SPDX SBOMs: Enabling Automation of Safety & Security Analysis

Kate Stewart
VP Dependable Embedded Systems, The Linux Foundation
kstewart@linuxfoundation.org
Software is Used in Critical Systems Today

- Chemical
- Communications
- Dams
- Emergency Services
- Financial
- Government Facilities
- Information Technology
- Transportation Systems
- Commercial Facilities
- Critical Manufacturing
- Defense Industrial Base
- Energy
- Food & Agriculture
- Healthcare & Public Care
- Nuclear Reactors, Materials, & Waste
- Water & Wastewater Systems

Source: https://www.cisa.gov/critical-infrastructure-sectors
Critical Infrastructure Today: Mix of Open & Proprietary

98% Percent of general codebases and Android apps that contained OSS
[Synopsys2021]

70% Percent of codebase that was OSS on average
[Synopsys2020]

Source:
https://www.synopsys.com/content/dam/synopsys/aig-assets/reports/2020-ossra-report.pdf
[Synopsys2021] "2021 Open Source Security and Risk Analysis Report" by Synopsys

Source: [Synopsys2021]
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

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

- **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)

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 enables recording and connecting the sources, assessments, vulnerabilities & patches, build & calibration data, tests, requirements and evidence ⇒ path to automation
Common Understanding of “SBOM”

“An SBOM is a formal record containing the details and supply chain relationships of various components used in building software.

These components, including libraries and modules, can be open source or proprietary, free or paid, and the data can be widely available or access-restricted.”

Source: NTIA's SBOM FAQ
# NTIA SBOM Guidance

<table>
<thead>
<tr>
<th>Minimum Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data Fields</strong></td>
</tr>
<tr>
<td><strong>Automation Support</strong></td>
</tr>
<tr>
<td><strong>Practices and Processes</strong></td>
</tr>
</tbody>
</table>

## 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) supports all required minimum elements (as well the optional that are mentioned in report)**

Checker available at: [https://github.com/spdx(ntia-conformance-checker](https://github.com/spdx(ntia-conformance-checker))

When should an SBOM be created or consumed?

Safety and Security expect that Configuration Management (CM) information will be maintained throughout the software lifecycle.

SBOMs provide a mechanism to track key artifacts and dependencies as well as other useful CM information.

Image derived from: NTIA's Survey of Existing SBOM Formats and Standards
## SBOM Types

<table>
<thead>
<tr>
<th>SBOM TYPE</th>
<th>DEFINITION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design</td>
<td>SBOM of intended, planned software project or product with included components (some of which may not yet exist) for a new software artifact.</td>
</tr>
<tr>
<td>Source</td>
<td>SBOM created directly from the development environment, source files, and included dependencies used to build an product artifact.</td>
</tr>
<tr>
<td>Build</td>
<td>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.</td>
</tr>
<tr>
<td>Deployed</td>
<td>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.</td>
</tr>
<tr>
<td>Runtime</td>
<td>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.</td>
</tr>
<tr>
<td>Analyzed</td>
<td>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.</td>
</tr>
</tbody>
</table>

KEY: Generate SBOMs **when** the data is available

- Source SBOM
- Design SBOM
- Runtime SBOM
- Build SBOM
- Deployed SBOM

SOFTWARE LIFECYCLE
Understanding System: Traceability
Understanding Safety Critical System: Traceability

- Runtime SBOM
- Deployed SBOM
- Build SBOM
- Source SBOM
Managing a security fix:
Customer & Integrator need to check Safety Profile

Customer Security

Search through Deployed SBOM (to Build SBOM and maybe to Source SBOM) to determine if impacted.

➔ If source is not included in build, or not reachable via configurations, document and no further action required.
➔ Else do impact analysis and determine mitigation.

Integrator

Get ‘Product’ update from Integrator with new Build SBOM, and confirm Safety Profile (when applicable) has been assessed.

