	AL3	AL2	AL1
Medical (IEC 62304)	Class A	Class B				Class C	-
Household (IEC 60730)	Class A	Class B				Class C	-
Machinery (ISO 13849)	PL a	PL b	PL c	PL d		PL e	-

Source: https://en.wikipedia.org/wiki/Automotive_Safety_Integrity_Level#Comparison_with_Other_Hazard_Level_Standards



ELISA
Enabling Linux in Safety Applications



ELISA
Enabling Linux in Safety Applications

Risk Management Elements in Systems with Linux

- Hardware
  - Traditional devices, with increasing CPUs, MCUs, GPUs and FPGAs incorporated
- Software
  - Functions are increasingly defined by software; different build configurations
  - Managing interaction between sensors, actuators, humans & environment
  - Managing trained AI/ML models that assist in the safe & efficient operation
- Training Data Sets
  - Data used to train, test & validate the models (AI/ML/…) in used the system
- Communication to Remote Services
  - Updates to the software, firmware & models
  - External environment awareness (GPS positioning, …)

ELISA
Enabling Linux in Safety Applications

Cybersecurity & Critical Infrastructure

Critical Infrastructure

Since 2005, the 'Cybersecurity Policy for Critical Infrastructure Protection' has been set as a common action plan shared between the government, which bears responsibility for promoting independent measures by CI operators relating to CI cybersecurity and implementing other necessary measures, and CI operators which independently carry out relevant protective measures, and the new edition was published in 2022.
This document identifies the 14 sectors as critical infrastructure and it expects stakeholders to undertake the five measures as below.

1. Enhancement of Incident Response Capability
2. Maintenance and Promotion of the Safety Principles
3. Enhancement of Information Sharing System
4. Utilization of Risk Management
5. Enhancement of the Basis for CIP

2. Maintenance and promotion of the safety principles	Basically keep the element of "[1] Maintenance and promotion of the safety principles"	○ Clarify that safety standards, etc., that contribute to the enhancement of incident response capability and risk management are to be developed. ○ Consider survey methods capable of continuously improving the activities of CI operators.

The Cybersecurity Policy for Critical Infrastructure Protection
Full Text
Guideline for Establishing Safety Principles for Ensuring Information Security of Critical Infrastructure(5th Edition)(Revised on May 2019)
Risk Assessment Guide Based on the Concept of Mission Assurance in Critical Infrastructure (1st Edition)(Revised on May 2019)

ELISA
Enabling Linux in Safety Applications

source: https://www.nisc.go.jp/eng/index.html#sec4

How can we address the "GAPS" to enable safe usage of systems based on Linux and keep up with the rate of change of Linux?

ELISA
Enabling Linux in Safety Applications

How can we address the "GAPS" to enable safe usage of systems with Open Source Components like Linux?

- Improve transparency of system components—using BOMs, explicitly documenting supply-chain and build dependencies—to enable automated updates of Safety Usage Analysis metadata in line with the pace of change in open-source component releases, feature updates, and vulnerability fixes.
- Improve ability to analyze components in systems by providing requirements and traceability to code and tests.
- Architect system to manage risk and enable analysis of interactions after change
- Formal certification of key open source components and build infrastructures.

ELISA
Enabling Linux in Safety Applications

Safety Standards Automation?


LINUX.COM TOPIC AUDIENCE RESOURCES ABOUT US

Home › Audience › SBOMs Supporting Safety Critical Software

Audience Contributed Developers Enterprise Topic Governance Open Source

SBOMs Supporting Safety Critical Software

By Linux.com Editorial Staff - March 21, 2022 1850

A software bill of materials (SBOM) is a way of summarizing key facts about the software on a system. At the heart of it, it describes the set of software components and the dependency relationships between these components that are connected together to make up a system.

Modern software today consists of modular components that get reused in different configurations. Components can consist of open source libraries, source code or other external, third-party developed software. This reuse lets innovation of new functionality flourish, especially as a large percentage of those components being connected together to form a system may be open source. Each of these components may have different limitations, support infrastructure, and quality levels. Some components may be obsolete versions with known defects or vulnerabilities. When software runs a critical safety system, such as life support, traffic control, fire suppression, chemical application, etc., being able to have full transparency about what software is part of a system is an essential first step for being able to do effective analysis for safety claims.

- Safety Standards expect to know
  - The **source** code at the time of production release
  - The **documentation of use** associated with the code
  - The **configuration** used to **build** the production software
  - The **specific versions of the tools** used to build the software
  - The **specific hardware** that the software is running on
- Safety Standards Configuration Management (CM) requirements are greatly simplified by leveraging Software Bill of Materials (SBOM) transparency.
  - An SBOM supports capturing the details of what is in a specific release and supports determining what went wrong if a failure occurs.
  - The goal is to be able to rebuild exactly what the executable or binary was at the time of release.
- To learn more, see: https://www.linux.com/featured/sboms-supporting-safety-critical-software/

ELISA Enabling Linux in Safety Applications

Maintenance and Promotion of Safety Principles

Safety Standards are looking for:

- **Unique ID**, something to uniquely identify the version of the software you are using.
  - Variations in releases make it important to be able to distinguish the exact version you are using.
  - The unique ID could be as simple as using the hash from a configuration management tool, so that you know whether it has changed.
- **Dependencies of the component**
  - Any chained dependencies that a component may require.
  - Any required and provided interfaces and shared resources used by the software component. A component can add demand for system-level resources that might not be accounted for.
- The component's **build configuration** (how it was built so that it can be duplicated in the future) and sources
- Any **existing bugs and their workarounds**

SBOMs Scope Today

- **Documentation** for application manual for the component
  - The **intended use** of the software component
  - **Instructions** on how to **integrate** the software component correctly and **invoke it properly**
- **Requirements** for the software component
  - Coverage for nominal operating conditions and behavior in the case of failure
  - For highly safety critical requirements, test coverage should be in accordance with what the specification expects (e.g., Modified Condition/Decision Coverage (MC/DC) level code coverage)
  - Any safety requirements that might be violated if the included software performs incorrectly. This is specifically looking for failures in the included software that can cause the safety function to perform incorrectly. (This is referred to as a cascading failure.)
  - What the software might do under anomalous operating conditions (e.g., low memory or low available CPU)
- **Evidence**
  - This should include the results of any testing to demonstrate requirements coverage

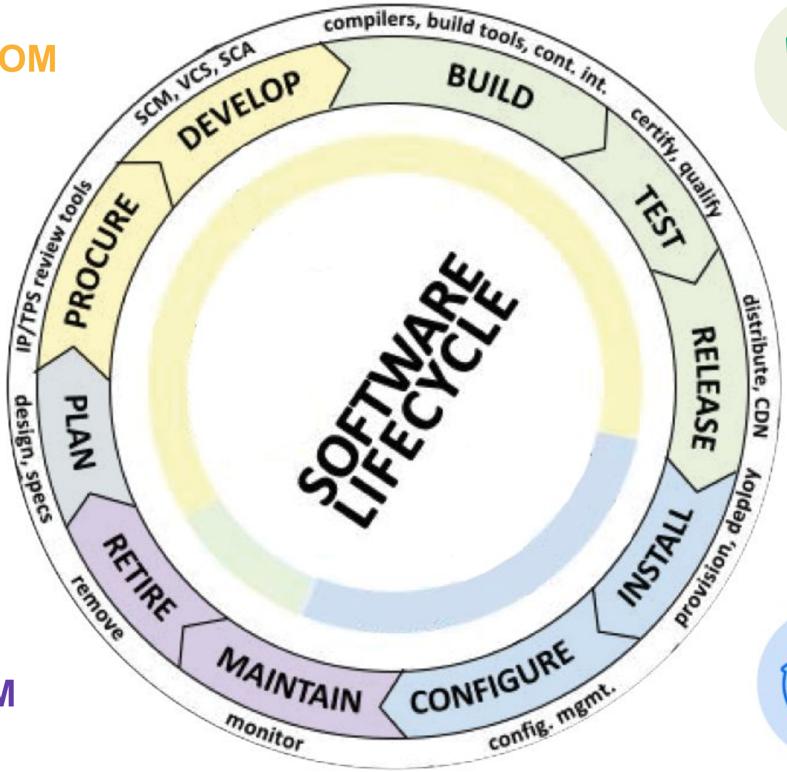
Source: https://www.linux.com/featured/sboms-supporting-safety-critical-software/

Generate Metadata for Elements when the Data is Known!

