

Building Dependable Embedded Systems with Open Source Components

Kate Stewart, VP, Dependable Embedded Systems
The Linux Foundation

May 9, 2024

LinkedIn: <https://www.linkedin.com/in/katestewartAustin/>
Email: kstewart@linuxfoundation.org



Photo by [Piron Guillaume](#) on [Unsplash](#)









“Ingredients” for a Modern MRI Machine

- Hardware
 - Traditional Hardware BOM, but with more CPUs, MCUs & GPUs incorporated
- Software
 - Managing interaction between sensors, actuators, humans & environment
 - Managing trained AI/ML models that assist in the safe & efficient operation
 - Significant portions are **moving to Open Source** (AGL, SDV, etc.)
- Training Data Sets
 - Data used to train, test & validate the AI/ML models used for diagnosis
- Communication to Remote Services
 - Logging and Storage of Patient data
 - AI/ML models to support analysis & diagnosis
 - Updates to the software, firmware & AI/ML models

MRI Machines Vulnerabilities

- **Hackers** can sneak into a hospital or medical network through phishing or clicking on a bad link in an email and take control of a medical device managed on network like an MRI.
- WannaCry ransomware 2017 leveraging component in devices
- Training Data Sets Poisoned
- **Software Upgrades**

FDA expecting a “Cybersecurity Bill of Materials”

H. R. 2617—1375

“(b) CYBERSECURITY REQUIREMENTS.—The sponsor of an application or submission described in subsection (a) shall—

“(1) submit to the Secretary a plan to monitor, identify, and address, as appropriate, in a reasonable time, postmarket cybersecurity vulnerabilities and exploits, including coordinated vulnerability disclosure and related procedures;

“(2) design, develop, and maintain processes and procedures to provide a reasonable assurance that the device and related systems are cybersecurity, and make available postmarket updates and patches to the device and related systems to address—

“(A) on a reasonably justified regular cycle, known unacceptable vulnerabilities; and

“(B) as soon as possible out of cycle, critical vulnerabilities that could cause uncontrolled risks;

“(3) provide to the Secretary a software bill of materials, including commercial, open-source, and off-the-shelf software components; and

“(4) comply with such other requirements as the Secretary may require through regulation to demonstrate reasonable assurance that the device and related systems are cybersecurity.

“(c) DEFINITION.—In this section, the term ‘cyber device’ means a device that—

“(1) includes software validated, installed, or authorized by the sponsor as a device or in a device;

“(2) has the ability to connect to the internet; and

“(3) contains any such technological characteristics validated, installed, or authorized by the sponsor that could be vulnerable to cybersecurity threats.

Source: <https://www.congress.gov/117/bills/hr2617/BILLS-117hr2617enr.pdf>

Q4: What requirements apply to manufacturers of cyber devices under section 524B of the FD&C Act?

A: Section 524B(a) of the FD&C Act provides that the sponsor of a premarket submission for a cyber device must include information to demonstrate that the cyber device meets the cybersecurity requirements in section 524B(b) of the FD&C Act. The requirements in section 524B(b) of the FD&C Act are:

- Submit a plan to monitor, identify, and address, as appropriate, in a reasonable time, postmarket cybersecurity vulnerabilities and exploits, including coordinated vulnerability disclosure and related procedures;
- Design, develop, and maintain processes and procedures to provide a reasonable assurance that the device and related systems are cybersecurity, and make available postmarket updates and patches to the device and related systems; and
- Provide a software bill of materials, including commercial, open-source, and off-the-shelf software components

The FDA may also issue regulations with other requirements to demonstrate reasonable assurance that the device and related systems are cybersecurity. See FAQs 6 through 9 for additional details on ways manufacturers might demonstrate that their devices are cybersecurity.

Source: <https://www.fda.gov/medical-devices/digital-health-center-excellence/cybersecurity-medical-devices-frequently-asked-questions-faqs>

Effective March 29, 2023, the FDA started enforcing cybersecurity requirements for medical devices, to include a Cybersecurity Bill of Materials (**SBOM + Device + Related Systems**)



“Ingredients” for a Self-Driving Car

- Hardware
 - Traditional BOM, but with more CPUs, MCUs & GPUs incorporated
- Software
 - Managing interaction between sensors, actuators, humans & environment
 - Managing trained AI/ML models that assist in the safe & efficient operation of the vehicle
 - Significant portions are **moving to Open Source** (AGL, SDV, etc.)
- Training Data Sets
 - Data used to train, test & validate the AI/ML models in use the system
- Communication to Remote Services
 - External environment awareness for navigation support
 - Updates to the software, firmware & AI/ML models



Chemical



Financial



Commercial
Facilities



Food &
Agriculture



Communi-
cations



Government
Facilities



Critical
Manu-
facturing



Healthcare &
Public Care



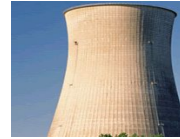
Dams



Information
Technology



Defense
Industrial
Base



Nuclear
Reactors,
Materials, &
Waste



Emergency
Services



Transpor-
tation
Systems

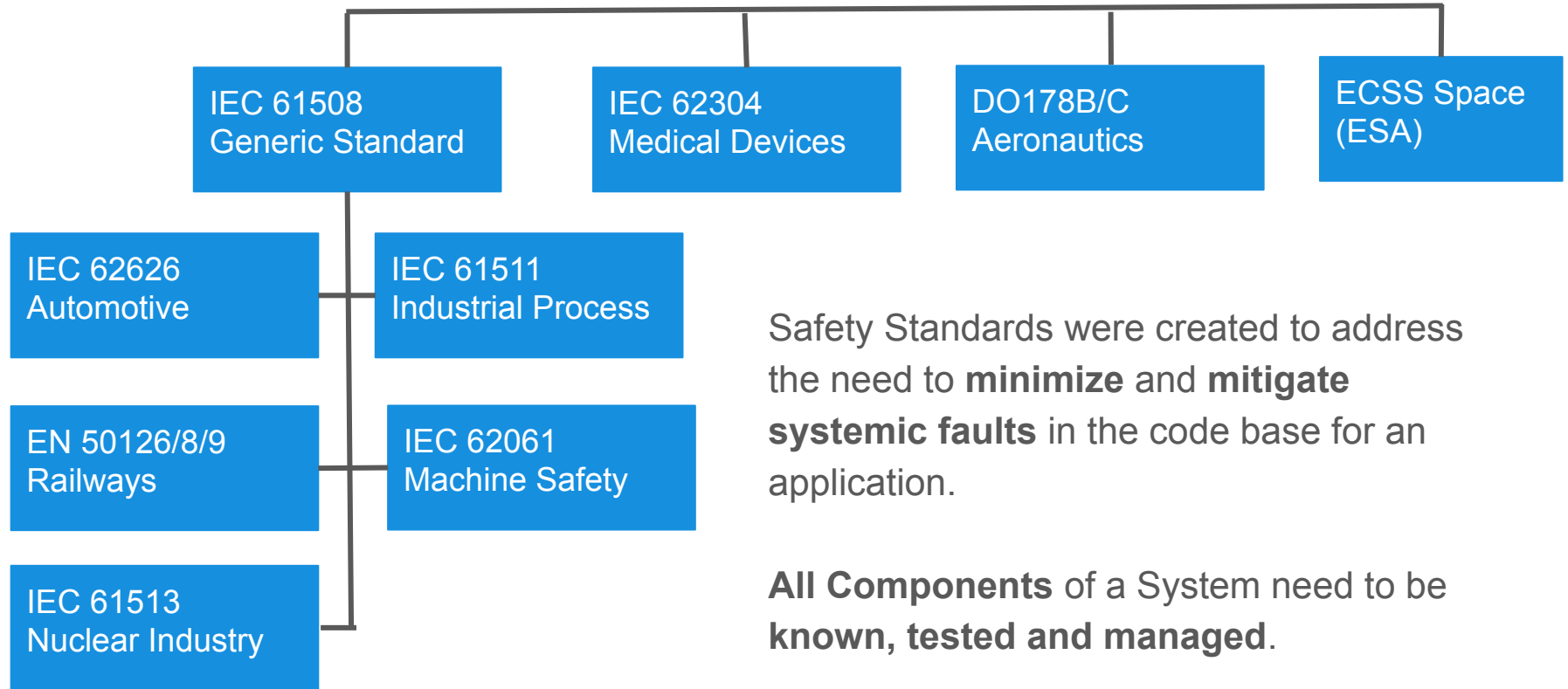


Energy



Water &
Wastewater
Systems

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.**

More Ingredients \Rightarrow More Ways Can Go Wrong

- **Software Vulnerabilities**
 - Interaction between proprietary and open source components in system
 - Assessment if a mitigation needs to be applied to an incorporated image or not.
- **Hazards from AI/ML model**
 - Biases in training data sets
 - Interaction issues after update of model and with other software on system
- **Training Data Sets**
 - Data used to train, test & validate the AI/ML models in use the system
- **Communication to Remote Services (Network)**
 - External Connectivity for proper functioning of device
 - Software & model updates

June 2022: Japan 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"	<ul style="list-style-type: none">◇ 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 \(1st Edition\)\(Revised on May 2019\)](#)

Safety Standards Automation?

- 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/>



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**

Can Be Available in SBOM

- **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
 - This should include the results of any testing to demonstrate requirements coverage
 - 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)

Product's Supply Chain Safety Analysis MetaData

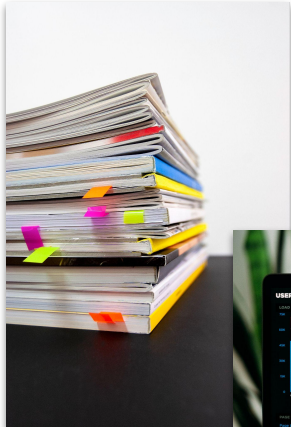
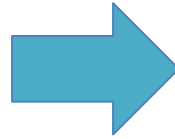


Photo by [Bernd Klutsch](#) on [Unsplash](#)



Photo by [Luke Chesser](#) on [Unsplash](#)



Knowledge base containing :

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

Challenge: Maintenance when Open Source Evolves

Applying a Vulnerability Fix

Requirements are needed to know you're "**done**" after applying a patch:

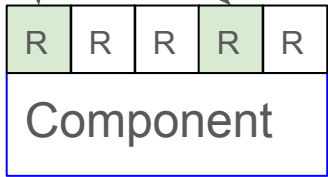
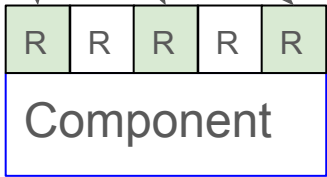
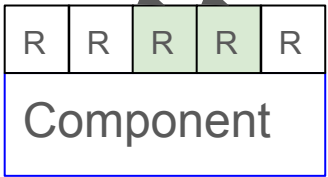
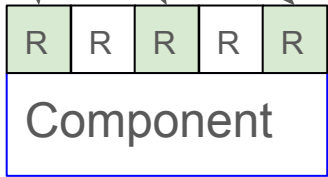
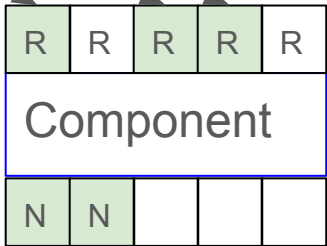
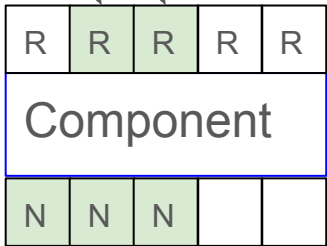
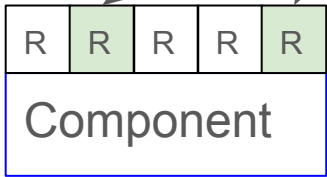
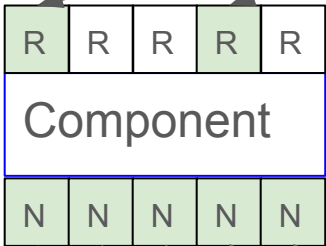
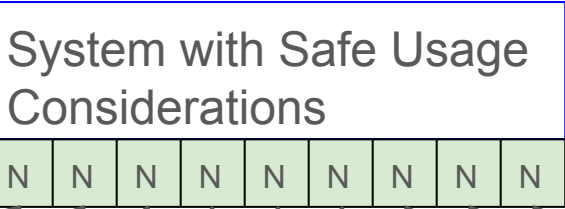
- Need to be able to ensure you have compliance to the updated system requirements after applying a patch
- Given the rate of change and vulnerabilities, we need a way to make this automated, so it needs to be **machine readable**
- For **each file patched**, what requirements does it interact with, **what tests need to be rerun** to regenerate the evidence

Software Bill of Materials (SBOMs) today:

- **Machine readable** - Identities & Dependencies are part of the minimum definition
- SPDX SBOMs can also enable recording and connecting the sources, assessments, vulnerabilities & patches, build & calibration data, tests, requirements and evidence ⇒ **path to automation**

Software BOM \Rightarrow System BOM?

We need to enhance **tracking dependencies** between the “**ingredients**” beyond software, especially when there are **safety elements** to be considered



Standardized Metadata From **All Supply Chains**

All supply chains contributing “ingredients” (hardware, software, data sets, services) need to provide **metadata in a standard format**, so risk can be accurately assessed and managed.