Customer Procurement

Create New Deployed SBOM to document new ‘Product’ installed and confirm Safety Profile (when applicable) and record new Deployed SBOM

Customer Operations

NVD

If indicator of compromise, request mitigation from integrator if needed

See VEX/VDR from Integrator or Vulnerability Database

New Deployed SBOM

New Build SBOM

Change monitoring to use new Build & Deployed SBOMs for monitoring

THE LINUX FOUNDATION
SPDX SBOM’s Enable Linking: Requirements to Code to Tests to Evidence

- App Source Files, Tests Framework, Build Options
- Design, Documentation, Requirements
- Log Files, Evidence
- Build Tool
- App Executable, Build Configurations
- App Deployed Image, Calibrations

THE LINUX FOUNDATION
Software Package Data Exchange (SPDX®) specification is a standard for communicating the component and metadata information associated with software.

Charter: To create a set of data exchange standards that enable companies and organizations to share human-readable and machine-processable software package metadata to facilitate software supply chain processes.
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 [ITTF site](https://www.iso.org/standard/81870.html) accessed on 2021/11/19

- Future updates are live tracked at: [https://spdx.github.io/spdx-spec](https://spdx.github.io/spdx-spec) and work on satisfying safety requirements is being included

- More information at [spdx.dev](https://spdx.dev)
The Software Package Data Exchange® (SPDX®) Specification Version 2.3

Copyright © 2010-2022 Linux Foundation and its Contributors. This work is licensed under the Creative Commons Attribution License 3.0 Unported (CC-BY-3.0) reproduced in its entirety in Annex J herein. All other rights are expressly reserved.


source: https://spdx.github.io/spdx-spec/v2.3/
SPDX Continuously Improves

- **2010/02** - specification drafting began in a work-group of FOSSBazaar under Linux Foundation that came to be called "SPDX", was originally referred to as Package Facts.
- **2010/08** - "SPDX" announced as one of the pillars of the Linux Foundation's Open Compliance Program.
- **2011/08** - SPDX 1.0 specification - handles packages.
- **2012/08** - SPDX 1.1 specification - fixed flaw in verification algorithm
- **2013/10** - SPDX 1.2 specification - improved interaction with license list, additional fields for documenting project info.
- **2015/05** - SPDX 2.0 specification - added ability to handle multiple packages, relationships between packages and files, annotations.
- **2016/08** - SPDX 2.1 specification - added snippets, support for associating packages with external reference sources of information about packages, using SPDX License identifiers in files
- **2019/06** - SPDX 2.1.1 - conversion of specification from google docs to github as repository
- **2020/05** - SPDX 2.2 - Includes SPDX-lite
- **2020/08** - SPDX 2.2.1 prepared for submission to ISO.
- **2021/08** - ISO/IEC 5962 available
- **2022/08** - SPDX 2.3 published to improve interoperability with other formats
- **2023/Q2** - SPDX 3.0 release candidate and prototyping in progress ...
Formal Model Enables Validation & Interchange Between Specific File Formats
## SPDX Relationships Clarify Dependency Types

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
<th>Type</th>
<th>Additional Term</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESCRIBES</td>
<td>DESCRIBES</td>
<td>DEPENDENCY_OF</td>
<td>PREREQUISITE_FOR</td>
</tr>
<tr>
<td>DESCRIBED_BY</td>
<td>DESCRIBED_BY</td>
<td>RUNTIME_DEPENDENCY_OF</td>
<td>HAS_PREREQUISITE</td>
</tr>
<tr>
<td>CONTAINS</td>
<td>CONTAINS</td>
<td>BUILD_DEPENDENCY_OF</td>
<td>ANCESTOR_OF</td>
</tr>
<tr>
<td>CONTAINED_BY</td>
<td>CONTAINED_BY</td>
<td>DEV_DEPENDENCY_OF</td>
<td>DESCENDENT_OF</td>
</tr>
<tr>
<td>DYNAMIC_LINK</td>
<td>DYNAMIC_LINK</td>
<td>OPTIONAL_DEPENDENCY_OF</td>
<td>DOCUMENTATION_OF</td>
</tr>
<tr>
<td>STATIC_LINK</td>
<td>STATIC_LINK</td>
<td>PROVIDED_DEPENDENCY_OF</td>
<td>BUILD_TOOL_OF</td>
</tr>
<tr>
<td>AMENDS</td>
<td>AMENDS</td>
<td>TEST_DEPENDENCY_OF</td>
<td>EXPANDED_FROM_ARCHIVE</td>
</tr>
<tr>
<td>COPY_OF</td>
<td>COPY_OF</td>
<td>OPTIONAL_COMPONENT_OF</td>
<td>DISTRIBUTION_ARTIFACT</td>
</tr>
<tr>
<td>DEPENDS_ON</td>
<td>DEPENDS_ON</td>
<td>DEPENDENCY_MANIFEST_OF</td>
<td>GENERATED_FROM</td>
</tr>
</tbody>
</table>