Source SBOM

Design SBOM

Runtime SBOM



Build SBOM

Deployed SBOM

ELISA
Enabling Linux in Safety Applications

Metadata needs to be **Standardized** & **Accurate**

From all supply chains (**hardware, software, datasets, services**) a standard format should:

- **Capture the data when it is known** in the product's development lifecycle and be explicit
  - **Design** - system requirements, plans, processes
  - **Source** - source files, make scripts, build processes, test files, …
  - **Build** - built applications, libraries, firmware, build configuration, …
  - **Deploy** - application configuration information, installed dependencies, validation,...
  - **Runtime** - system configuration information, …

- **Modular Metadata**
  - Trusted linkage between components - "Your Product is My Component"
  - Metadata automatically generated and updated per component
  - Database friendly for tracking

- **Assemble the facts** into **knowledge** about the **system** and it's intended behavior
  - Use **relationships to link** between metadata about each component
  - Create **knowledge graph** to represent **product line facts** at any point in time including requirements, sources, tests, and evidence that the requirement are satisfied.

ELISA
Enabling Linux in Safety Applications

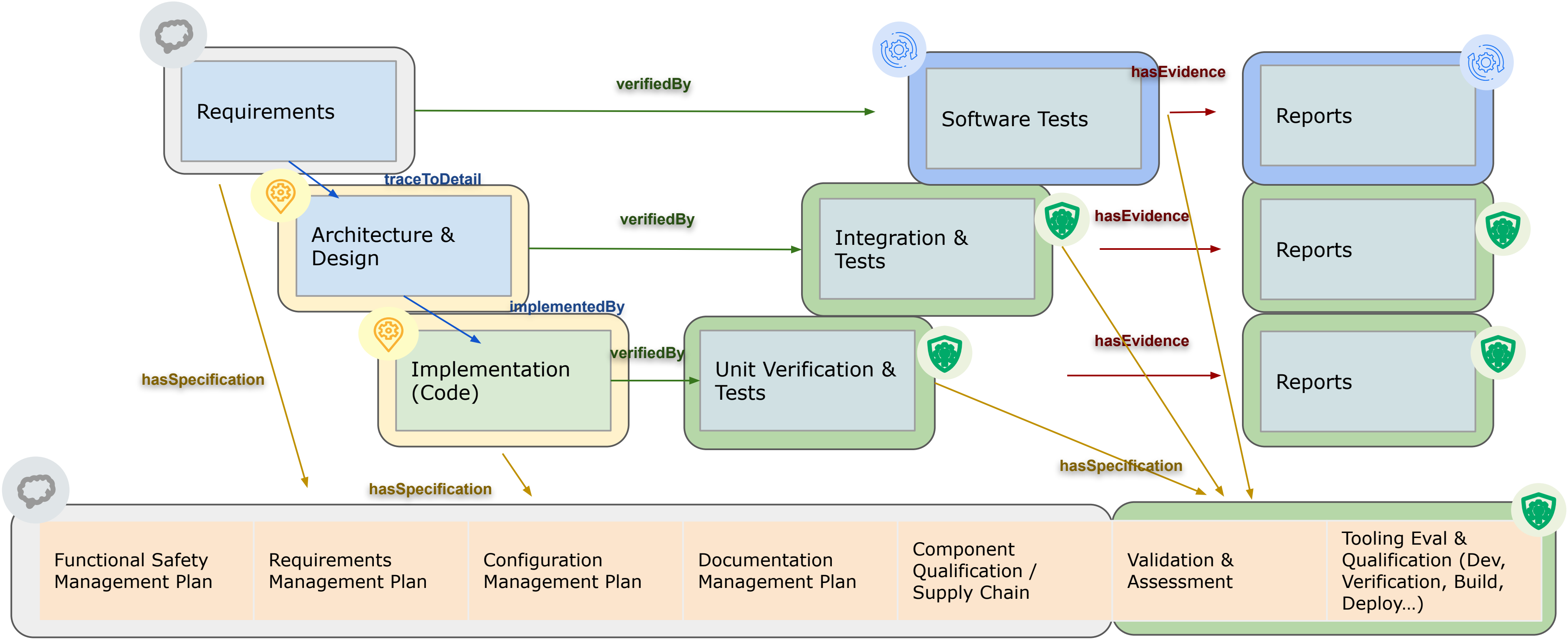
Extending SPDX to Support System Analysis

- **Vulnerabilities occur in Systems** of which Software is only a part.
- **SPDX 3.0** extended already to support
  - **Security** and **safety critical application compliance requirements**
  - **AI/ML** and **Datasets** increasing need for system transparency
  - Software **build provenance**
- Flexibility
  - Designed for **online access**
  - Support **optional inclusion** properties for specific profiles
  - Enhanced **relationship** structure to enable metadata **knowledge graph**
- **SPDX 3.1** extends metadata language to support **system level engineering analysis** and metadata necessary for the **safety cases.**
  - Hardware Profile
  - Services Profile
  - Supply Chain Profile
  - **Safety Profile**

SPDX

Dependencies in a FuSa Project

SPDX SAFETY



SPDX

How can we address the "GAPS" to enable safe usage of systems with Open Source Components like Linux?

- Improve transparency of components in systems (leverage BOMs, make explicit supply chain dependencies, build dependencies), to enable automation to keep Safety Usage Analysis metadata up to date with the rate of change (releases/feature updates/vulnerability fixes) in open source components in systems.
- Improve ability to analyze components in systems by providing requirements and traceability to code and tests.
- Architect system to manage risk and enable analysis of interactions after change
- Formal certification of key open source components and build infrastructures.

ELISA
Enabling Linux in Safety Applications

Challenge: Safety Profile Conformant after Security Fix?