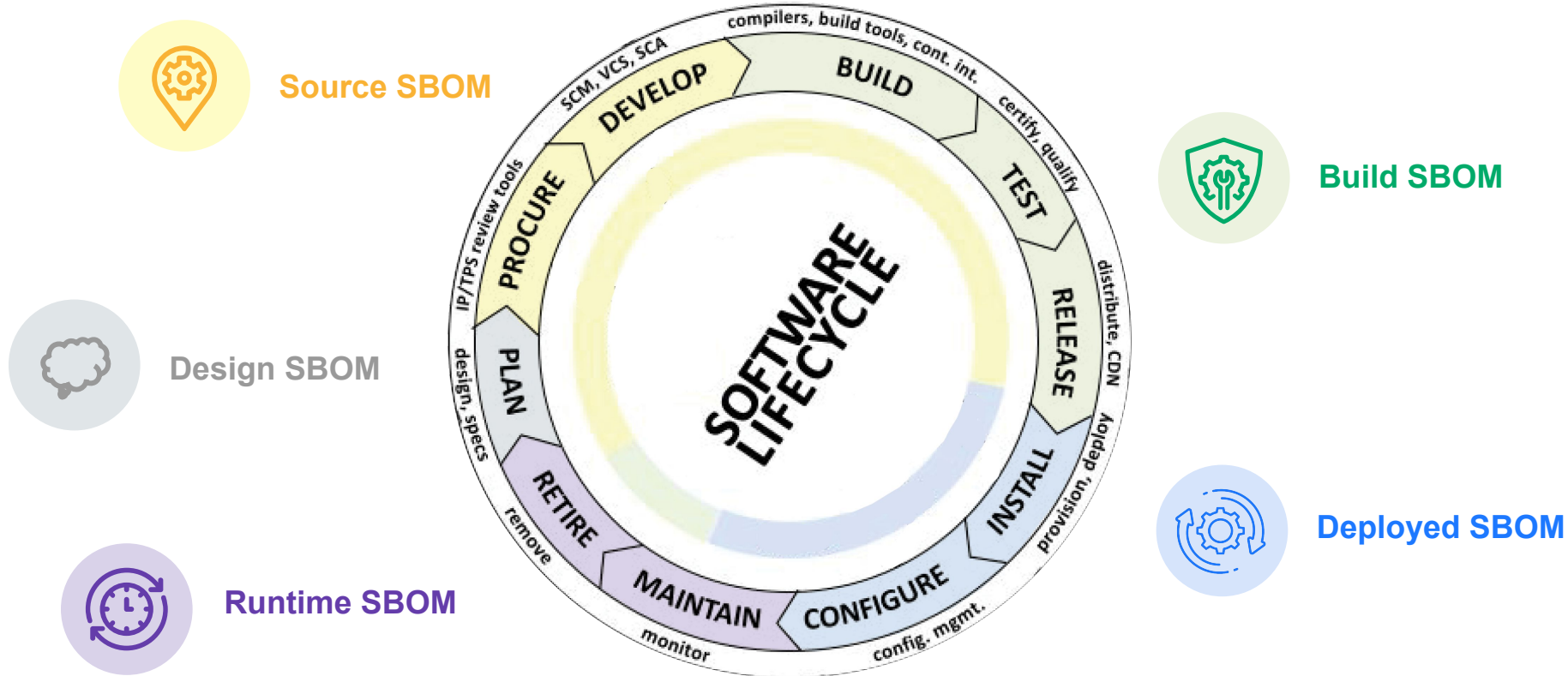
- What software component versions are executing on which specific hardware devices (and/or models, and/or simulators/FPGAs)?
- What software components direct and transitive dependencies should be monitored for vulnerabilities?
- What is the provenance of how a model was trained? What datasets were used for testing and validation?
- How were the datasets used for training created? Are there known biases?
- How were the software components and models integrated and tested?
- What APIs are used to manage updates through remote services?
- What remote services does the running software and trained models depend on? What happens when the service is not available?
- How tracking updates to software, model, data sets in a product line, so current picture at any point in time?

Standardized Metadata Needs to be **Accurate**

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

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

Best practice in Software is to generate SBOMs when the **facts** are known... lets extend this to Systems!

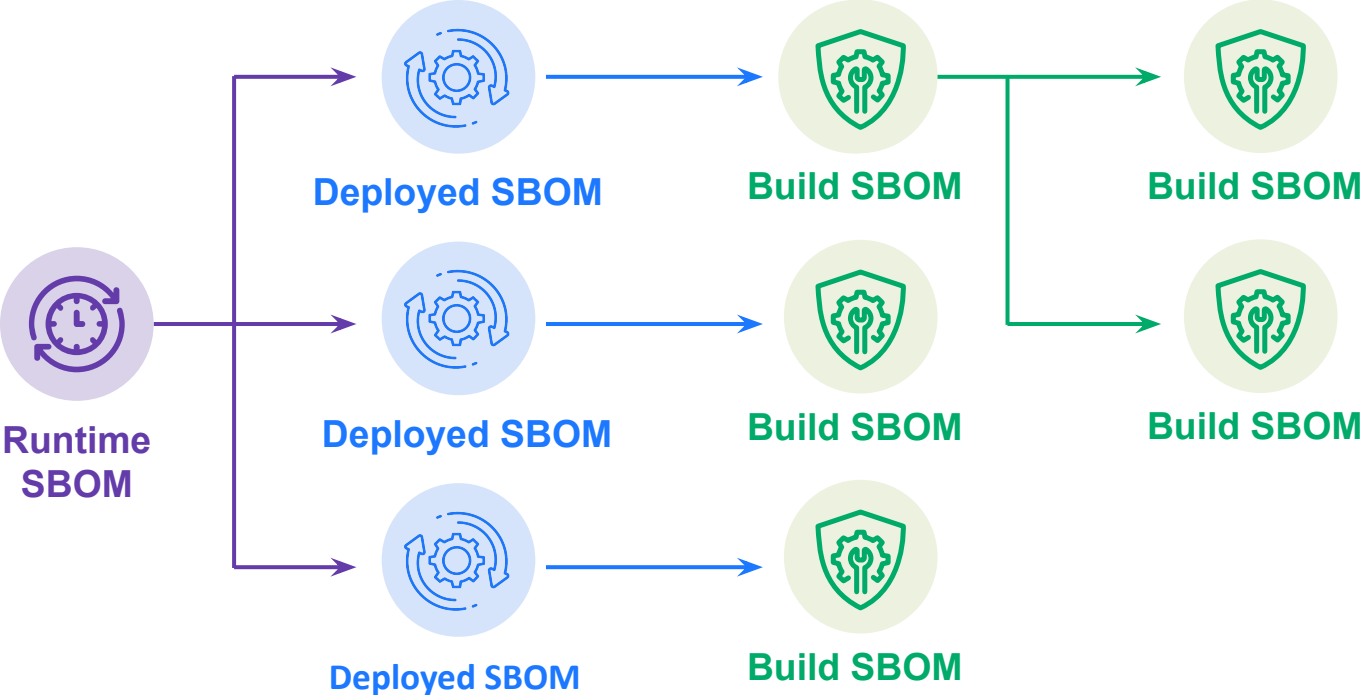


SBOM Types - CISA Definition provide a framework

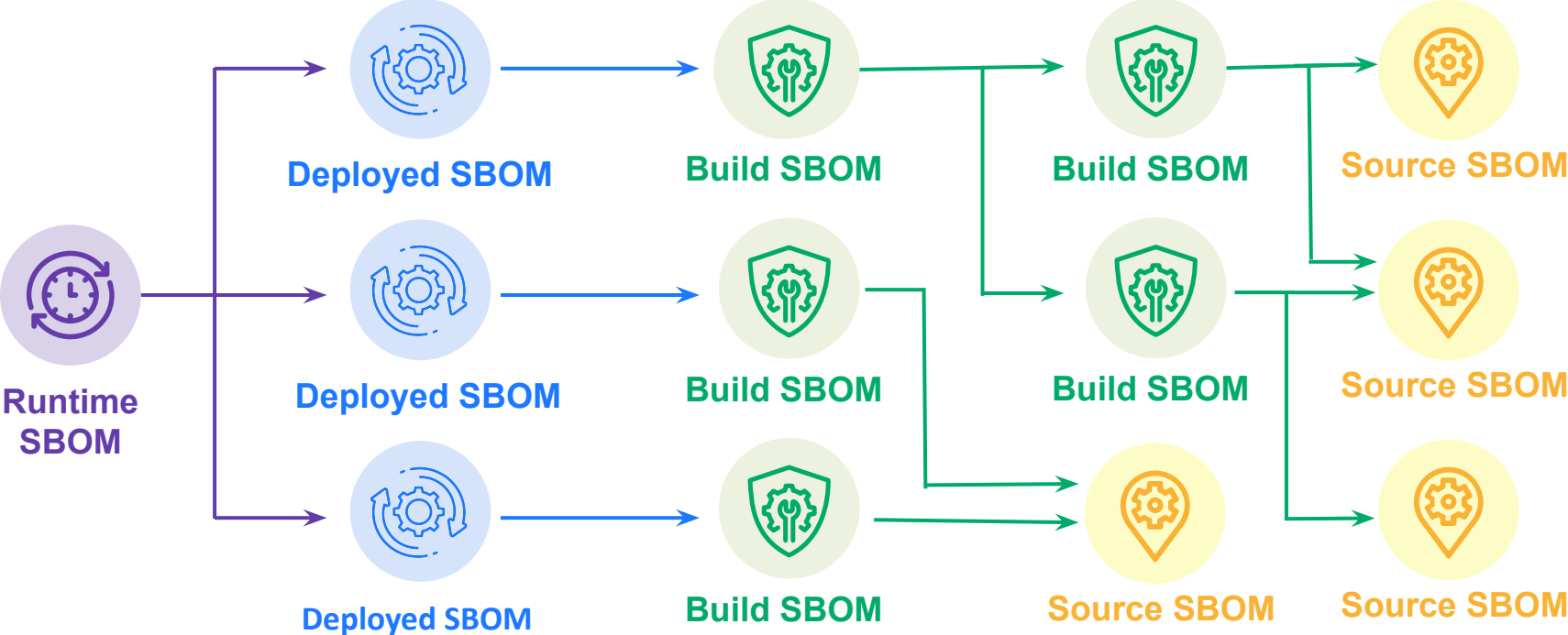
SBOM TYPE	DEFINITION
Design	SBOM of intended, planned software project or product with included components (some of which may not yet exist) for a new software artifact.
Source	SBOM created directly from the development environment, source files, and included dependencies used to build an product artifact.
Build	SBOM generated as part of the process of building the software to create a releasable artifact (e.g., executable or package) from data such as source files, dependencies, built components, build process ephemeral data, and other SBOMs.
Deployed	SBOM provides an inventory of software that is present on a system. This may be an assembly of other SBOMs that combines analysis of configuration options, and examination of execution behavior in a (potentially simulated) deployment environment.
Runtime	BOM generated through instrumenting the system running the software, to capture only components present in the system, as well as external call-outs or dynamically loaded components. In some contexts, this may also be referred to as an “Instrumented” or “Dynamic” SBOM.
Analyzed	SBOM generated through analysis of artifacts (e.g., executables, packages, containers, and virtual machine images) after its build. Such analysis generally requires a variety of heuristics. In some contexts, this may also be referred to as a “3rd party” SBOM.

Source: [Types of Software Bills of Materials \(SBOM\)](#) published by CISA on 2023/4/21

Understanding System: Traceability



Understanding Safety Critical System: Traceability

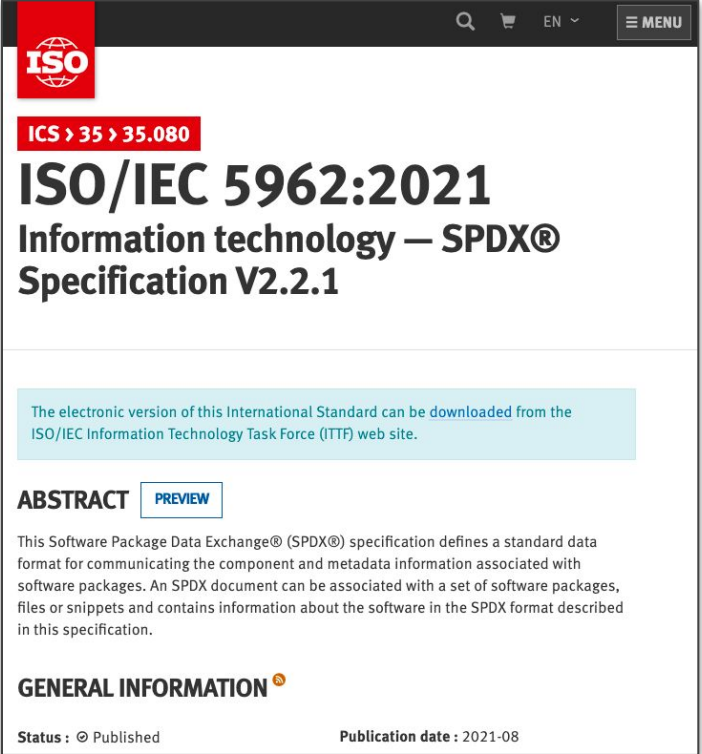


Risk Analysis for Products?