For more details see: [https://spdx.github.io/spdx-spec/v2.3/relationships-between-SPDX-elements/](https://spdx.github.io/spdx-spec/v2.3/relationships-between-SPDX-elements/)
SPDX Generation Tooling

- **tools-java**
  - Aug 12, 2022 - v1.1.0 update to support 2.3
  - Sept 2022 → Feb 2023 5 releases for performance improvements & fixes

- **tools-python**
  - OpenSSF Funded cleanup & restructuring
  - Dec 8, 2022 - v0.7.0 - update to support 2.3 & clean up bug backlog
  - Prototyping of 3.0 in progress

- **tools-golang**
  - Jan 12, 2023 - v0.4.0 - update to support 2.3

- **spdx-online-tools** (validator & translator)
  - Aug 12, 2022 - v1.0.7 add support SPDX 2.3
  - Nov 15, 2022 - v1.0.9 add in NTIA Conformance Checker Tool
SPDX Consumption Tooling

- **spdx-online-tools** (validator & translator)
  - Aug 12, 2022 - v1.0.7 add support SPDX 2.3
  - Nov 15, 2022 - v1.0.9 add in [NTIA Conformance Checker](#) Tool

- **SPDX-to-OSV** (vulnerability lookup)
  - Produce an Open Source Vulnerability JSON file based on information in an SPDX document
  - Jan 10, 2022 - v0.1.1 - pick up tooing updates

- **ntia-conformance-checker** (minimum SBOM fields present)
  - Check that an SBOM meets the minimum field requirements
  - Started as GSOC project, and maintainer from Chainguard has adopted
  - Feb 8, 2022 - v0.2.1 - fix NOASSERTION supplier case

Also worth looking at: [https://github.com/nyph-infosec/daggerboard](https://github.com/nyph-infosec/daggerboard)
SBOMs Everywhere in 2022...

- OpenSSF - Work Stream 9 → **SBOM Everywhere SIG**
  - SPDX Python Library rework funded.
    - [Test suite](#) and [release candidate available now](#)
  - Started work on consolidating definitions of types of SBOMs → CISA working group to get broader adoption
  - Started documentation of use cases

- NTIA efforts have transferred to US DHS CISA
  - 4 working groups
    - SBOM Sharing, SBOM Adoption, SBOM Cloud, SBOM Tooling
  - International coordination with CERTs (like Japan’s CERT) and other international government agencies
Why the Changes for SPDX 3.0?

- **Additional Use Cases**
  - AI and Data
  - Security and Defect information

- **Simplify**
  - Profiles
  - Remove confusing names

- **Flexibility**
  - Can communicate a single Element
  - Enhanced relationship structure with less relationship types (work in progress)
SPDX 3.0 - Increases Modularity

<table>
<thead>
<tr>
<th>Core Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Profile</td>
</tr>
<tr>
<td>Licensing</td>
</tr>
</tbody>
</table>

SPDX 3.0 (Core Model + Software Profile + Licensing Profile) == SPDX 2.3
SPDX 3.0 Core Model Permits Extensions
SPDX 3.0 Software Profile
SPDX 3.0 Specification Infrastructure

Specification is being transformed into markdown describing
- Classes, Properties, Enumerations
- Metadata (type & cardinality) and description for each element.
- Will be able to automatically generate schema from this version (for JSON, YAML, RDF, XML, tag-value, etc.) and reduce errors.

Profiles can add their own Classes and Properties and may also restrict other profiles (e.g. values, cardinalities, ...)