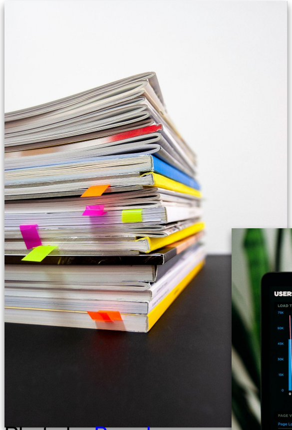


Photo by Bernd Klutsch on Unsplash

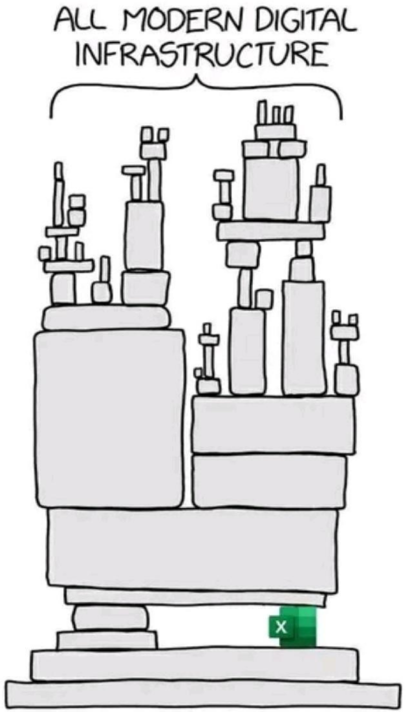
Photo by Luke Chesser on Unsplash

Knowledge base containing:

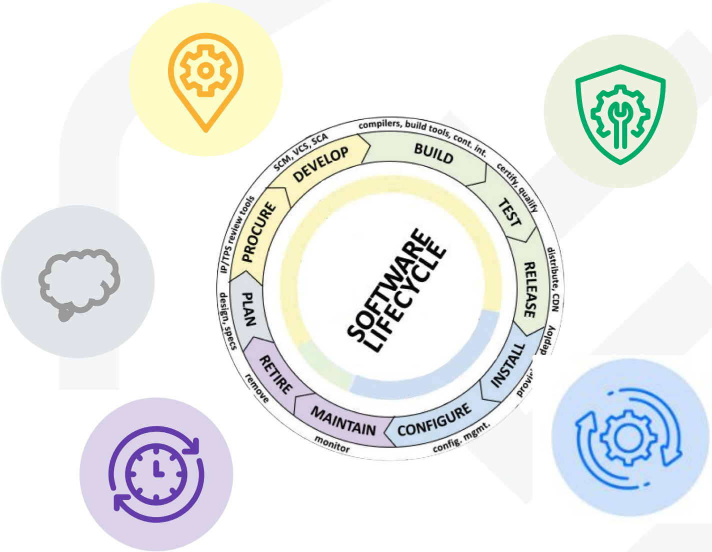
- component **metadata**
- **relationships** between components
- safe usage **requirements**
- **evidence** requirements satisfied

ELISA
Enabling Linux in Safety Applications

Safety Information Analysis



SPDX Safety Elements in Knowledge Graphs!



… instead of inconsistent Spreadsheets, manual import/export of half decent ReqIFs…

ELISA
Enabling Linux in Safety Applications

Relationships Between Elements* Enable **Software Engineering** Analysis for **Risk Management** to be **Automated**

RelationshipType

Meta
describes [element->element]
amendedBy [element->element]
modifiedBy [element->element]
other [element->element] (comment)

Structure
contains [element->element]

Behavioral
configures [element->element]
delegatedTo [element->element]
dependsOn [element->element]

Pedigree
generates [artifact->artifact]
expandsTo
hasAddedfile [element->element]
hasDatafile [element->element]
hasDeletedfile [element->element]
copiedTo [element->element]
packages (obsolete?)

Provenance
ancestorOf [element->element]
descendantOf [element->element]
availableFrom[element->element]
variant [artifact->artifact]

Licensing
hasConcludedLicense [SoftwareArtifact->AnyLicenseInfo]
hasDeclaredLicense [SoftwareArtifact->AnyLicenseInfo]

Security
affects
doesNotaffect
exploitCreatedBy
fixedBy
foundBy
hasAssessmentFor
hasAssociatedVulnerability
publishedBy
reportedBy
republishedBy
underInvestigationFor

Dataset/AI
hasEvidence [element->element]
testedOn [element->element]
trainedOn [element->element]

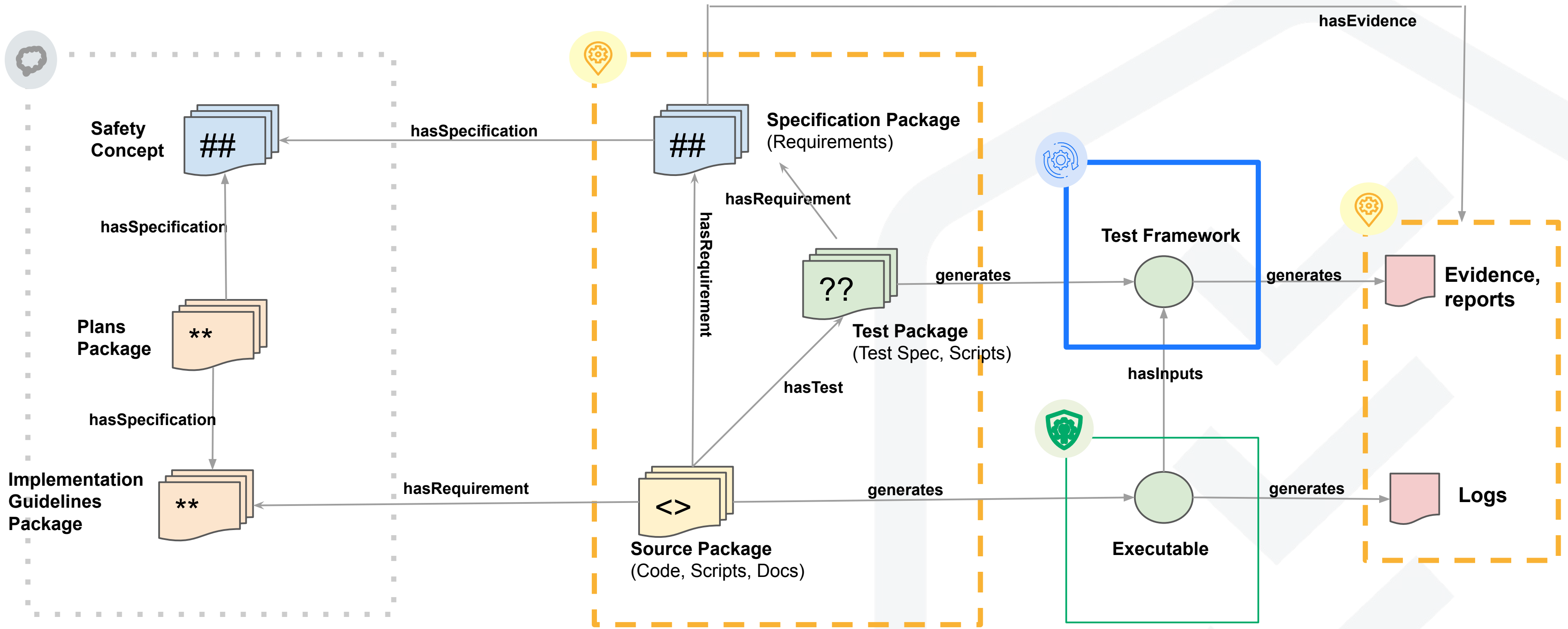
Serialization
serializedInArtifact [SpdxDocument->artifact]

Build
hasDependencyManifest [element->element]
hasDistributionArtifact [element->element]
hasDocumentation [element->element]
hasDynamicLink [element->element]
hasExample [element->element]
hasHost [build->element]
hasInputs [build->element]
hasMetadata [element->element]
hasOptionalComponent [element->element]
hasOptionalDependency [element->element]
hasOutputs [build->element]
hasPrerequisite [element->element]
hasProvidedDependency [element->element]
hasRequirement [element->element]
hasSpecification [element->element]
hasStaticLink [element->element]
hasTest [element->element]
hasTestCase [element->element]
hasVariant [element->element]
invokedBy [element->agent]
packagedBy [element->element]
patchedBy [element->element]
usesTool [element->element]