Evolving SPDX to provide the **framework for connecting** the metadata **beyond software** so that other components, processes, requirements and evidence can be made available to **support product line management**

ISO/IEC 5962:2021

- Able to represent SBOMs from binary images and track back to the source files and snippets.
- Specification is [freely available from ISO site](#).
- Future updates are live tracked at: <https://spdx.github.io/spdx-spec> and work on satisfying safety requirements is being included
- More information at spdx.dev



The screenshot shows the ISO/IEC 5962:2021 specification page. At the top, there is a navigation bar with the ISO logo, a search icon, a shopping cart icon, the language 'EN', and a 'MENU' button. Below the navigation bar, the breadcrumb 'ICS > 35 > 35.080' is displayed. The main title is 'ISO/IEC 5962:2021 Information technology – SPDX® Specification V2.2.1'. A light blue box contains the text: 'The electronic version of this International Standard can be downloaded from the ISO/IEC Information Technology Task Force (ITTF) web site.' Below this, there is a section for 'ABSTRACT' with a 'PREVIEW' button. The abstract text reads: 'This Software Package Data Exchange® (SPDX®) specification defines a standard data format for communicating the component and metadata information associated with software packages. An SPDX document can be associated with a set of software packages, files or snippets and contains information about the software in the SPDX format described in this specification.' At the bottom, there is a 'GENERAL INFORMATION' section with a small orange icon. The status is '© Published' and the publication date is '2021-08'.

Timeline of SPDX Evolution - Use Case by Use Case

I AM THE
Cavalry
Legislation



NTIA:
Software
Transparency
begins



3T-SBOM:
CONOPs for
Tool-to-Tool
SBOM



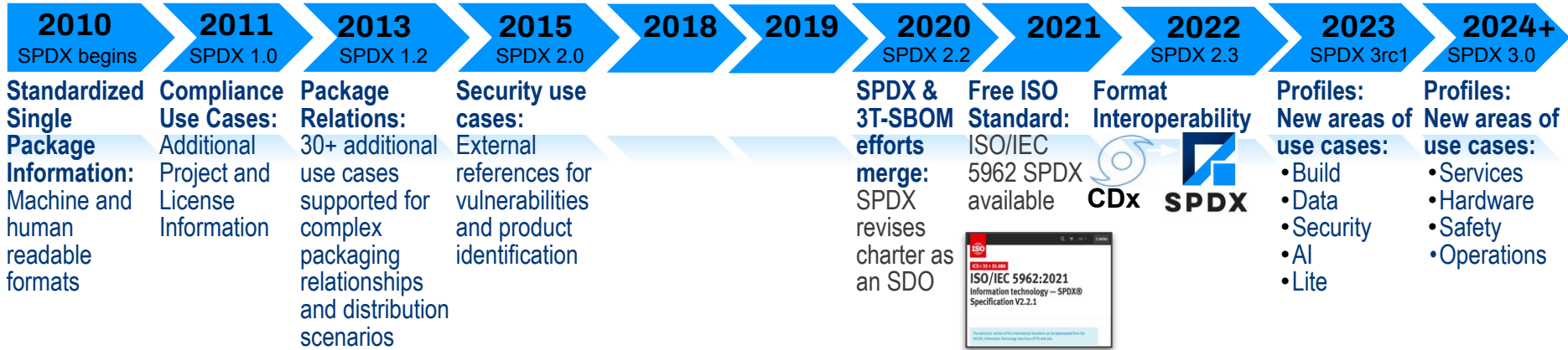
**Transition of
SBOM work
to DHS**



**Executive
Order 14028**



**EU Cyber
Resilience Act**



NTIA Software Bill Of Materials (SBOM) Guidance - Minimum Elements

Data Field	Description
Supplier Name	The name of an entity that creates, defines, and identifies components.
Component Name	Designation assigned to a unit of software defined by the original supplier.
Version of the Component	Identifier used by the supplier to specify a change in software from a previously identified version.
Other Unique Identifiers	Other identifiers that are used to identify a component, or serve as a look-up key for relevant databases.
Dependency Relationship	Characterizing the relationship that an upstream component X is included in software Y.
Author of SBOM Data	The name of the entity that creates the SBOM data for this component.
Timestamp	Record of the date and time of the SBOM data assembly.

SPDX 2.2 +
([ISO/IEC 5962:2021](https://www.iso.org/standard/72431.html))
supports all required minimum elements
(as well the optional that are mentioned in report)
and **many more use cases**

Checker available at:
<https://github.com/spdx/ntia-conformance-checker>

Source: https://www.ntia.gov/files/ntia/publications/sbom_minimum_elements_report.pdf

SPDX Evolution

SPDX 2.2+ ([ISO/IEC 5962:2021](#)) supports exchanging metadata between systems

- Software BOM metadata and relationships between components.
- Supports traceability between requirements, code, tests & evidence

SPDX 3.0 to supports **knowledge graph databases** for **product lines** more efficiently

- Supports product lifecycle metadata and incorporation of updates to remediate vulnerabilities
- Introduces profiles to capture domain specific metadata about components and their interactions at points in time
- Extends beyond software to capture **AI/ML model** and **dataset provenance**
- Import from suppliers and export to customers current state at point in time

SPDX 3.1 extend beyond software to support safety profile needs for “all ingredients”

- Work already in progress on Hardware, Services, Operations and Safety Profiles

SPDX 3.0 Profiles



Security information - vulnerability details related to software



Build related information - provenance and reproducible builds



Information about AI models - ethical, security, and model data



Information about datasets - AI and other data use cases



Minimal subset to support industry supply chain workflows



Information about copyrights and licenses - supports compliance



Information specific to software



Information used across all profiles

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]

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

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

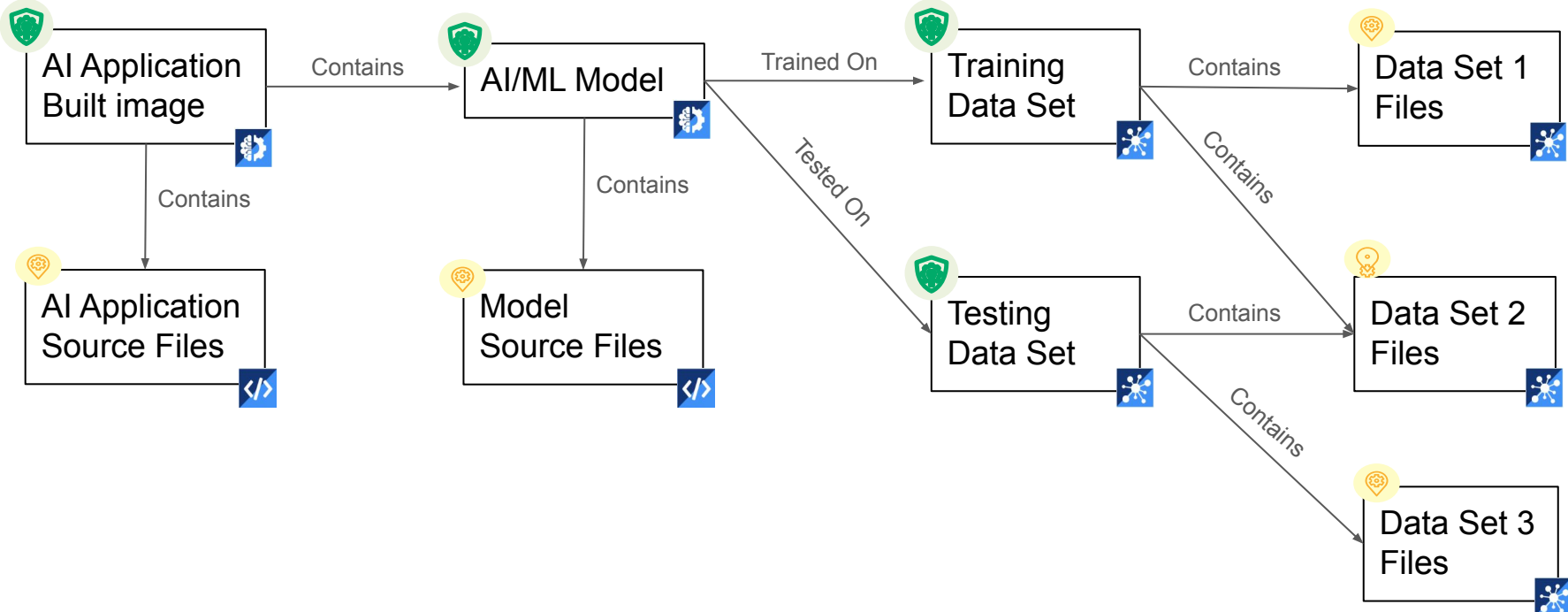
RelationshipType
Meta
describes [element->element]
amendedBy [element->element]
modifiedBy [element->element]
other [element->element] (comment)
Structure
contains [element->element]
Behavioral
configures [element->element]
delegatedTo [element->element]
dependsOn [element->element]
Pedigree
generates [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]

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

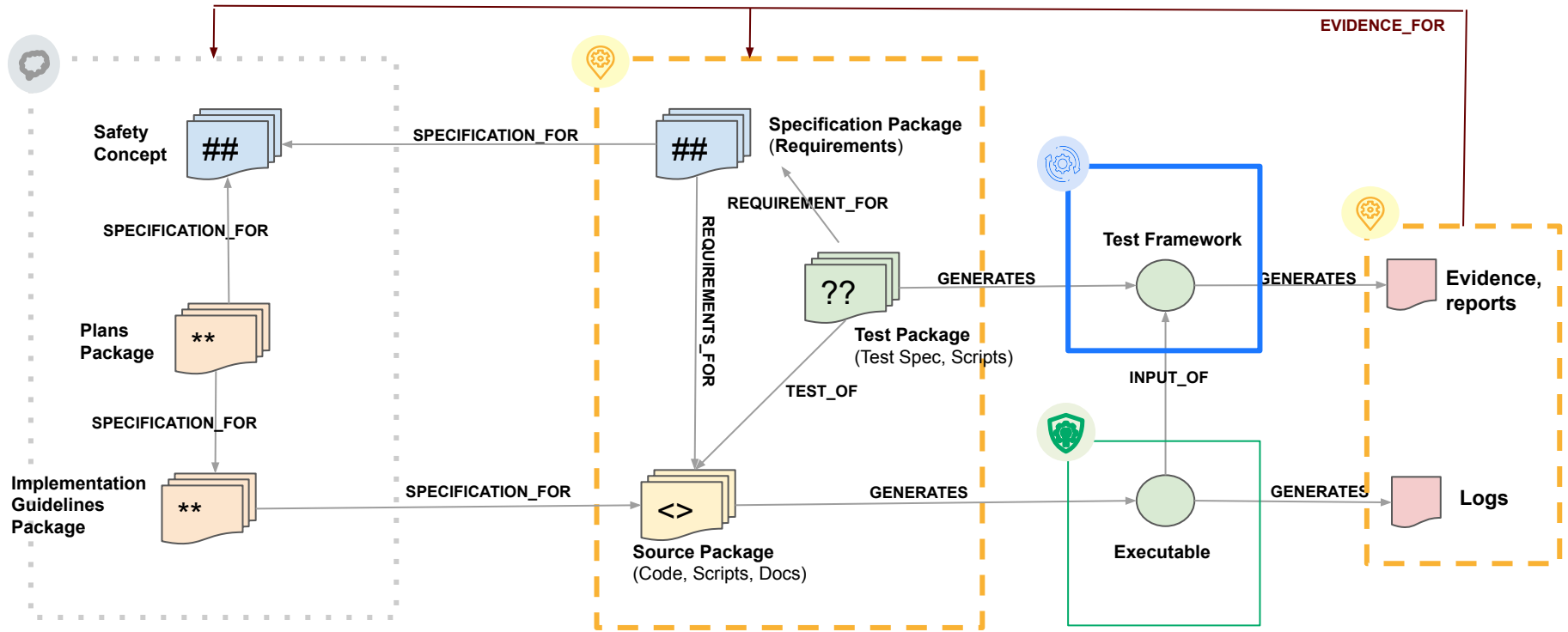
Representing AI Application as a Set of BOMs



Extending SPDX beyond 3.0 to “All the Ingredients”






- Extend to support **safety critical application** (including **Critical Infrastructure**) to satisfy **safety analysis requirements**
 - Capture **Requirements** and **Traceability** for code
 - Evolve **AI/ML** and **Datasets** increasing need for system transparency, we're just starting in SPDX 3.0
 - **Virtual and Physical Hardware** Support, connected versions of software, models & trained data. ⇒ **Digital Twin Support**
 - **External Services** increasing importance for support key functionality
- Extend to support efficient use of software components in organizations
 - **Operations** for the needs of business to do risk assessments