See: https://github.com/spdx/spdx-3-model
Licensing Profile Update

- Based on licensing-related fields in pre-existing SPDX spec, with updates:
  - More consistency across artifact types (package, file, snippet)
  - Aligning with SPDX 3.0 data model
  - Documenting the object model for license expressions

- Current status:
  - Fields were discussed in joint tech/legal calls in late 2020 and early 2021
  - Initial starting point draft shared in Mar. 2021
  - Revised earlier draft for new SPDX 3.0 model formatting
  - Initial draft included in main at: https://github.com/spdx/spdx-3-model/tree/main/model/Licensing

For more information, contact: Steve Winslow or Alexios Zavras
Security Profile Update

- Communicating vulnerabilities in software
  - Associate vulnerabilities with specific elements, like packages
  - Conveying vulnerability assessment (severity, impact, exploitability)

- Linking to external security information
  - securityAdvisory
    - Advisories and miscellaneous security related document
    - Common Security Advisory Framework (CSAF)
    - CycloneDX formatted security information
    - Open Source Vulnerability (OSV) document
    - Vulnerability Disclosure Report (VDR per NIST EO 14028)
  - securityFix
    - Code fix or patch for a security issue
  - securityOther
    - Any unspecified type of security information

- VEX support to assert status of a vulnerability

- Details at: https://github.com/spdx/spdx-3-model/tree/main/model/Security

For more information contact: Thomas Steenbergen, Jeff Schutt or Rose Judge
Build Profile Overview

Use Cases:
- Security
- Reproducibility
- Auditing quality/pedigree of build
- Safety
  - The source code at the time of release
  - The configuration used to build the software
  - The specific versions of the tools used to build the software

Producers: Build systems, secondarily, analysis tools

Initial draft has been submitted to model for discussion at:
https://github.com/spdx/spdx-3-model/tree/main/model/Build

Contact: Brandon Lumm or Nisha Kumar
AI BOM ⇒ SPDX AI profile + SPDX Dataset profile

Traditional Software

AI Software

Has additional elements that nuances that need to be captured to ensure its traceability

AI BOM

Enhances SPDX to describe AI software including and AI Software's components, licenses, copyrights, and security references.

AI components

Software components

AI Package Profile

Dataset profile

Spdx AI profile

AI BOM

SPDX AI profile + SPDX Dataset profile
Components of AI Application

Source: IBM Data Science Best Practices
Build Up Original Model

Original Model

- Build SBOM for Model
  - depends on Runtime Libraries
    - generated from

- Source SBOM for Model
  - Source Files
    - Model Source Package
      - contains
      - depends on...
      - Static libraries

Building the AI Application

**Source SBOM for AI Application**

- AI Application Source Package
  - contains
  - Libraries
  - Source Files

**Build SBOM for AI Application**

- AI Application Executable
  - Generated from
  - Trained Model
  - depends on

Executable depends on

- Source SBOM for AI Application
Deploying the AI Application

Deployed Application SBOM

- AI Application Executable
- Runtime dependency of
  - Runtime Data

Generated from

Depends on

Trained Model

Source SBOM for Model

- AI Application Source Package
  - Contains
    - Source Files
    - Libraries

But what about the trained model?

SPDX can represent most of this today.
GAP: Representing Training the model

Build SBOM for Trained Model

Trained Model

Original Model

Validation Data

Training Data
AI BOM Transparency Survey: Fields Required?

Datasheets

Model Cards

FactSheets

---

### Datasheets

Trinity Goets, Justin Moritzstein, Brian Sweeney, Jennifer Yeomans Young, Hanna Wallach

**Abstract**

The machine learning community has been using diverse methods to document AI systems. While this has created a wealth of information, it often lacks a cohesive way to understand the full lifecycle of an AI system. This paper motivates and conceptualizes a new approach to the problem: AI BOM transparency. We argue that, similar to how other fields have developed datasheets and model cards to encourage transparency, AI should do the same. We then propose the concept of datasheets for datasets, model cards for model reporting, and fact sheets for factoring.