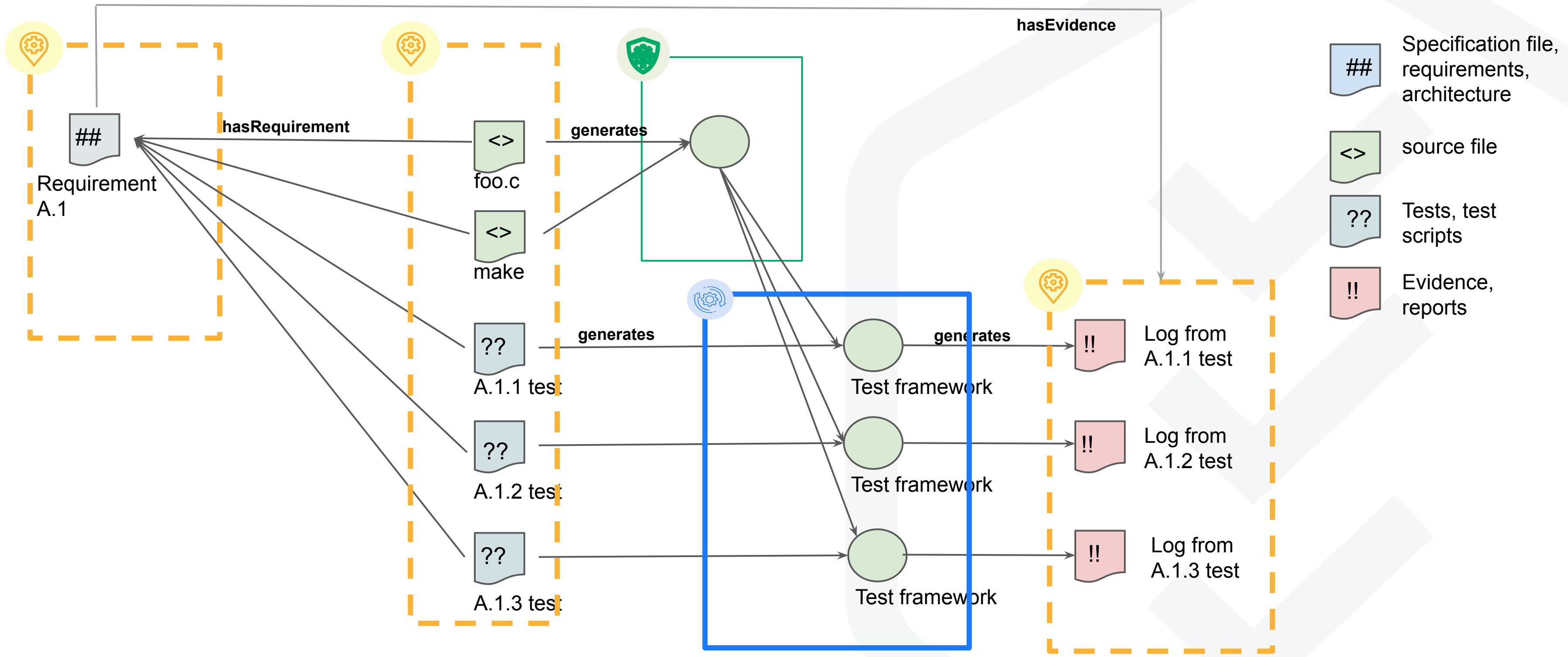
ELISA
Enabling Linux in Safety Applications

* Elements = Collections, SBOMs, Packages, Files, Snippets

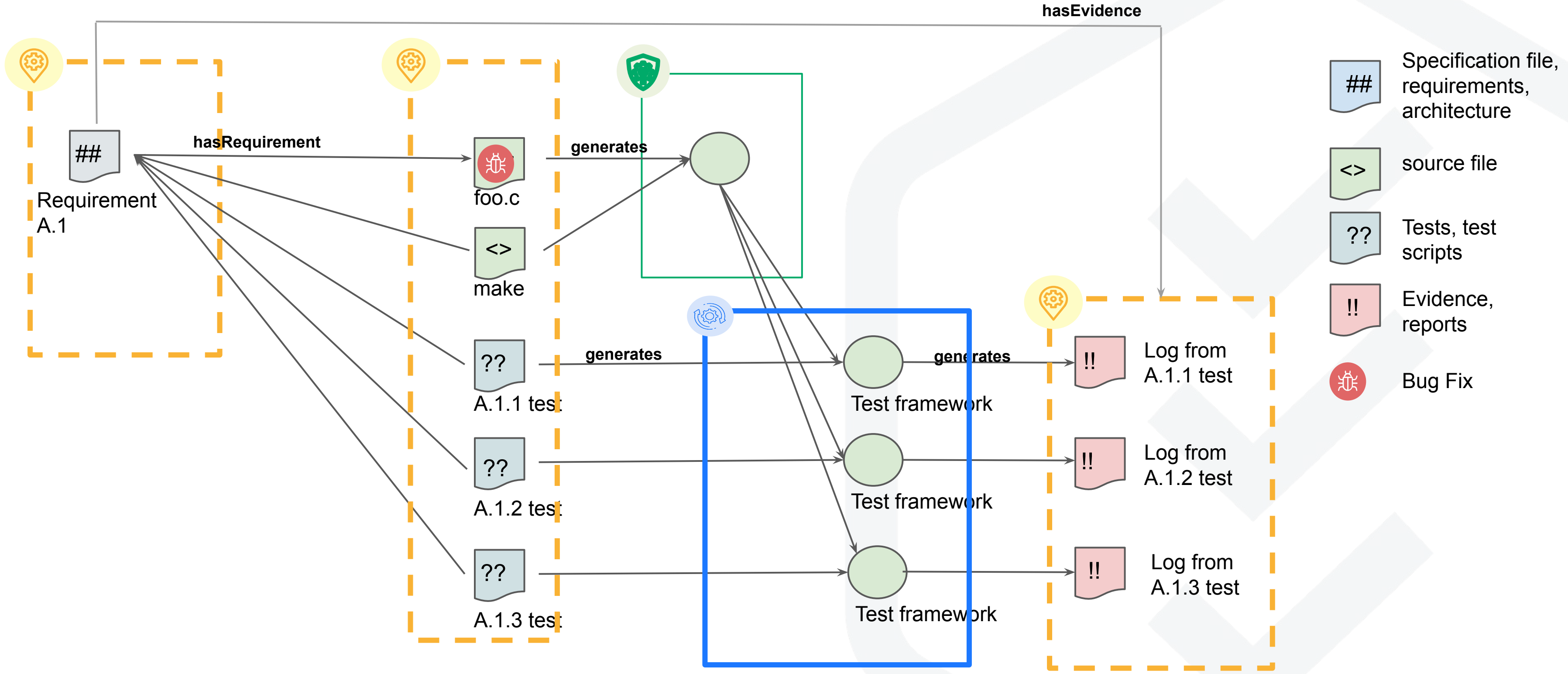
Leveraging SPDX Relationships for Traceability