Supporting System Knowledge Graph Creation

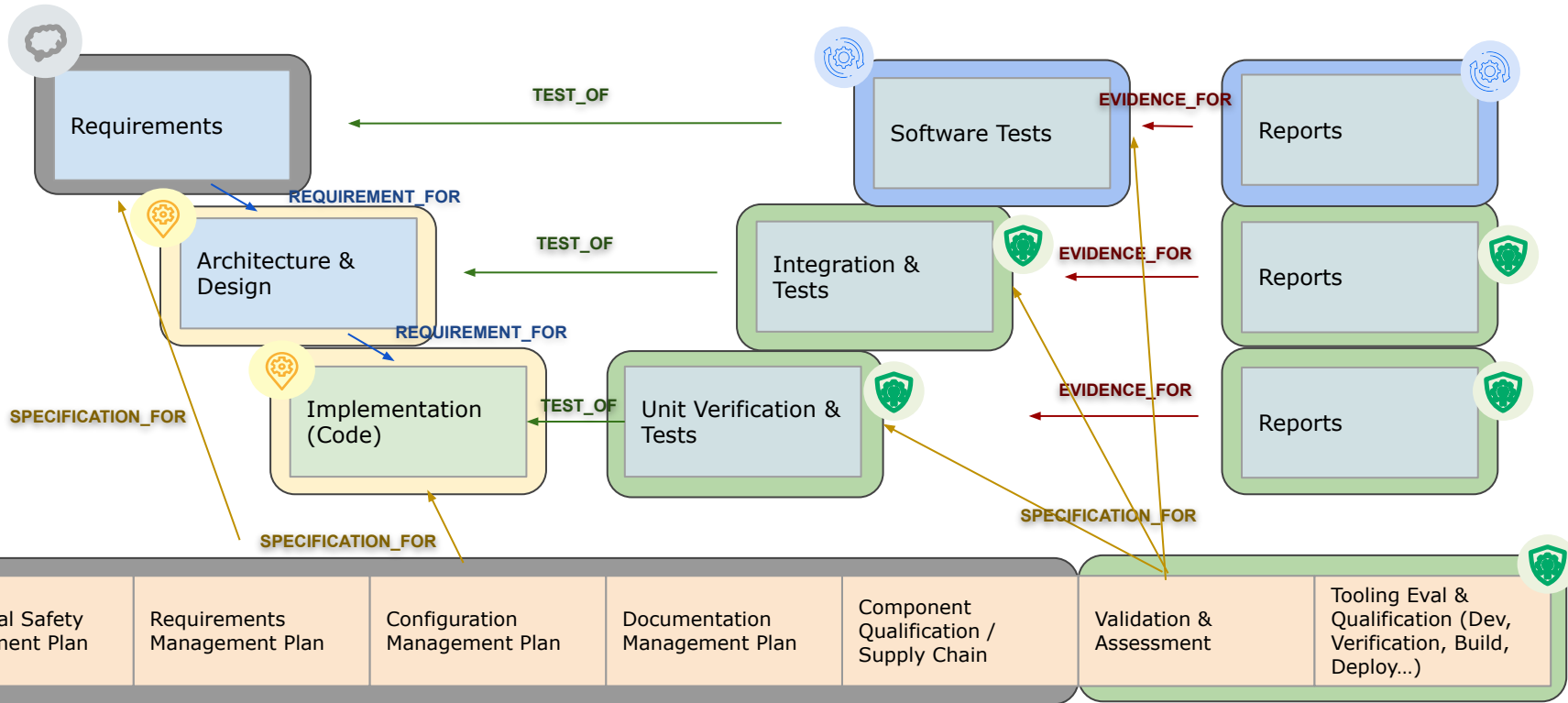


SPDX supports component metadata modularity and relationships between components, allows us to create a knowledge graph inside a database for accurate and efficient Safety & Security Analysis; as well as change management & updates

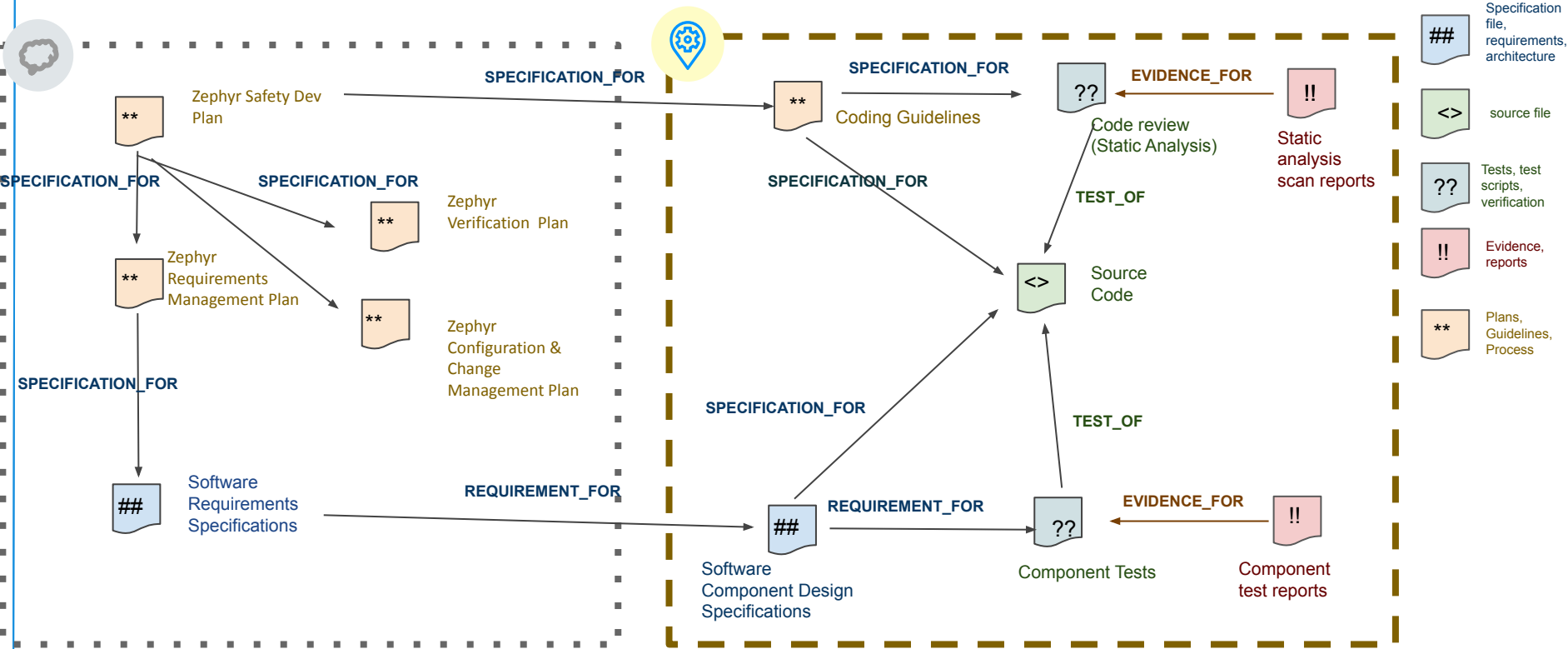
Align Safety Artifacts with SBOMs

 A grey circular icon containing a white speech bubble with a gear inside.	Design SBOM	Functional Safety Management (Plans) and Safety Concept
 A yellow circular icon containing a white gear with a location pin.	Source SBOM	Requirements, Design, Safety Analysis, Source Code, Test Cases
 A green circular icon containing a white shield with a gear inside.	Build SBOM	Build Framework, Build configuration and environment data, Test Framework, Executable, Test Reports
 A blue circular icon containing a white gear with a location pin.	Deploy SBOM	Deployed configuration and environment data, Hardware architecture specific information and data, deployment tests and reports
 A purple circular icon containing a white clock with a gear inside.	Runtime SBOM	Runtime relevant data (configuration data), training data, error logging data

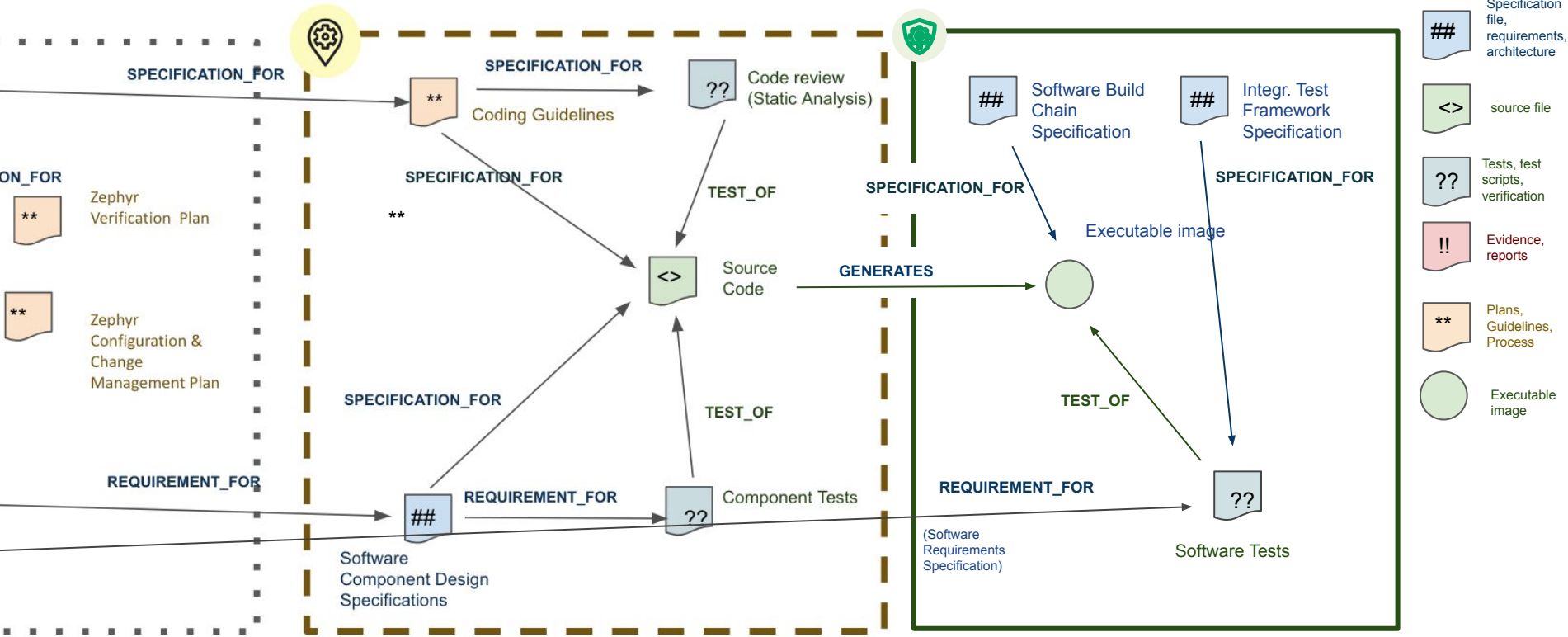
SPDX Style Dependencies in a FuSa Project