- **Keywords:** Datasheets, Model Cards, FactSheets

### Model Cards

**Model Cards for Model Reporting**

Margaret Mitchell, Simona Wu, Andrew Zaldivar, Parker Burns, Lora Wasserman, Ben Hutchinson, Bina Singh, Ines Dehghani, Raj, Trinity Goets

**Abstract**

Model cards are a transparent, subjective approach to sharing a model’s architecture, assumptions, and intended use. Model cards are a way to provide technical input to help users understand the assumptions and limitations of a model.

- **Keywords:** Datasheets, Model Cards, FactSheets

### FactSheets

**Increasing Trust in AI Services through Supplier’s Declarations of Conformity**


**Abstract**

This paper describes a model card for a particular service, the Datasheets, Model Cards, and FactSheets. Datasheets, Model Cards, and FactSheets are all different types of documentation that can be used to describe different types of machine learning models. Datasheets are concise, standardized, and can be applied to any type of machine learning model. Model cards are more detailed and can be applied to more complex models. FactSheets are the most detailed and can be applied to the most complex models.
AI Profile Properties

Required Fields:
- Creator
- Supplier
- PackageVersionInfo
- DownloadLocation
- PackageDescription
- LicenseConcluded
- LicenseDeclared
- ReleaseTime

Optional Fields:
- Originator
- Checksum
- ValidUntilTime
- BuildTime
- PackageComments
- SensitivePersonalInformation
- EnergyConsumption
- StandardsCompliance
- InformationAboutTraining
- Hyperparameters
- SafetyRiskAssessment
- DataPreprocessingSteps
- ModelExplainabilityMechanisms
- MetricsDecisionThresholds
- Metrics
- Autonomy
- Domain
- Limitations
- Type
Dataset Profile Properties

AI/ML Application

Required Fields:
- Name
- Originator
- DownloadLocation
- LicenseConcluded
- LicenseDeclared
- PackageDescription
- BuiltTime
- ReleaseTime
- DatasetType

Optional Fields:
- Supplier
- VersionInfo
- Checksum
- ValidUntilTime
- IntendedUse
- DatasetCollectionProcess
- DatasetUpdateMechanism
- DatasetSize
- DatasetNoise
- KnownBias
- Errata
- SensorsUsed
- StandardCompliance
- SensitivePersonalInformation
- ConfidentialityLevel
- AnonymizationMethodUsed

Model

Data

#ossummit
Usage Profile: to tell intentions as “Usage” for Delivery Product

Package Supplier

Product Package X (rev.0.1)
- Package A (rev.1.0)
- Package B (rev.0.1)

... Package X (rev.0.8)
- Package A (rev.1.1)
- Package B (rev.0.2)

Product Package X (rev.1.0)
- Package A (rev1.2)
- Package B (rev1.0)

Product Maker etc.

Design
Prototype
Test
...

Product Maker must know
1. License Conditions
2. Intended usage
3. Build conditions
4. Test conditions
5. Valid Until Date
6. Valid Until Event

Both parties must control the **Terms and Conditions** of using of the Delivery

Product Package X (rev.N) is only available for the specified period.
What’s next after After 3.0?
Future Direction: Hardware

- Safety Standards expect to know “system” that software is running on
- Vulnerabilities come from interaction between hardware and software (ie. heartbleed)
- Potential participants:
  - RISC-V & ARM core adopters,
  - Chips Alliance Members
  - Board Manufacturers
- For more information, contact: Kate Stewart
Future Direction: Safety Standards Automation

- Safety Standards expect to know
  - The source code at the time of production release
  - The documentation associated with the code
  - The configuration used to build the production software
  - The specific versions of the tools used to build the software

- Safety Standards Configuration Management (CM) Requirements are greatly simplified by following an effective SBOM process.
  - 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:
Leverage SPDX Relationships to Support Safety Analysis

Using SPDX Relationship Information

Assumption: process to create and maintain all artefacts (requirements, architecture, tests, analysis report) is accepted and applied

Still the biggest pain: Keeping a complete and consistent set of documentation and verifying that the evidences are complete and consistent

SPDX style solution: Create SPDX Relationships between all documentation artefacts to track all possible system combinations!

SPDX for product documentation

What kind of product documentation do we need to manage?

Specifications, Reports, Tests...