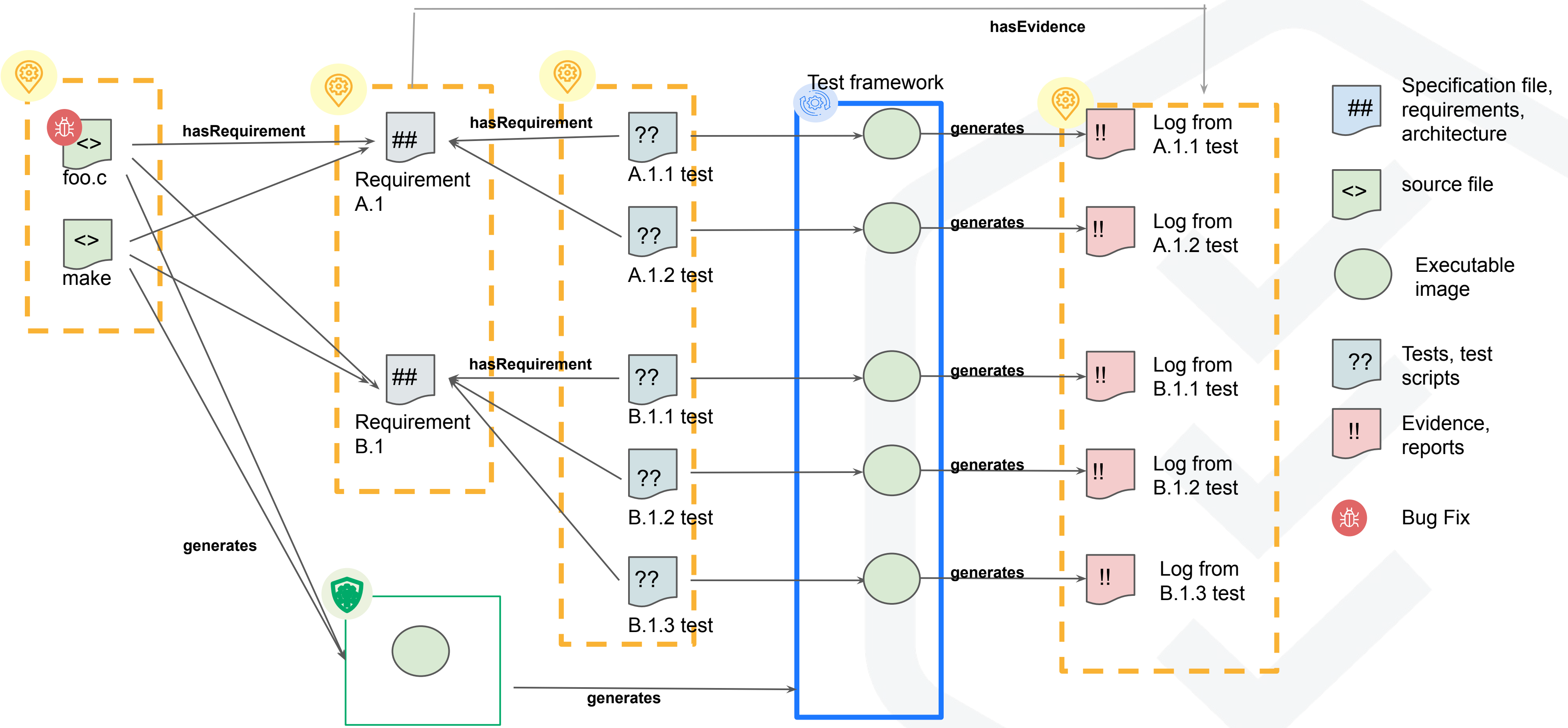
Requirement to Code to Tests to Evidence Traceability