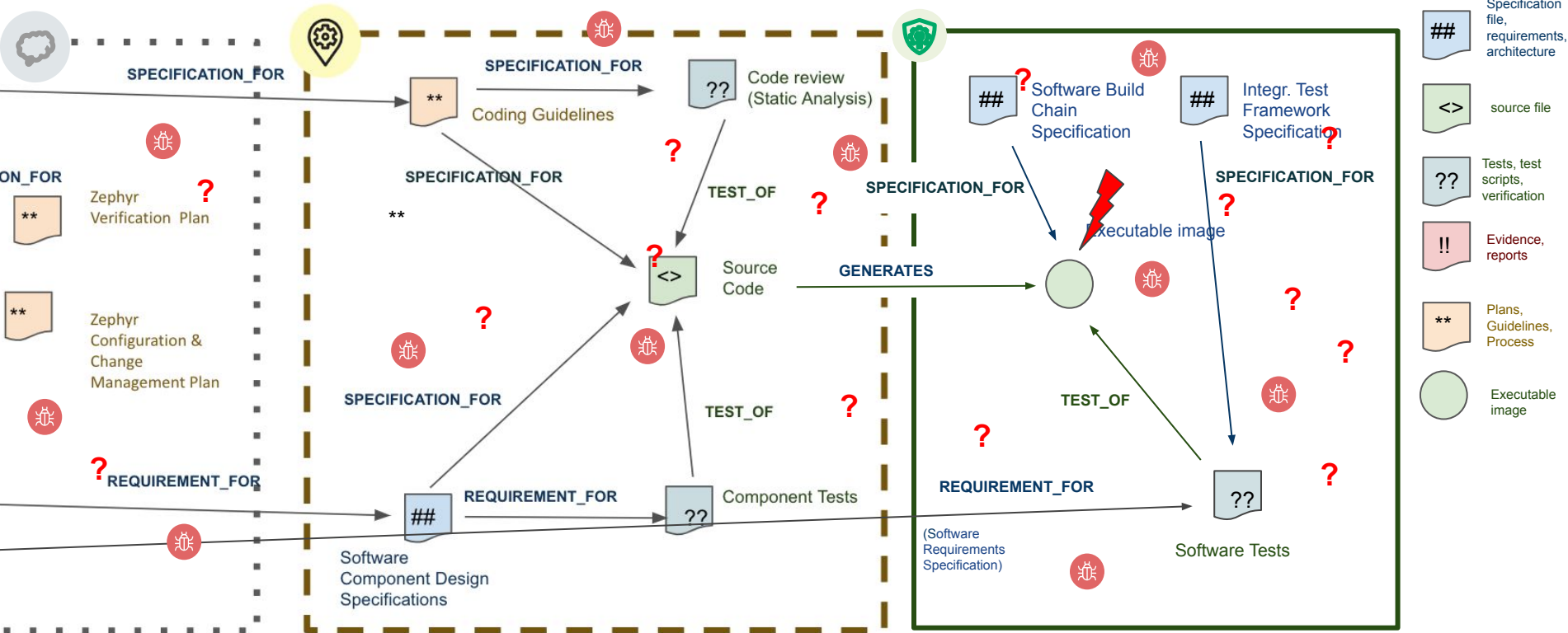
Design SBOM to Source SBOM



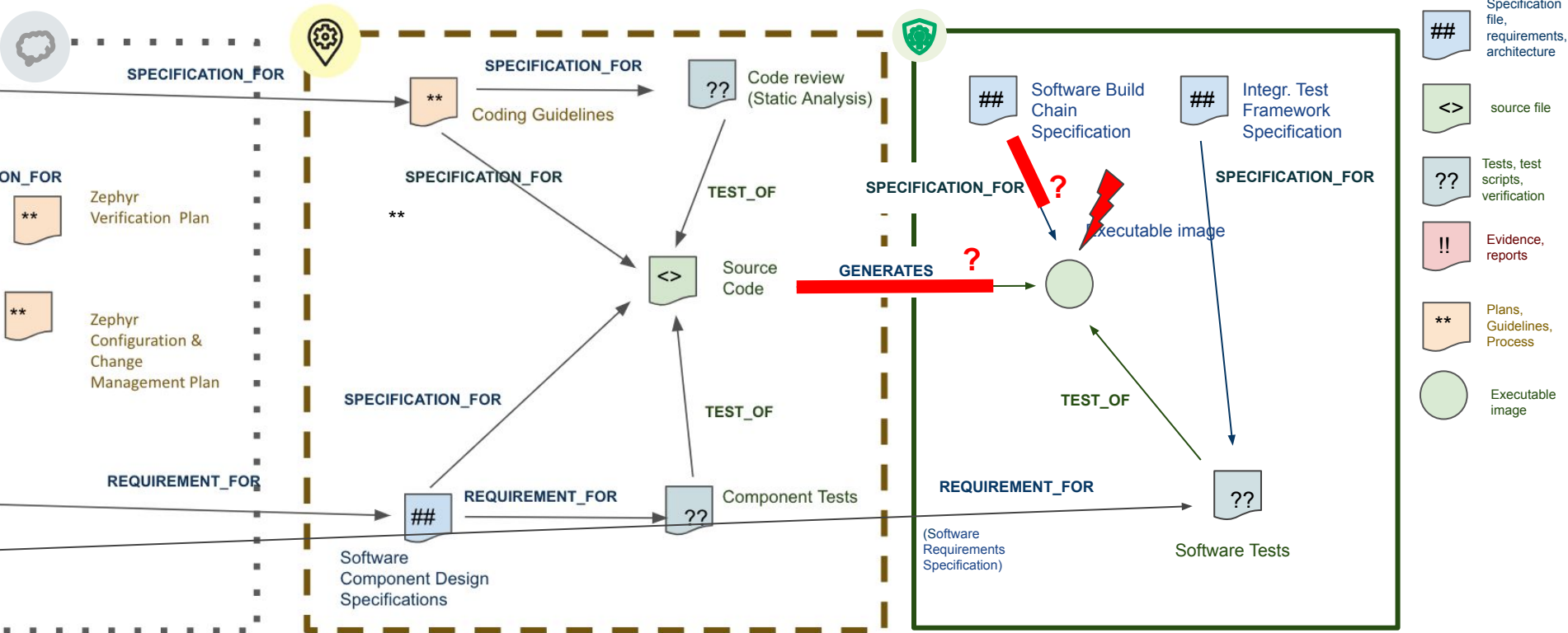
Source SBOM to Build SBOM



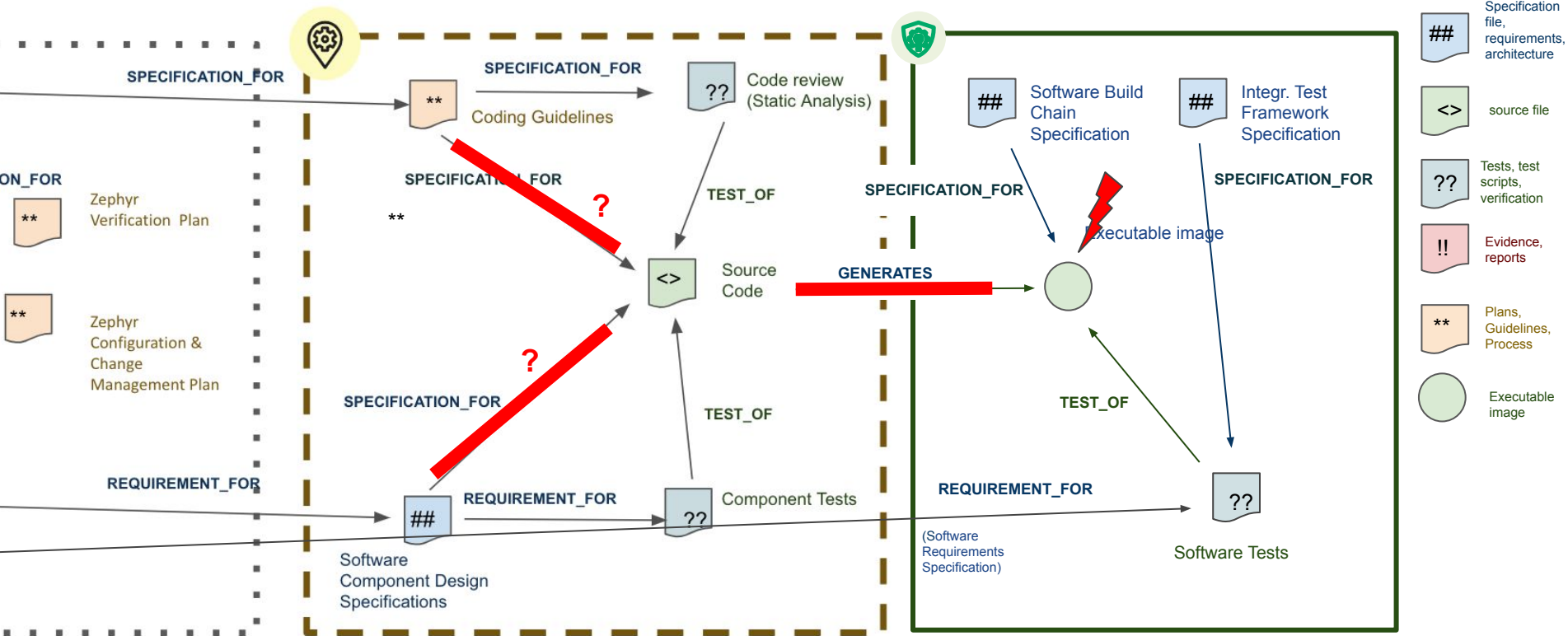
Dependency Identification between Components



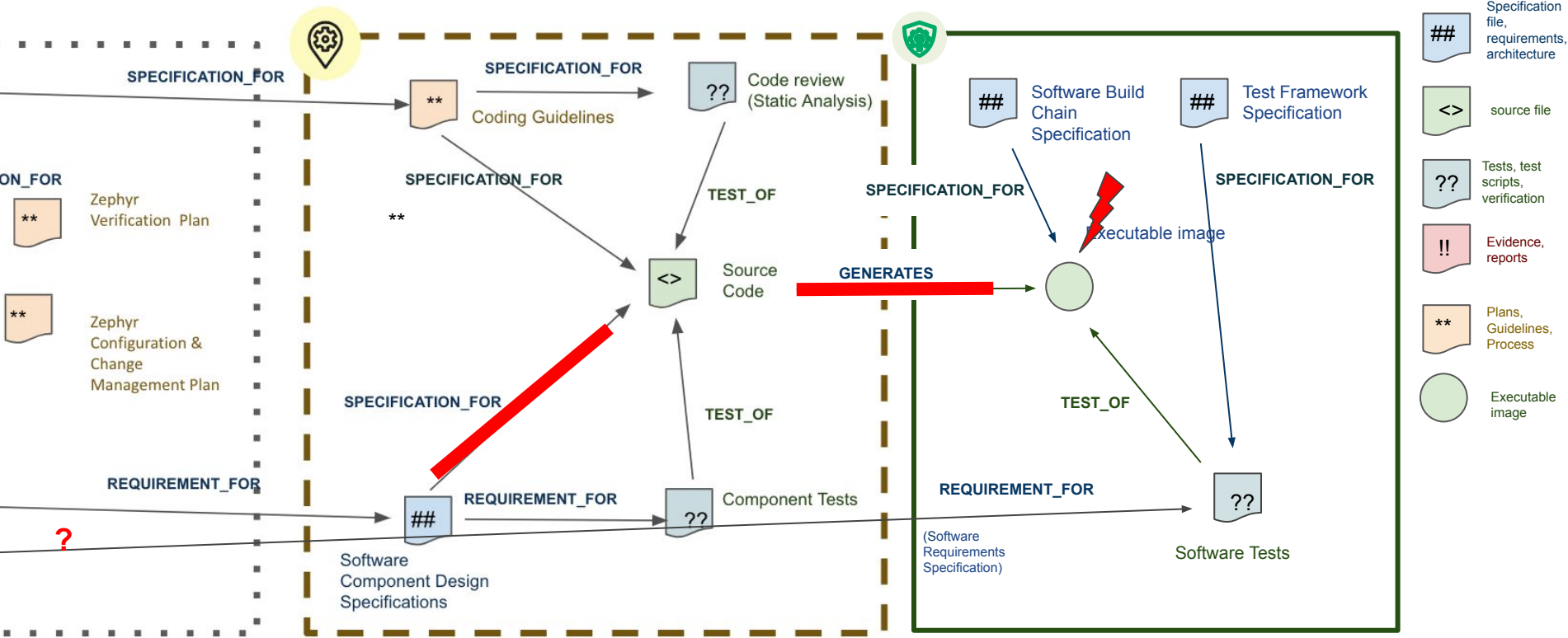
Dependency Identification at Component Level



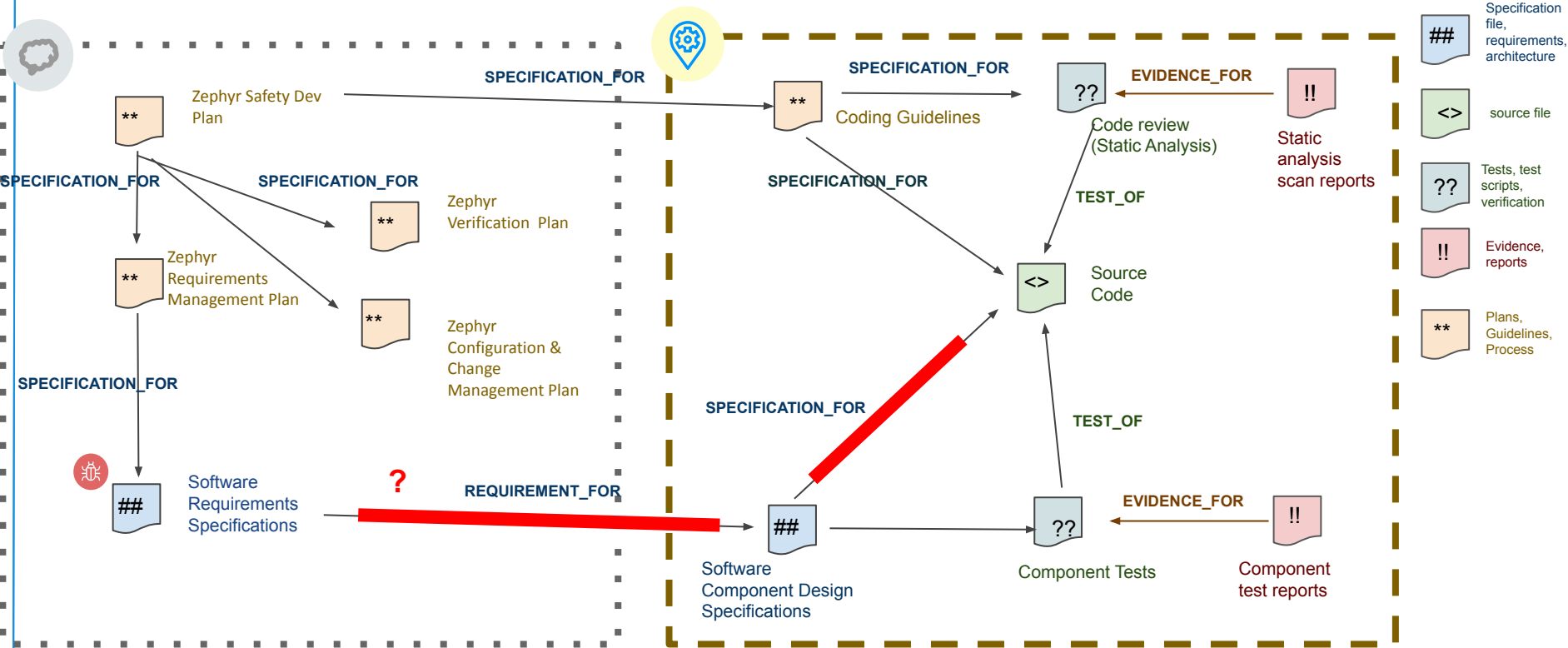
Dependency Identification at Component Level