- Safety Requirement Specification: a SPECIFICATION for functional requirements, architectural elements etc.
- Unit Test: the TEST_CASE related to code or a specification artefact
- Unit Test Report: DOCUMENTATION of a unit test EVIDENCE of how the test was performed as planned
- Code: usually a GENERATED from or according to some specification artefact
- Coding Guidelines: SPECIFICATION about the project specific details for the code

Source: https://fosdem.org/2023/schedule/event/sbom_fusa/
Leveraging SPDX Relationships

- **Safety Concept**
  - SPECIFICATION_FOR

- **Plans Package**
  - SPECIFICATION_FOR
  - REQUIREMENTS_FOR

- **Implementation Guidelines Package**
  - REQUIREMENTS_FOR
  - Source Package (Code, Scripts, Docs)
  - GENERATES Executable
  - GENERATES Evidence, reports
  - GENERATES Logs

- **Specification Package (Requirements)**
  - REQUIREMENTS_FOR
  - TEST_FOR
  - GENERATES Test Package (Test Spec, Scripts)

- **Test Framework**
  - INPUT_OF Test Package
  - GENERATES Evidence, reports

- #ossummit
Requirement to Code to Tests to Evidence Traceability

- Requirement A.1
  - foo.c
  - make
  - A.1.1 test
  - A.1.2 test
  - A.1.3 test

- Log from A.1.1 test
- Log from A.1.2 test
- Log from A.1.3 test

- Specification file, requirements, architecture
- Source file
- Tests, test scripts
- Evidence, reports
Requirement to Code to Tests to Evidence Traceability

- **Requirement A.1**: foo.c
- **make**
  - A.1.1 test
  - A.1.2 test
  - A.1.3 test
- **Specification file, requirements, architecture**
- **source file**
- **Tests, test scripts**
- **Evidence, reports**
- **Bug Fix**
Requirement to Code to Tests to Evidence Traceability

Requirement A.1

New Requirement From Impact Analysis

NR test

A.1.3 test

A.1.2 test

A.1.1 test

foo.c

make

Test framework

Test framework

Test framework

Test framework

Log from A.1.1 test

Log from A.1.2 test

Log from A.1.3 test

Log from NR test

## Specification file, requirements, architecture

<> source file

?? Tests, test scripts

!! Evidence, reports

Bug Fix
Safety Profile Introduction (3.1 target)

- Also known as Functional Safety (or FuSa).
- Purpose is to link together all the safety artifacts (including code and relevant tests) with the aim of being able to automatically detect what a file update may need to force retesting.
- Goal is to support continuous certification of safety artifacts after security updates are applied.
- Overview can be found at: https://fosdem.org/2023/schedule/event/sbom_fusa/

For more information, contact: Nicole Pappler or Kate Stewart
Want to Help?

- If you have a **use-case** you want to make sure can be supported in the future SPDX specification,
  - join the SPDX tech team mailing list ([https://lists.spdx.org/g/Spdx-tech](https://lists.spdx.org/g/Spdx-tech)),
  - open an issue in [https://github.com/spdx/spdx-spec](https://github.com/spdx/spdx-spec) and
  - join in on the discussion!

- Try it, and let us know if you see issues.
Thank you!  Questions?
How to Get Involved - PRs & Issues

https://github.com/spdx

- Specification
  - https://github.com/spdx/spdx-spec ← ISO submission format
  - https://github.com/spdx/spdx-3-model ← 3.0 development
  - https://github.com/spdx/spdx-examples

- Tooling
  - https://github.com/spdx/tools-python
  - https://github.com/spdx/tools-golang
  - https://github.com/spdx/tools-java

- License List
  - https://github.com/spdx/license-list-XML
Embedded Projects Generating SBOMs

Zephyr’s west spdx

Presentation / Demo:
https://www.youtube.com/watch?v=KYC3YpSu9zs

Yocto builds

Presentation / Demo:
https://www.youtube.com/watch?v=y0N4FnkwTOY
Relationship between SBOMs

Learn more at: https://www.youtube.com/watch?v=KYC3YpSu9zs
SBOMs Included By Default ... Automatically