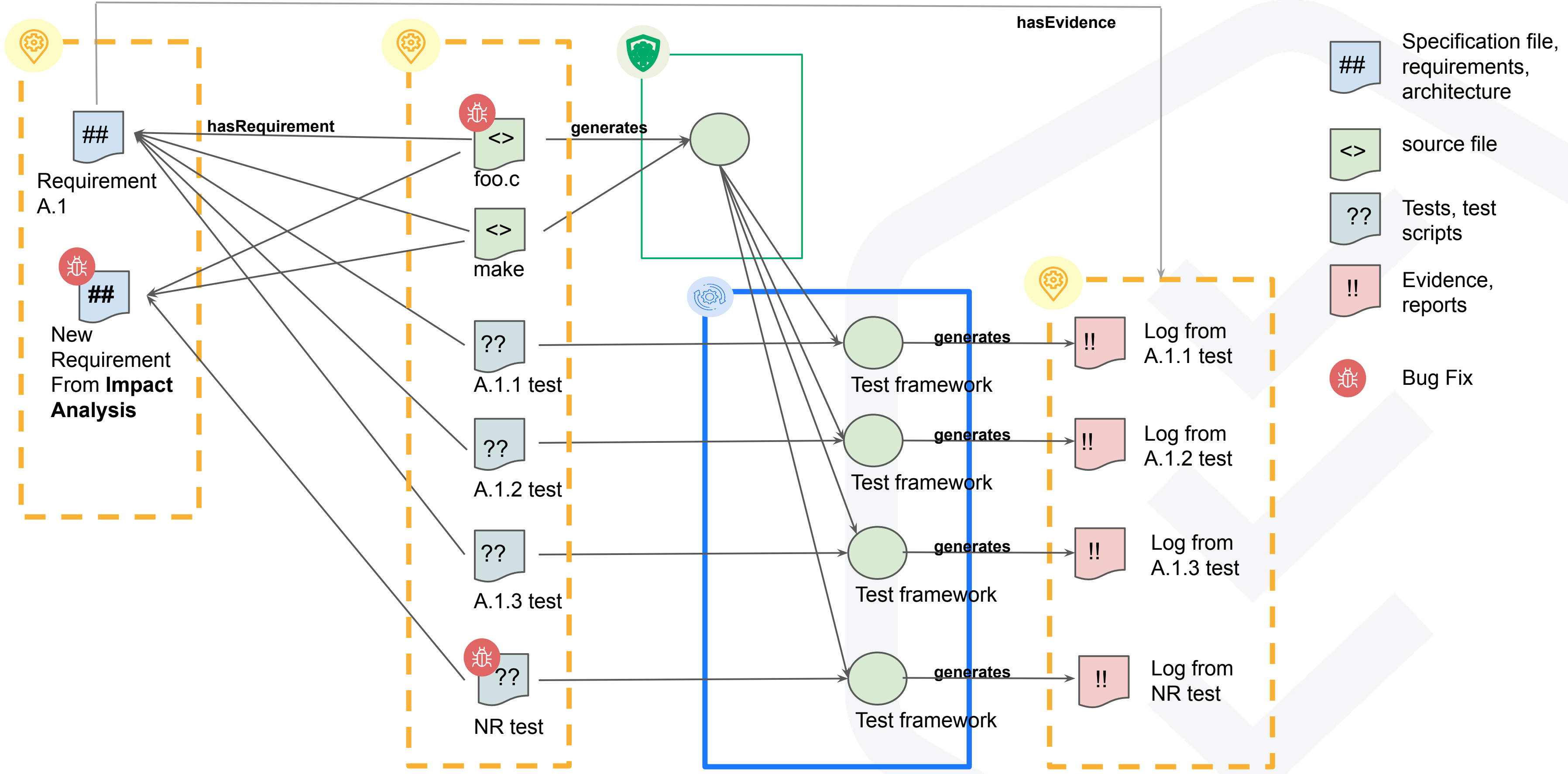
Traceability Enabled After Bug Fix



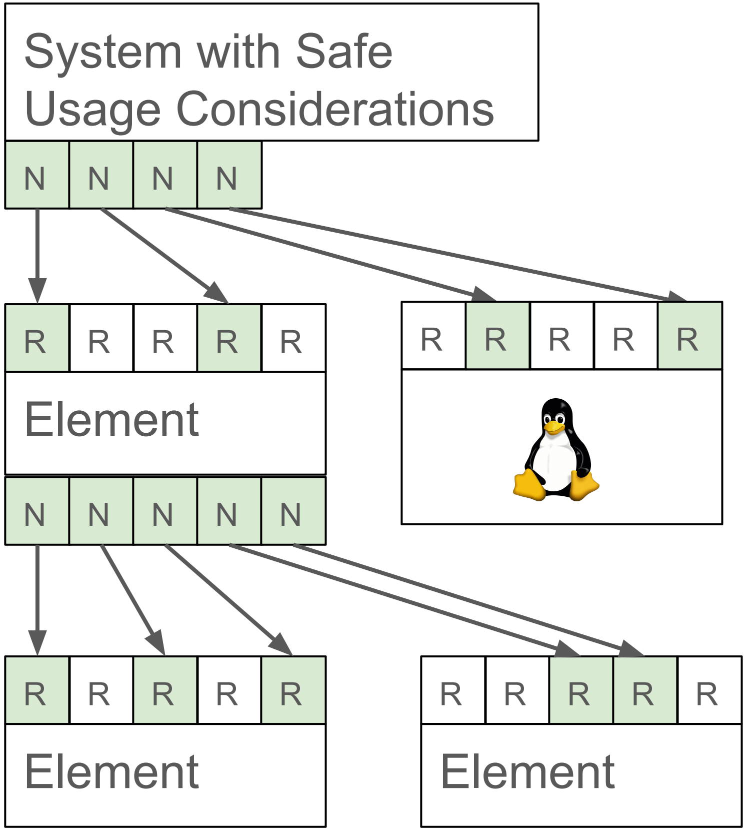
Traceability Enabled After Bug Fix



Requirement to Code to Tests to Evidence Traceability



Safety Element out of Context - SEooC



Development and verification independent of a specific context or application

Provides integration and operation information for safe system integration

Comes with sufficient evidence, that it can be integrated to a safety relevant system.

Automatic SBOM Generation for Linux

The KernelSbom tool

A tool that analyzes a kernel build and produces SBOM documents

https://github.com/TNG/KernelSbom

(The name and home might change)

Example Call:

```
$ export SRCARCH=x86
$ python3 sbom/sbom.py \
    --src-tree path/to/linux \
    --obj-tree path/to/kernel_build \
    --roots arch/x86/boot/bzImage \
    --generate-spdx
```

After merge:

```
$ make sbom
```

Three interlinked files

Output SBOM: `sbom-output.spdx.json`

Describes the packages for the Linux kernel and the kernel modules.

Source SBOM: `sbom-source.spdx.json`

Describes the files that went into the build, with their metadata.

Build SBOM: `sbom-build.spdx.json`

Links the sources to the output artifacts and represents the build graph.

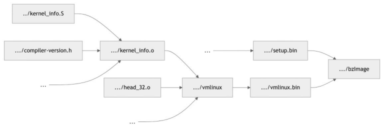
🚧 **This only works with out-of-tree builds right now**

The heuristic to identify which files are source is whether they are in the source directory.

Summary

The graph is created by starting at roots and finding dependencies with the following strategies:

- Parsing the `.cmd` files
  - Parsing the savedcmd_${target} command
  - Parsing source_${target} and deps_${target}
- Parsing the actual files
  - Parsing `.incbin` statements in `.S` files
- Applying some hardcoded dependencies

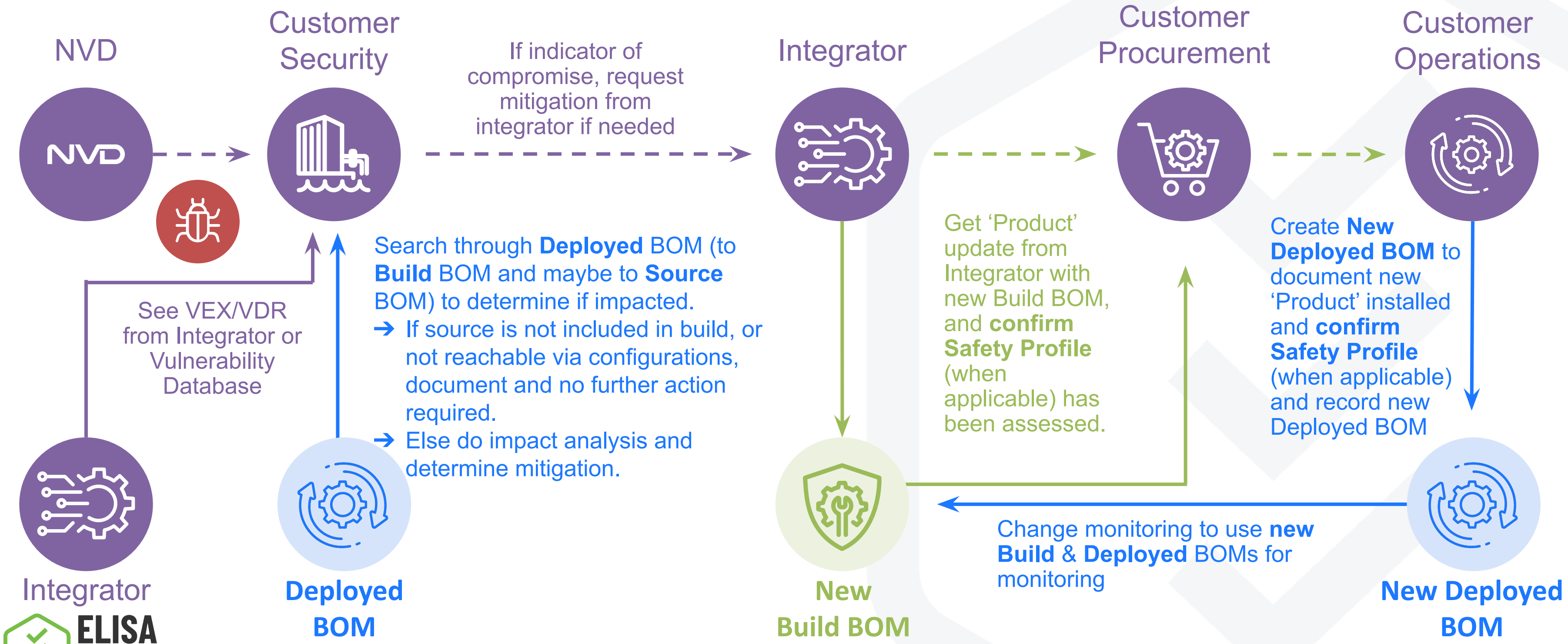


Video and Slides at: https://fosdem.org/2026/schedule/event/938CKB-how_to_create_the_sbom_for_the_linux_kernel/

Prototype at: https://github.com/TNG/KernelSbom

ELISA Enabling Linux in Safety Applications

Managing a security fix:
Customer & Integrator need to check Safety Profile



ELISA
Enabling Linux in Safety Applications

How can we address the "GAPS" to enable safe usage of systems with Open Source Components like Linux?

- Improve transparency of system components—using BOMs, explicitly documenting supply-chain and build dependencies—to enable automated updates of Safety Usage Analysis metadata in line with the pace of change in open-source component releases, feature updates, and vulnerability fixes.
- Improve ability to analyze components in systems by providing requirements and traceability to code and tests.
- Architect system to manage risk and enable analysis of interactions after change
- Formal certification of key open source components and build infrastructures.