Dependency Identification at Component Level

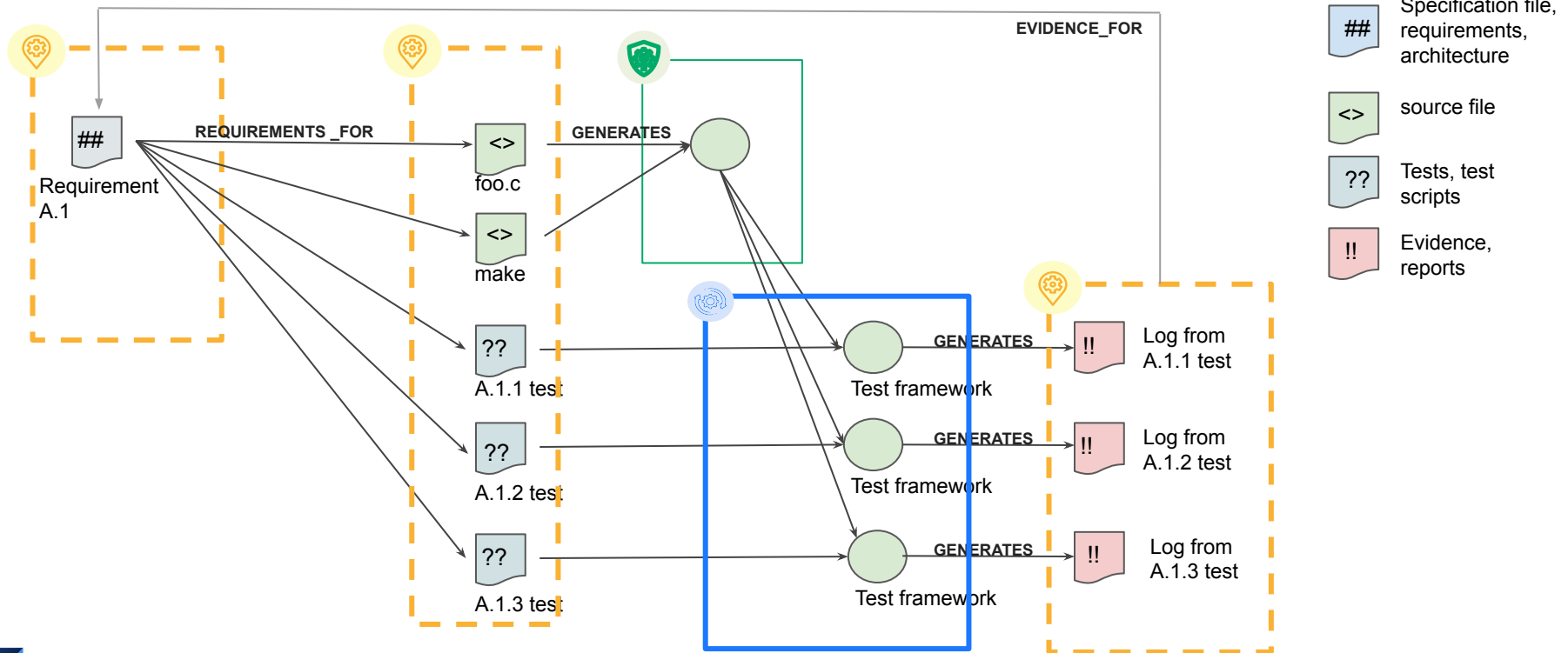


Dependency Identification at Component Level



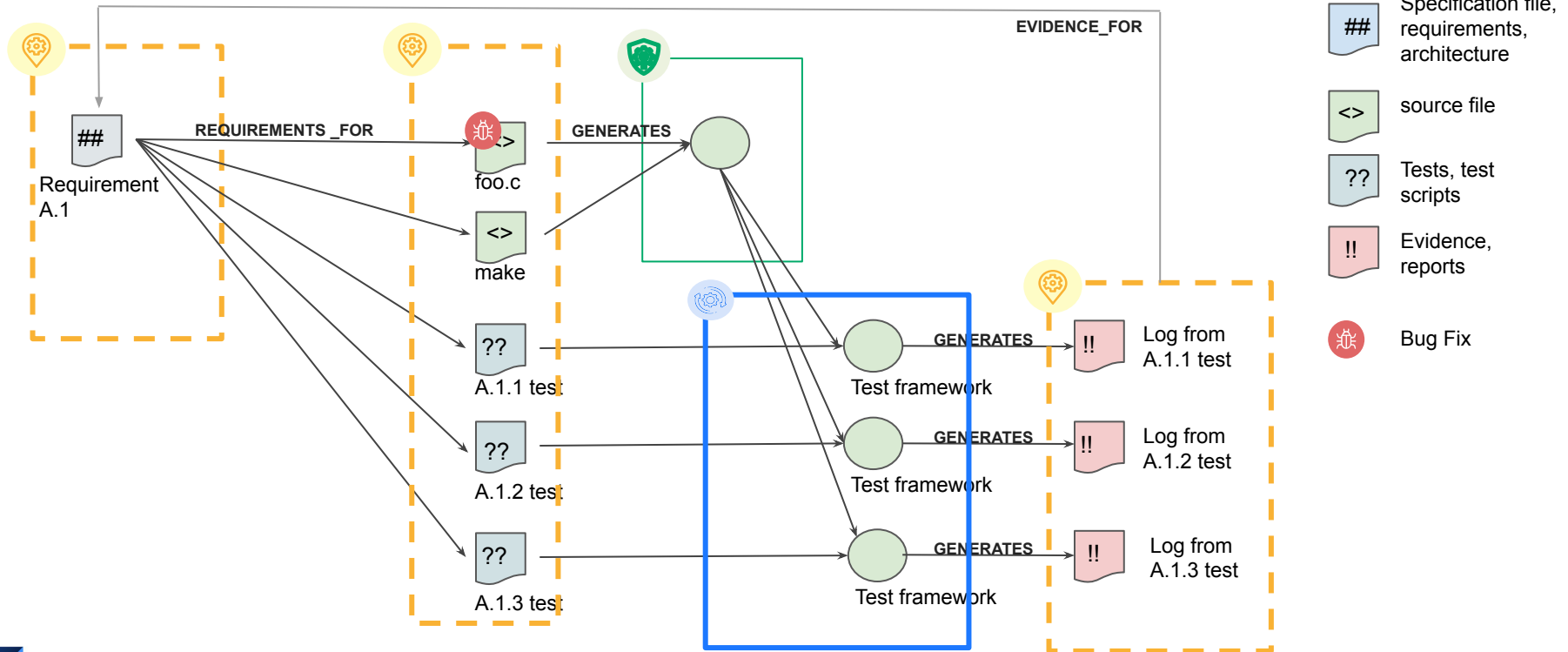
When needed: Traceability Inside Component

Requirement to Code to Tests to Evidence



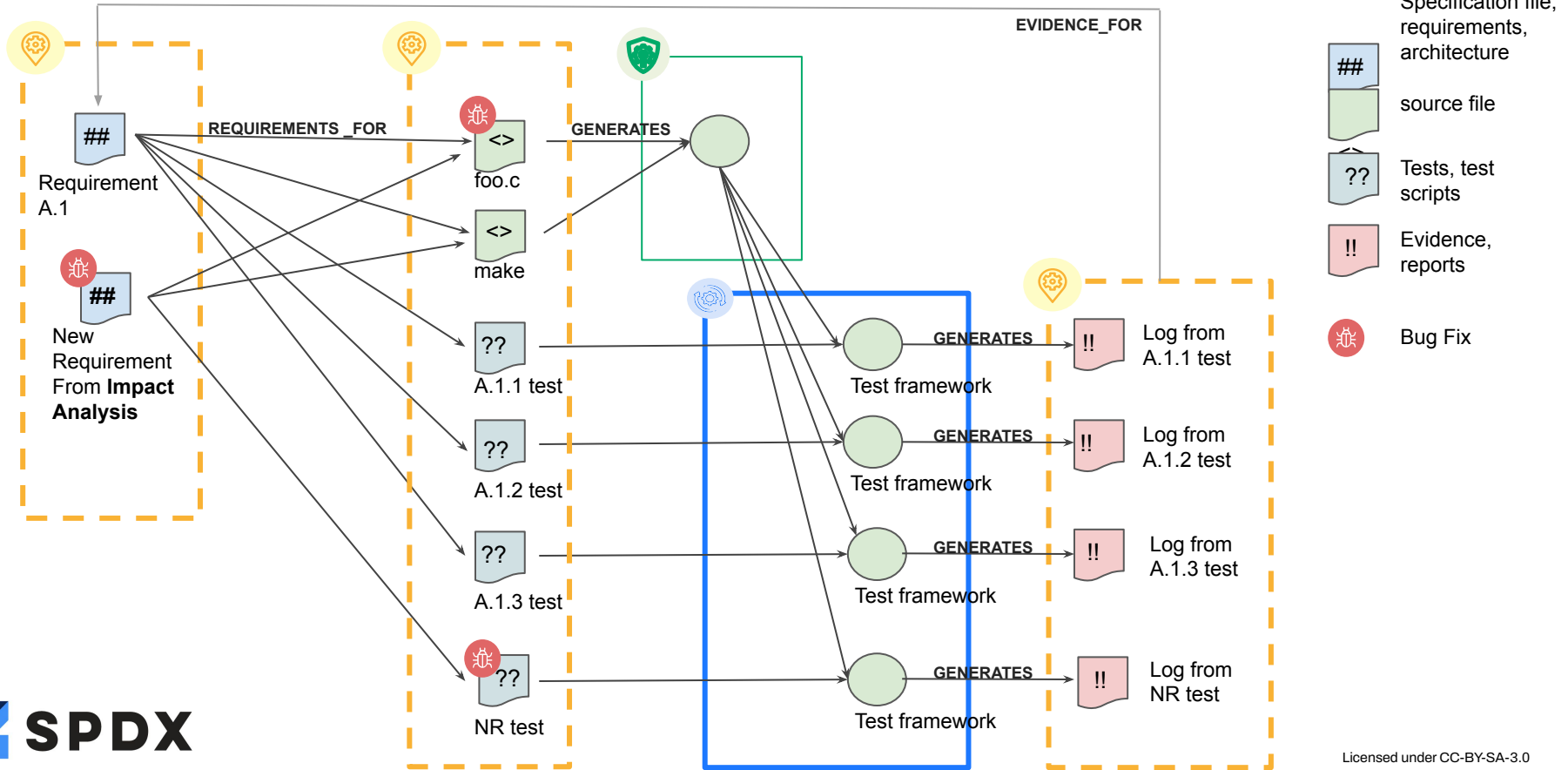
When needed: Traceability Inside Component

Requirement to Code to Tests to Evidence



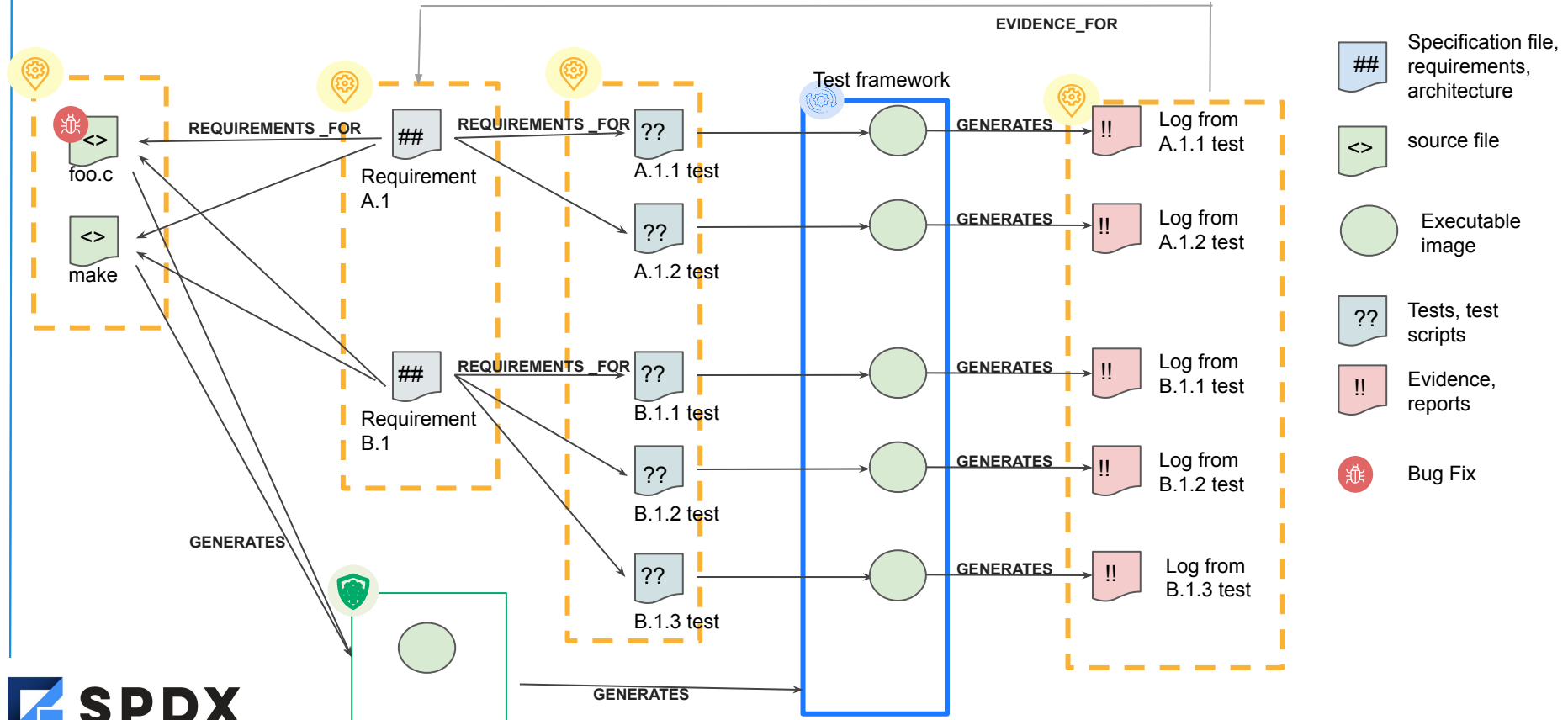
Traceability Inside Component

New Requirement to Code to Tests to Evidence



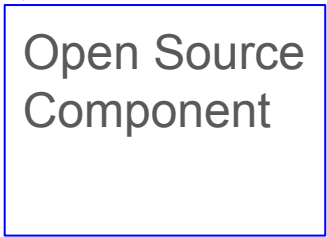
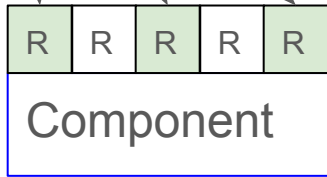
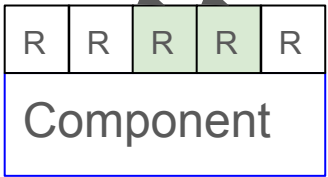
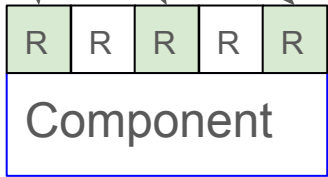
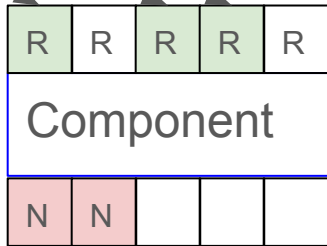
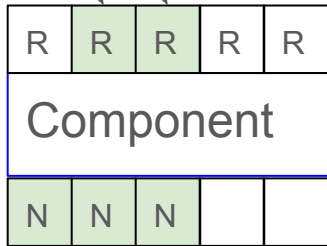
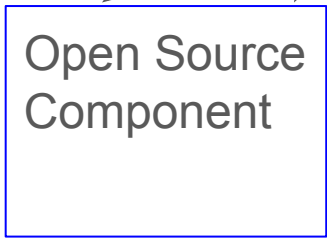
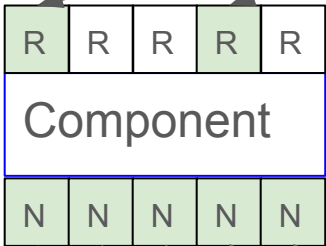
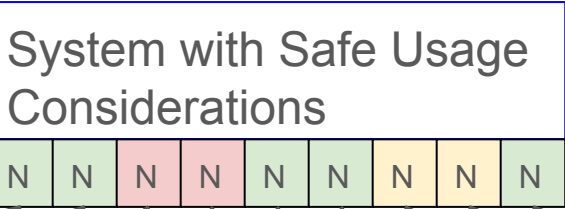
Inside Component: Traceability of Source to Requirements

Code to Requirements to Tests to Evidence



Requirements from Open Source?

How can we establish “Requirements” and “Traceability” for Open Source Components that the System Engineering & Safety Analysis need?



Projects Evolving to Support Functional Safety Analysis

Linux:



ELISA
Enabling **Linux** in
Safety Applications

RTOS:



Zephyr[™]

Virtualization/
Hypervisor:



yocto .
PROJECT

Reproducible Build Framework

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

ELISA Project Goals

- Support safety certification of **Linux-based systems** with a set of elements, processes and tools.
- Enable companies to incorporate the output of the project into products.
- The work is accepted by the open source community, safety community, regulation authorities, standardization bodies and system developers.
- Focus the project activities using a Linux-based reference system to safety-integrity standards.

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.

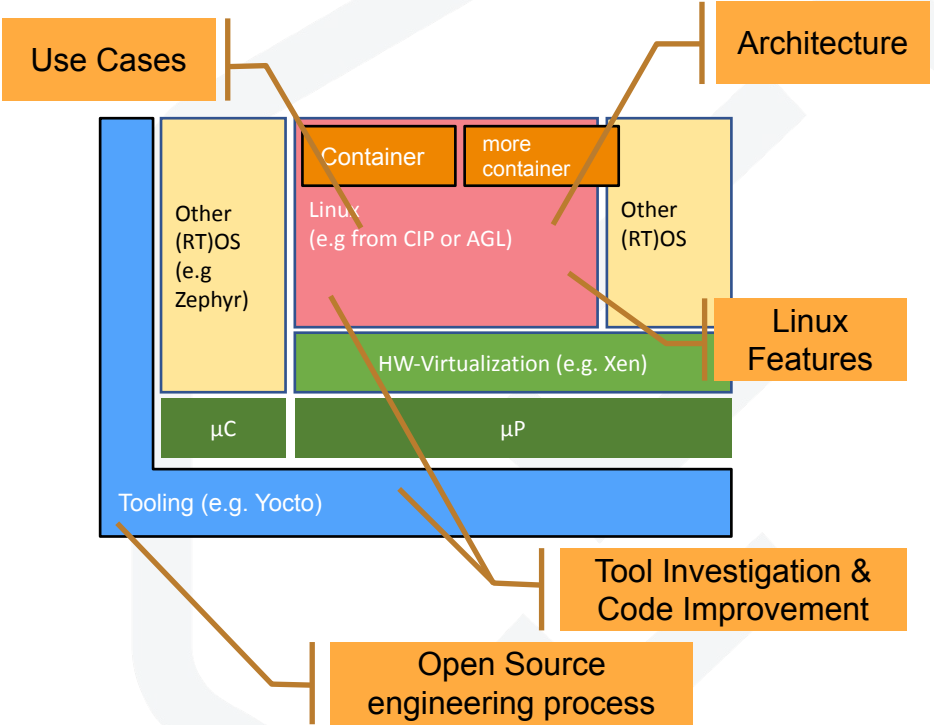
Activities:

- Working on systems to connect Linux with hypervisor and RTOS & explore implications of OSS projects interacting mixed criticality systems, prototyping SPDX Safety Profile
- Last year Illustrating Linux, Xen & Zephyr interacting with AGL; this year with Apertis

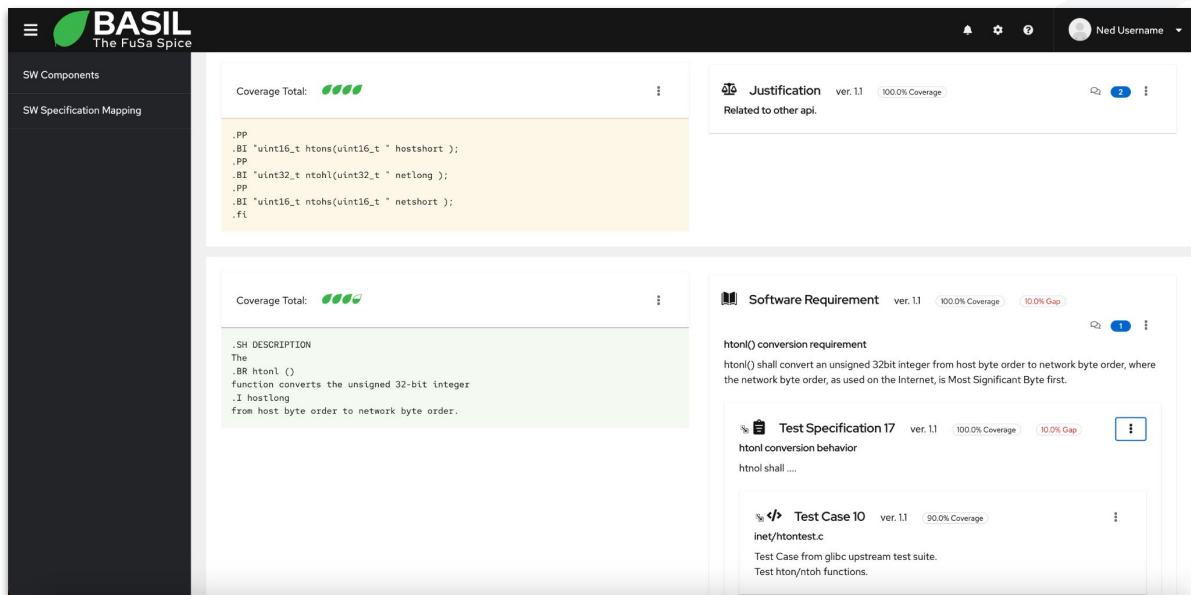


Reference Open Integration

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



New Open Source Requirements Tool: BASIL



The screenshot displays the BASIL web interface, titled "The FuSa Spice". The interface is divided into several sections:

- Left Sidebar:** Contains "SW Components" and "SW Specification Mapping".
- Top Panel:** Shows "Coverage Total: ■■■" and "Justification ver. 1.1 100.0% Coverage". Below this is a code block:

```
.PP
.BI "uint16_t htons(uint16_t  hostshort );
.PP
.BI "uint32_t ntohl(uint32_t  netlong );
.PP
.BI "uint16_t ntohs(uint16_t  netshort );
.ft
```
- Middle Panel:** Shows "Coverage Total: ■■■" and "Software Requirement ver. 1.1 100.0% Coverage 10.0% Gap". Below this is a description:

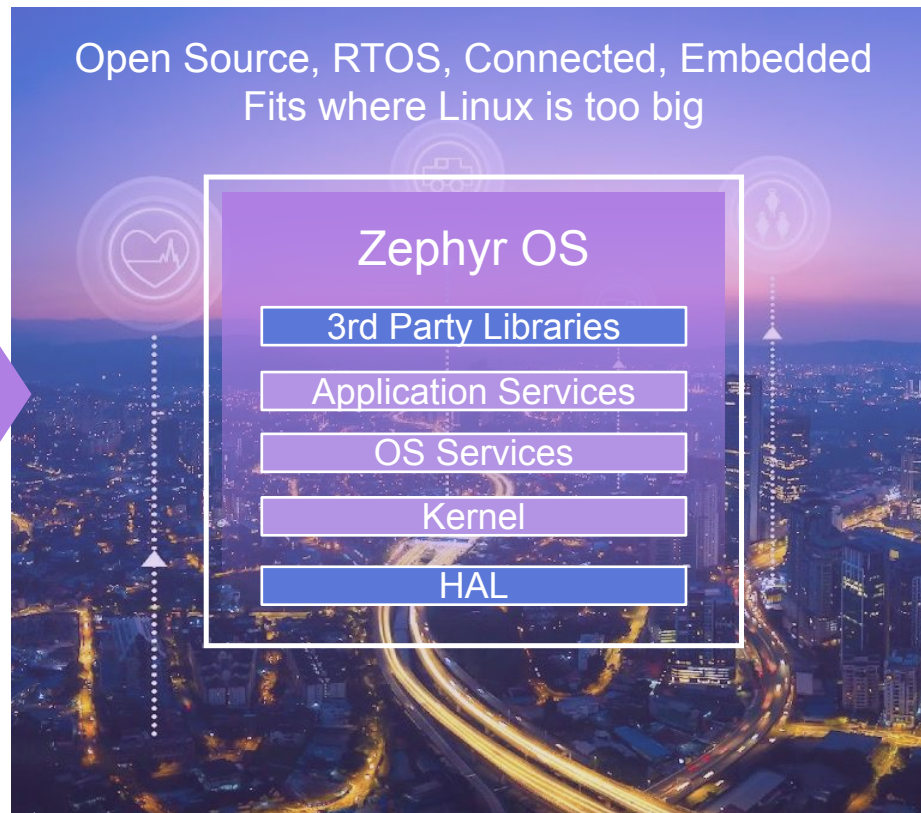
```
.SH DESCRIPTION
The
.BR htonl ()
function converts the unsigned 32-bit integer
.I hostlong
from host byte order to network byte order.
```
- Right Panel:** Shows "Test Specification 17 ver. 1.1 100.0% Coverage 10.0% Gap" and "Test Case 10 ver. 1.1 90.0% Coverage". Below this is a description:

```
inet/htontest.c
Test Case from glibc upstream test suite.
Test hton/ntoh functions.
```

Zephyr Project



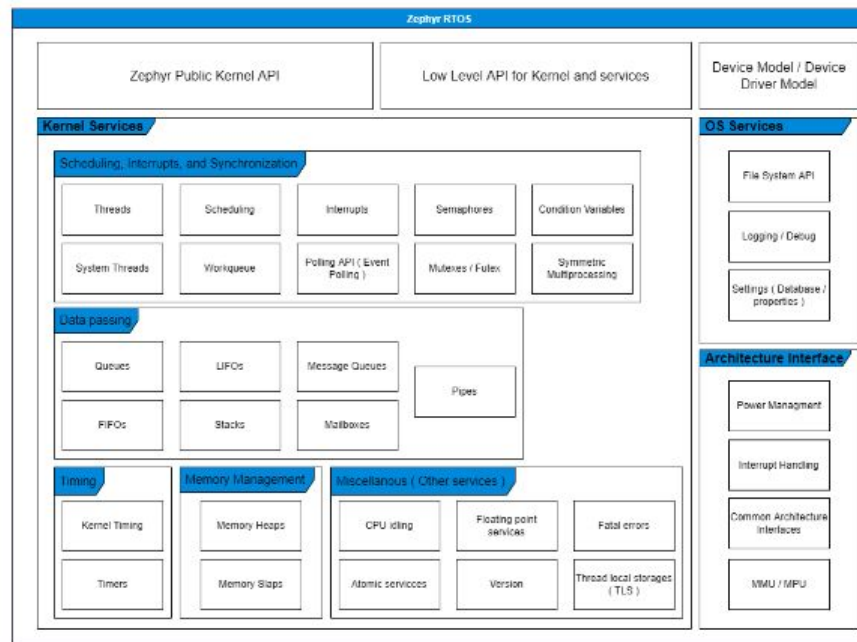
- **Open source** real time operating system
- **Developer friendly** with vibrant community participation
- Built with **safety and security** in mind
- **Broad SoC, board and sensor support.**
- **Vendor Neutral** governance
- **Permissively licensed** - Apache 2.0
- **Complete**, fully integrated, highly configurable, **modular** for **flexibility**
- **Product** development ready using LTS includes **security updates**
- **Certification** ready with Zephyr Auditable



Safety: Initial certification focus



- Start with a limited scope of kernel and interfaces
- Initial target is IEC 61508 SIL 3 / SC 3 (IEC 61508-3, 7.4.2.12, Route 3s)
- Option for 26262 ASIL D certification has been included in contract with certification authority should there be sufficient member interest



Scope can be **extended** to include **additional components** with associated **requirements** and **traceability** as determined by the safety committee