ELISA
Enabling Linux in Safety Applications

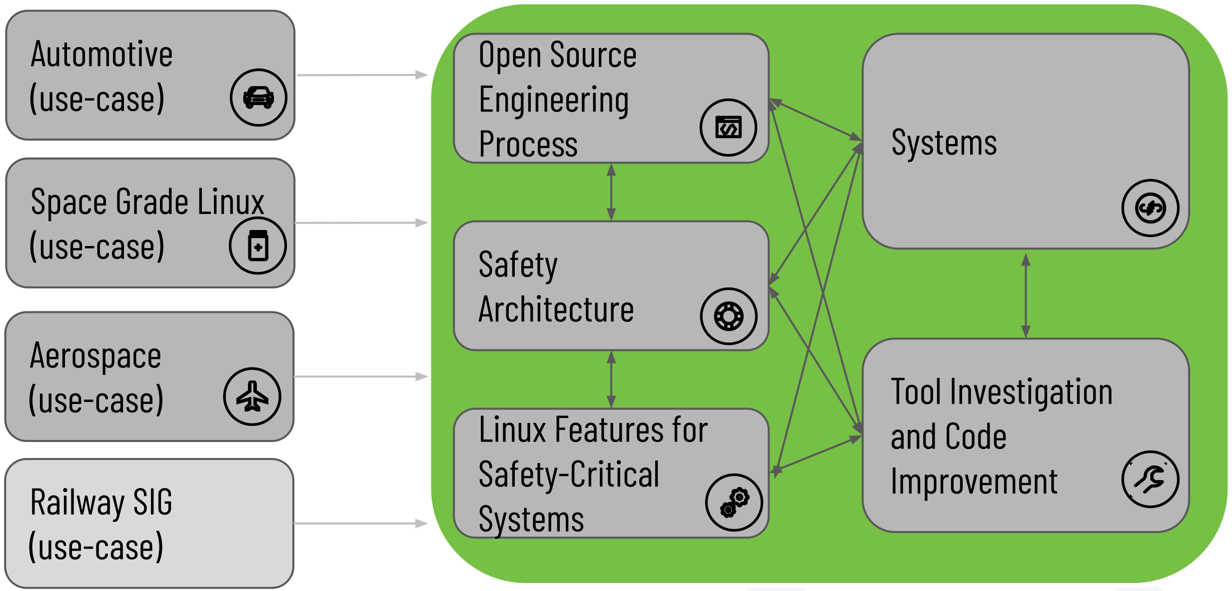
ELISA Project

- Enabling **Safety-critical applications** with **Linux** (beyond Security)
- Increase **dependability & reliability** for whole Linux ecosystem
- **Various use cases**: Aerospace, Automotive, Medical & Industrial
- Supported by major **industrial grade Linux distributors** known for mission critical operation and various industries representatives
- Close community collaboration with **Xen, Zephyr, SPDX, Yocto & AGL** projects
- **Reproducible system** creation from specification to testing
- SW **elements**, engineering **processes**, development **tools**

ELISA : Architecture, Processes, Features, Tools, Systems

ELISA
Enabling Linux in Safety Applications

Current Verticals & Horizontal Working Groups



ELISA Enabling Linux in Safety Applications

Safety Architecture Working Group

ELISA Enabling Linux in Safety Applications

Started with evaluating and improving the Linux Kernel documentation:

- Analyzed the current templates and guidelines that are available in the Linux Kernel documentation,
- Evaluated if and how they fulfill architecture and design aspects required by functionals safety standards,
- Assess maintenance challenges deriving from a continuously evolving code baseline

This document triggered a session presented at Linux Plumbers 2024, and following the discussion in the Safe Usage with Linux miniconference audience, we realized that we need to create a template to document testable requirements in Linux. This lead to work

- Prototyping Linux Kernel Requirements
- Prototyping the automation to check patchsets against requirements
- Refining the initial requirements framework, automation and examples

In 2025, team worked on:

- Prototyping initial requirements framework, automation and examples
- Re-focusing subsystem requirements analysis work to key subsystems identified by the LFSCS working groups (or subsystems that are key to domain specific working groups)
- Using Linux Kernel requirements to support safety analyses in the Kernel

ELISA Enabling Linux in Safety Applications

Linux Kernel Requirements Initiative

- Requirements and code comments do not compile.
- Email based git commit history is hard to mine, to understand "why" a patch was introduced.
- High level design does not trace to low level OSS implementation.
- OSS maintainers cannot be burdened with additional process overhead.
- Proprietary solutions infeasible. OSS code moves too quickly.
- Linux achieved world domination and reliability pressures are immense.

ELISA
Enabling Linux in Safety Applications

Why Attempt Documenting Requirements?

- Improve quality of kernel code
- Improve the efficiency of testing the kernel after changes
- Improve the accuracy of the documentation about functionality
- Share the work that is happening in individual companies today
- Reduce Technical Debt in Kernel
- Support maintainer transitions (Retirement, Change of Focus, etc.)
- Support language transitions (C → Rust → ?)

ELISA
Enabling Linux in Safety Applications

Possible elements impacting the expected behavior

- **Input parameters**: parameters passed to the API being documented;
- **State variables**: global and static data (variables or pointers);
- **Software dependencies**: external SW APIs invoked by the code under analysis;
- **Hardware dependencies**: HW design elements directly impacting the behavior of the code in scope;
- **Firmware dependencies**: FW design elements that have an impact on the behavior of the API being documented (e.g. DTB or ACPI tables, or runtime services like SCMI and ACPI AML);
- **Compile time configuration parameters**: configuration parameters parsed when compiling the Kernel Image;
- **Runtime configuration parameters** (AKA calibration parameters): parameters that can be modified at runtime.

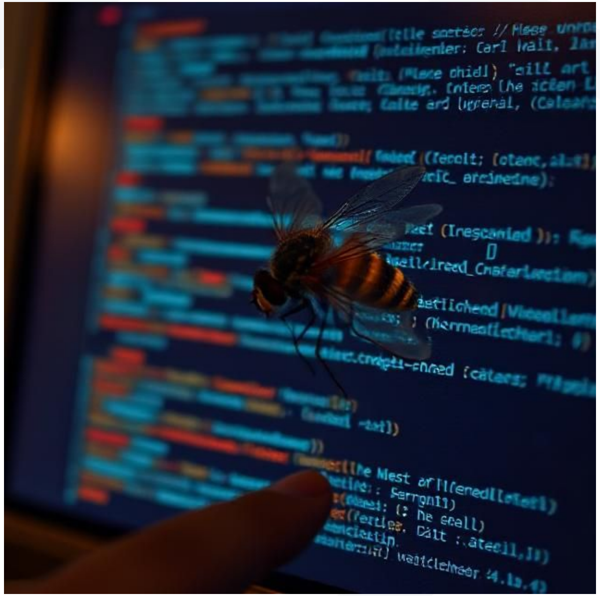
ELISA
Enabling Linux in Safety Applications

Design elements impacted by the expected behavior

- **Return values**, including pointer addresses;
- **Input pointers**:
  pointers passed as input parameter to the API being documented;
- **State variables**:
  global and static data (variable or pointers);
- **Hardware design elements** (e.g. HW registers);

Bug Free Code Principles

1. Document testable expectations (low level requirements).
2. Trace testable expectation to developer intent (e.g. Maintainer Signed-off-by on requirement patch.)
3. Develop pass/fail test and validate testing with code coverage.