Current requirements work



- Used tooling: StrictDoc (<https://github.com/strictdoc-project/strictdoc>)
- Hierarchical structure of requirements that works for the project
- Capturing the requirements in StrictDoc which is working towards import/export of SPDX

A screenshot of a GitHub repository interface. The top part shows the file explorer for the repository 'stanislaw / reqmgmt', with the file 'zephyr_02_functional_requirements.sdoc' selected. Below the file explorer, the content of the file is displayed in a code editor. The code is a StrictDoc document for 'Zephyr Functional Requirements'. It includes a title, grammar, and a requirement. The requirement is a functional requirement for a math library, with a title 'Math library' and a statement 'Zephyr shall support floating point math libraries for processors where floating point is available'. The requirement is linked to a parent requirement 'ZEP-CLIB-001' and has a user story and discussion date. The bottom part of the screenshot shows the rendered HTML output of the StrictDoc file, which is a structured document with sections for 'Requirement', 'User Story', and 'Discussion Date'.

```
1 [DOCUMENT]
2 TITLE: Zephyr Functional Requirements
3
4 [GRAMMAR]
5 ELEMENTS:
6 - TAG: REQUIREMENT
7   FIELDS:
8   - TITLE: UID
9     TYPE: String
10  - REQUIRED: False
11  - TITLE: STATUS
12    TYPE: String
13  - REQUIRED: False
14  - TITLE: TYPE
15    TYPE: String
16
17
18 [REQUIREMENT]
19 UID: ZEP-CLIB-003
20 STATUS: Draft
21 TYPE: Functional
22 COMPONENT: C Library
23 REFS:
24 - TYPE: Parent
25   VALUE: ZEP-CLIB-001
26 TITLE: Math library
27 STATEMENT: >>>
28 Zephyr shall support floating point math libraries for processors where floating point is available
29 <<<<
30 USER_STORY: >>>
31 https://github.com/zephyrproject-rtos/zephyr/blob/main/lib/libc/minimal/include/math.h
32 <<<<
33 DISCUSSION_DATE: >>>
34 20221122.0
35 <<<<
```

Xen Project

- Bring the power of **Open source virtualization** everywhere
- **Trusted by leading organizations** including cloud providers and data centers
- Reliable and future proof, Xen is trusted for it's **performance and maturity for the past 20 years**
- Designed for **scalability and efficiency**
- **Industry leading security** protects against threats and vulnerabilities
- **Vendor neutral** governance
- Vibrant and **collaborative developer community**
- Xen Project working towards **safety certification**
- Licensed under **GNU General Public License (GPL) V2**



Xen Support

Today:

- Xen is chosen for safety critical applications due to its maturity and robust security features
- Can be configured to provide real-time scheduling for VMs
- Allows critical tasks to run within predefined time constraints

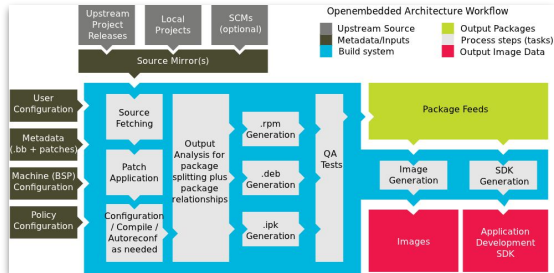
Work in Progress:

- Improve Xen coding style with MISRA-C
- Implement features to improve real-time and reduce interference
- Project members working on getting Xen safety certified for 61508 & 26262
- Using [OpenFastTrace](#) for requirements tracking and aligning to use SPDX



Builds customised Linux/open source distribution in a maintainable way

Codename	Series Version	Originally Released	Latest Release	Status	Downloads	Release Notes
Stylhead	5.1			Active Development		
Scarthape	5.0	April 2024	5.0.0 / April 2024	LTS until Apr. 2028	Download	Release Notes
Kirkstone	4.0	April 2022	4.0.17 / March 2024	LTS until Apr. 2026	Download	Release Notes
Dunfell	3.1	April 2020	3.1.33 / April 2024	LTS until Apr. 2024	Download	Release Notes



- Layer structure to isolate/contain and stack customisation
- Can build Linux/RTOS/firmware and combine
- 100% reproducible binaries down to timestamps
- Allow ease of update to latest components
- Releases every 6 months
- LTS release every 2 years with 4 year lifespan
- Well established independent community, no vendor lock in
- Extensive automated testing
- Tools to assist with legal obligations & vulnerability analysis
 - Software license manifests
 - SPDX software bill of materials(SBOMs) automatically
 - CVE analysis

Yocto Features Supporting Dependable System Creation

Reproducible Builds:

- Binary identical images including timestamps
- Can reproduce an image in X years time identical to today
- Daily automated testing to prove it:

<https://www.yoctoproject.org/reproducible-build-results/>

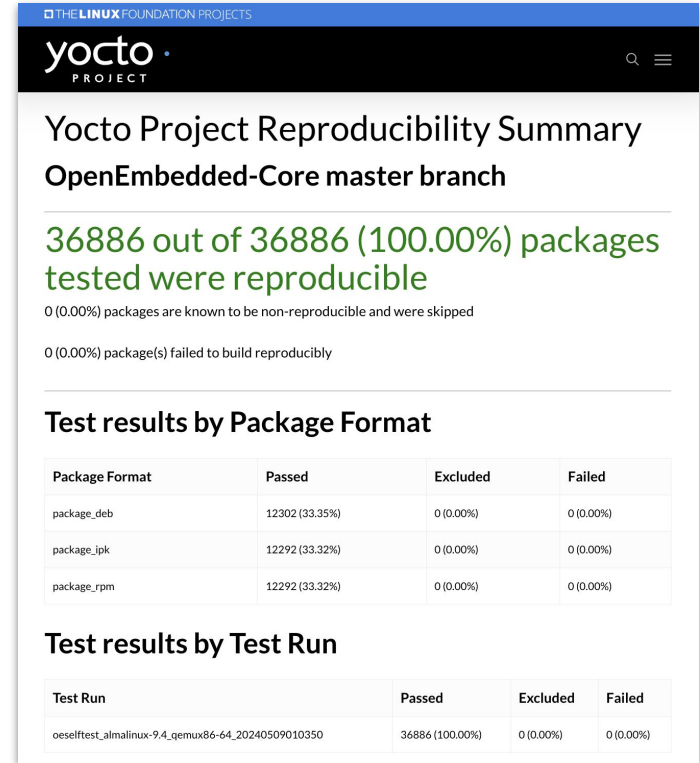
Automated testing:

- Covers multiple architectures, C libraries, init systems
- Runs upstream test suites of components including toolchain (gcc, glibc)
- 5.0 QA process included 3,364,120 individual tests:

<https://downloads.yoctoproject.org/releases/yocto/yocto-5.0/testreport.txt>

Software manifests:

- Provide SBOM with configuration option
- Allow CVE analysis (present and future)
- Enable license compliance
- Modelling and leading industry best practice
- Aims to meet current and future legal requirements
- Based upon public/community standards (SPDX 2.x and 3.0)



THE LINUX FOUNDATION PROJECTS

yocto PROJECT

Yocto Project Reproducibility Summary

OpenEmbedded-Core master branch

36886 out of 36886 (100.00%) packages tested were reproducible

0 (0.00%) packages are known to be non-reproducible and were skipped

0 (0.00%) package(s) failed to build reproducibly

Test results by Package Format

Package Format	Passed	Excluded	Failed
package_deb	12302 (33.35%)	0 (0.00%)	0 (0.00%)
package_ipk	12292 (33.32%)	0 (0.00%)	0 (0.00%)
package_rpm	12292 (33.32%)	0 (0.00%)	0 (0.00%)

Test results by Test Run

Test Run	Passed	Excluded	Failed
oeselftest_almaLinux-9.4_qemux86-64_20240509010350	36886 (100.00%)	0 (0.00%)	0 (0.00%)

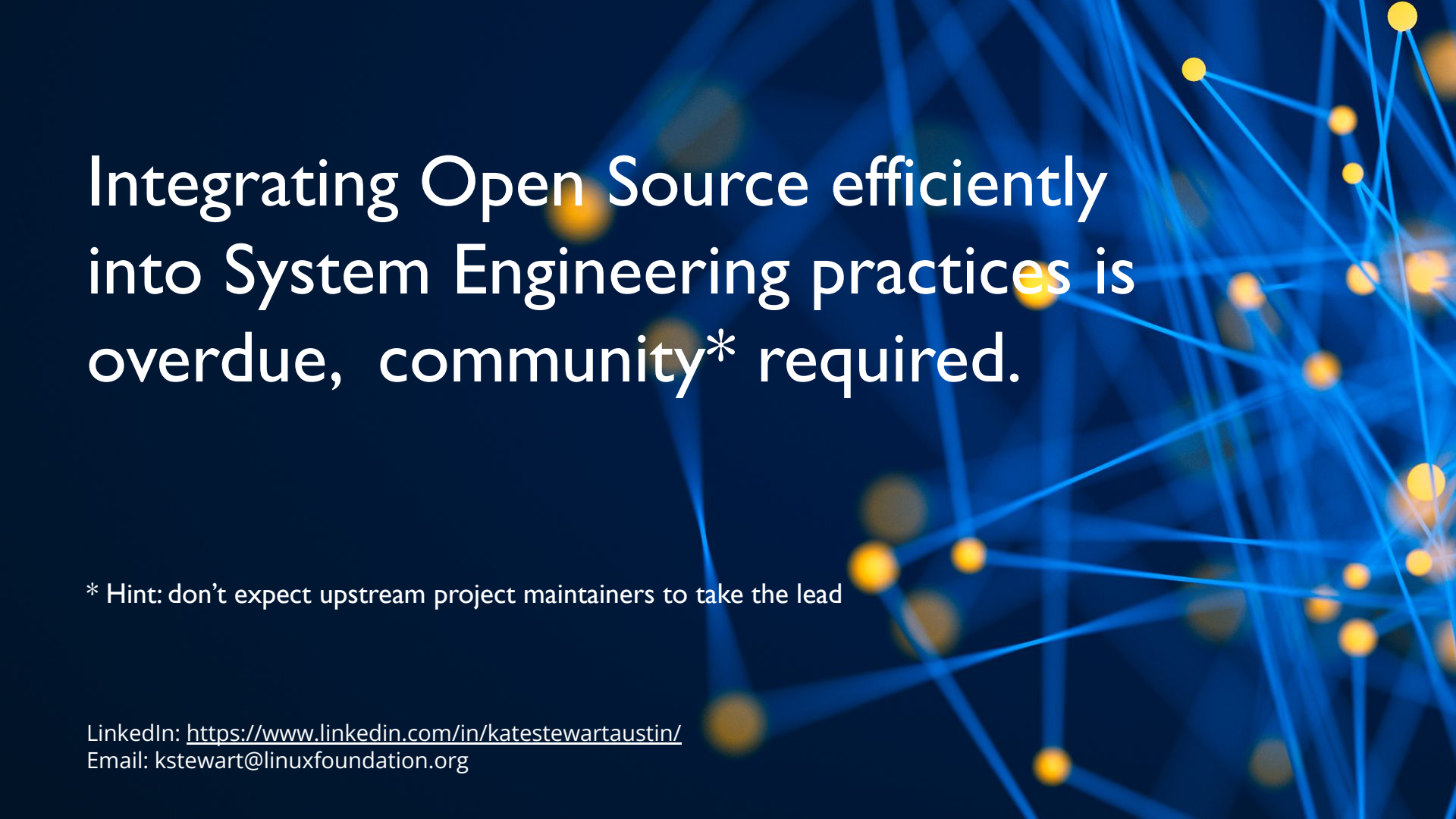
Next steps to continue the discussion?

Augmenting open source components:

- **Linux:** join in [ELISA working groups](#)
- **Zephyr:** join in the [safety](#) working group
- **Xen:** join the [FuSa](#) special interest group
- **Yocto:** join the build & release communities

Framework for connecting “All the Ingredients”:

- **SPDX:** join the Functional Safety(FuSa) profile group [meetings](#) and/or [mailing list](#)

A background image featuring a network diagram with blue lines connecting yellow nodes, set against a dark blue gradient.

Integrating Open Source efficiently
into System Engineering practices is
overdue, community* required.

* Hint: don't expect upstream project maintainers to take the lead

LinkedIn: <https://www.linkedin.com/in/katestewartAustin/>
Email: kstewart@linuxfoundation.org

Backup Slides

TCCoE

Conference: TCCOE Summit - May 9-10

Title: Building Dependable Embedded Systems with Open Source Components

Name of Presenter: Kate Stewart, VP Dependable Embedded Systems, The Linux Foundation

Short bio of the presenter: Kate works with the safety, security and license compliance communities to advance the adoption of best practices into embedded open source projects. Since joining The Linux Foundation, she has launched the ELISA and Zephyr Projects, as well as supporting other embedded projects. With more than 30 years of experience in the software industry, she has held a variety of roles in software development, architecture, and product management, primarily in the embedded ecosystem. She has presented on SBOMs, embedded systems and more, at industry conferences like RSA Conference, IoT World, Embedded World, Open Source Summit among others.

Abstract:

Systems are no longer created from monolithic code bases, they are composed of components that are integrated over time, and maintained by different entities. Yet for a system to be dependable, they all need to be integrated together and tested as updates occur to demonstrate they still adhere to the necessary requirements. Open Source projects are increasingly being used as the components in these systems. Effective system engineering depends on requirements being tested for the system as a whole and for the components, however open source projects frequently don't have requirements expressed in a form that is consumable. This talk will look at a proposed framework for a system bill of materials that will enable those components providing requirements to be integrated so that product lines can be managed, and those open source components able to surface up their requirements can be integrated.