The Linux Kernel Requirements initiative is focused on points 1) and 2) while point 3) can be ground for future initiatives.

ELISA
Enabling Linux in Safety Applications

Next Steps

1) Bring more maintainers and respective subsystems on board for expanding the requirements pilots
2) Recruit more people and keep working on requirements pilots and automation
3) Work with Kernel CI to link requirements to testing infrastructure
4) Refine the requirements' template and tooling to get a requirements' framework adopted by upstream Linux Community.

ELISA
Enabling Linux in Safety Applications

Interested in Helping Us Move this Forward?

- Subscribe to mailing list and post questions/comments - https://lists.elisa.tech/g/safety-architecture

- Join us on weekly calls on Friday https://lists.elisa.tech/g/safety-architecture/calendar

- Experiment in a kernel subsystem you're familiar with
  - start prototyping the requirements for code
  - get review from team before upstreaming please!
  - Identify tests from kernel CI that test the subsystem you've highlighted
  - Match kernel CI tests to the requirements

ELISA
Enabling Linux in Safety Applications

Some of the Tools Evolving for Requirements Management

ELISA
Enabling Linux in
Safety Applications

BASIL
The FuSa Spice

Zephyr™

StrictDoc

Xen Project

OFT

Systems Working Group

Enable other working groups to put their **safety claims towards Linux in a system context**.

Focus Points:

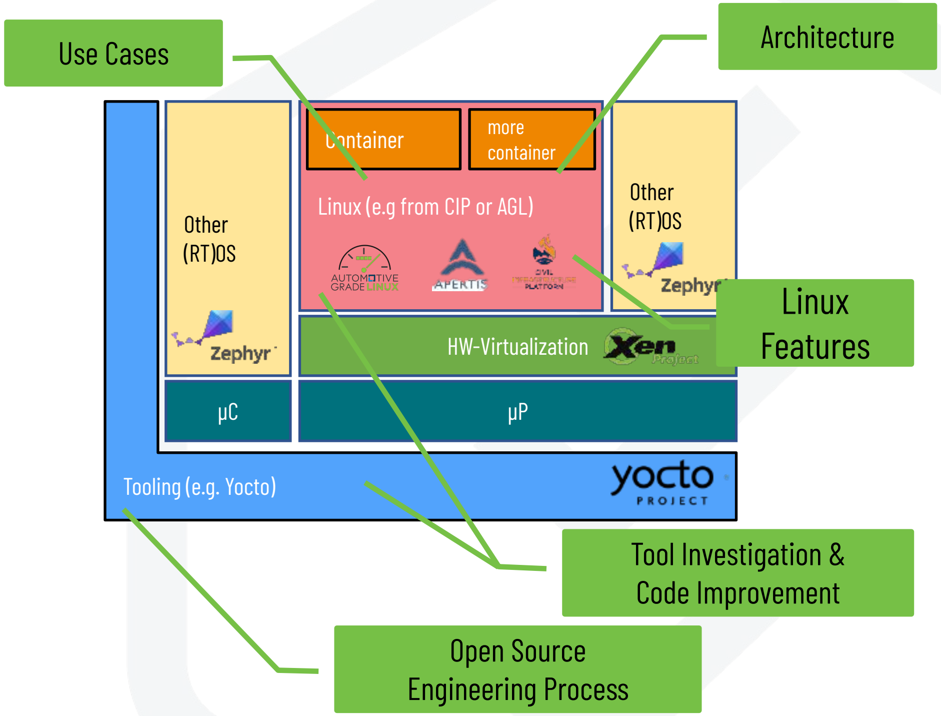
- Provide a reproducible reference system based on real world architectures.
- Reference system fully automated and fully based on Open-Source technologies.
- Interactions with other OSS projects with relevance to mixed-criticality system elements.

Activities:

- Working on systems to connect Linux with hypervisor and RTOS & explore implications of OSS projects interacting mixed criticality systems, prototyping SPDX Safety Profile
- Sandbox for **Linux**, **Xen** & **Zephyr** interacting with **AGL** and automatic SBOM generation from **Yocto.**
- Working with SOAFEE to create reference knowledge graph database from CI/CD pipeline.

ELISA Working Groups - Fit in a sample system

- **Linux Features, Architecture** and **Code Improvements** should be integrated into the reference system directly.
- **Tools** and **Engineering process** should serve the reproducible product creation.
- **Automotive, Aerospace, Space Grade Linux** and future WG use cases should be able to strip down the reference system to their use case demands.



ELISA Working Groups - Deliverables

- Elements / Software
- Processes
- Tools
- Documentation

meta-elisa

Reproducible system

STPA

Codechecker

Workload tracing

ks-nav

Basil

RT Linux

GitHub / Gdrive / Blog / Whitepaper

ELISA
Enabling Linux in Safety Applications

How can we address the "GAPS" to enable safe usage of systems with Open Source Components like Linux?

- Improve transparency of system components—using BOMs, explicitly documenting supply-chain and build dependencies—to enable automated updates of Safety Usage Analysis metadata in line with the pace of change in open-source component releases, feature updates, and vulnerability fixes.
- Improve ability to analyze components in systems by providing requirements and traceability to code and tests.
- Architect system to manage risk and enable analysis of interactions after change
- Formal certification of key open source components and build infrastructures.

Communities with Safety Analysis Support Activity

- ELISA Enabling Linux in Safety Applications
- SPDX
- Xen Project
- Zephyr™
- KernelCI
- yocto PROJECT
- AUTOMOTIVE GRADE LINUX
- SOAFEE
- Eclipse SDV
- CIVIL INFRASTRUCTURE PLATFORM

ELISA Enabling Linux in Safety Applications

Subset of Open Source Projects Evolving to Support Requirements Traceability → Enable higher Risk Systems

Linux: ELISA Enabling Linux in Safety Applications, KernelCI

RTOS: Zephyr™

Virtualization/ Hypervisor: Xen Project

yocto PROJECT

Reproducible Build Framework

ELISA Enabling Linux in Safety Applications

Known Active Certification of Open Source Projects

Toolchain:	Language has been certified, libraries WIP
Linux:	System certifications exist, details not visible. Working on requirements template to make it easier to share/reuse traceability
Virtualization/ Hypervisor: Xen Project	Requirements traceability WIP, Concept approval
RTOS: Zephyr™	Requirements traceability WIP, Concept approval

ELISA
Enabling Linux in Safety Applications

JOIN THE ELISA COMMUNITY

Our infrastructure and tools are open by default, so jump in and introduce yourself, ask questions and share ideas. Please consider this your invitation to participate.

Subscribe to Mailing Lists

Join Community Meetings

Contribute to Tools and Docs on GitHub

Participate in Working Groups

Attend Events

ELISA
Enabling Linux in Safety Applications

Recommendations for new contributors

- Just show up – All presented projects are open for the adaptation of new use cases, input, domain-specific working groups etc. Join the mailing list, show up at the meetings.
- Share Safety Best Practice: Functional and structural expectations of the component used in the context of the entire system, volunteer to do a seminar on the problems you're seeing.
- Become an OSS evangelist: Open source can already be used in a variety of safety contexts. Knowledge of the actual structure and potential is very scarce in the field of assessors, notified bodies and related authorities.

ELISA
Enabling Linux in Safety Applications

Engage with ELISA

https://elisa.tech

https://github.com/elisa-tech

https://www.linkedin.com/company/elisa-project/

https://www.youtube.com/@elisaproject8453

ELISA
Enabling Linux in
Safety Applications