The SPDX Safety Profile

Nicole Pappler

CTO,

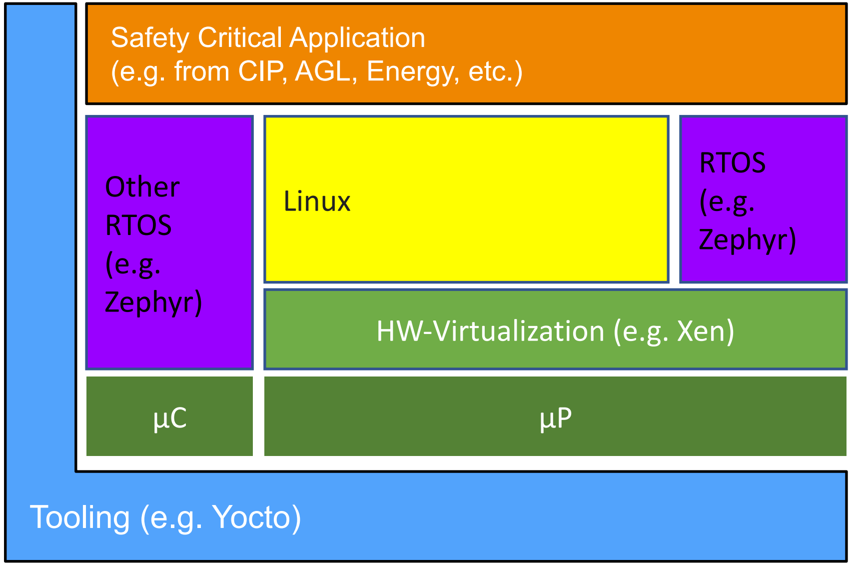
AlektoMetis.com

Stanislav Pankevich

Software Architect,

Reflex Aerospace GmbH

Safety Analysis is Performed on Systems



Definition of Functional Safety

- **Safety** – the freedom from unacceptable risk of physical injury or of damage to the health of people, either directly, or indirectly as a result of damage to property or to the environment

- **Functional Safety**
  - the part of safety that depends on a system or equipment operating correctly in response to its inputs
  - Detecting potentially dangerous conditions, resulting either in the activation of a protective or corrective device or mechanisms to prevent hazardous events or in providing mitigation measures to reduce the consequences of the hazardous event.

Functional Safety - systematic capability

SPDX SAFETY

Safety is a system property!

But:

Systematic capability is the general assumption, that

- if development, test and deployment of a system follow a specific set of tasks and
- there is evidence for adherence to these tasks
- (and under the assumption that the system architecture supports safety)

⇒ **Software is capable of performing as intended**

SPDX

Functional Safety - Standards

SPDX SAFETY

What are these tasks and evidences?

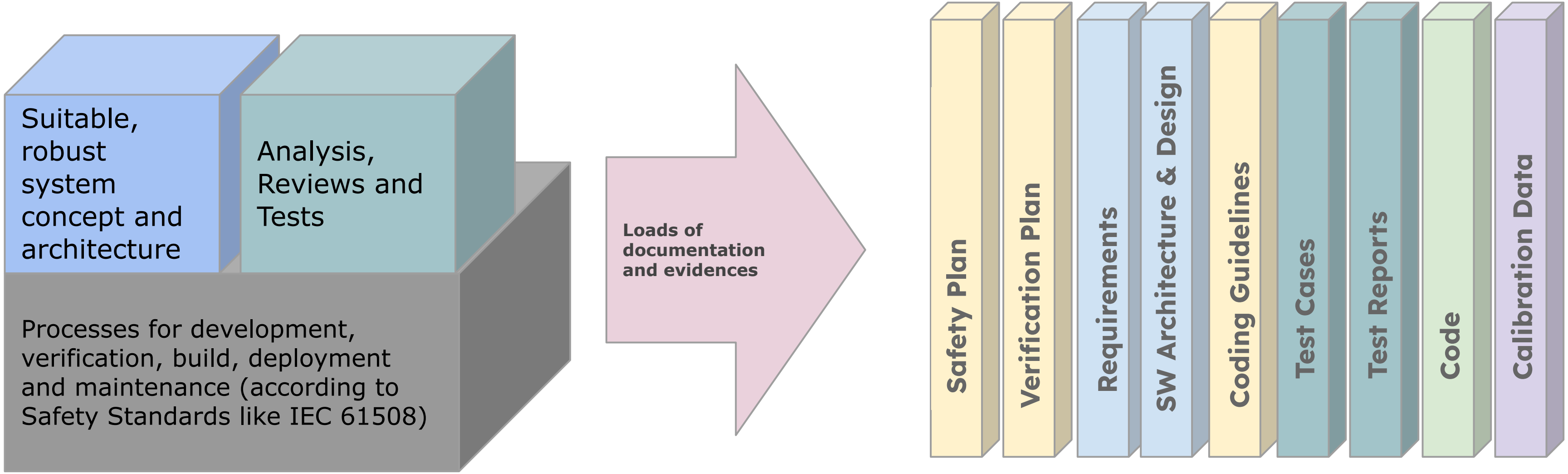
- Usually defined in Safety Standards
- Focus: Unique IDs, traceability, completeness, evidences

⇒ define your dependencies (also inside of your project!) **and keep them up to date**!

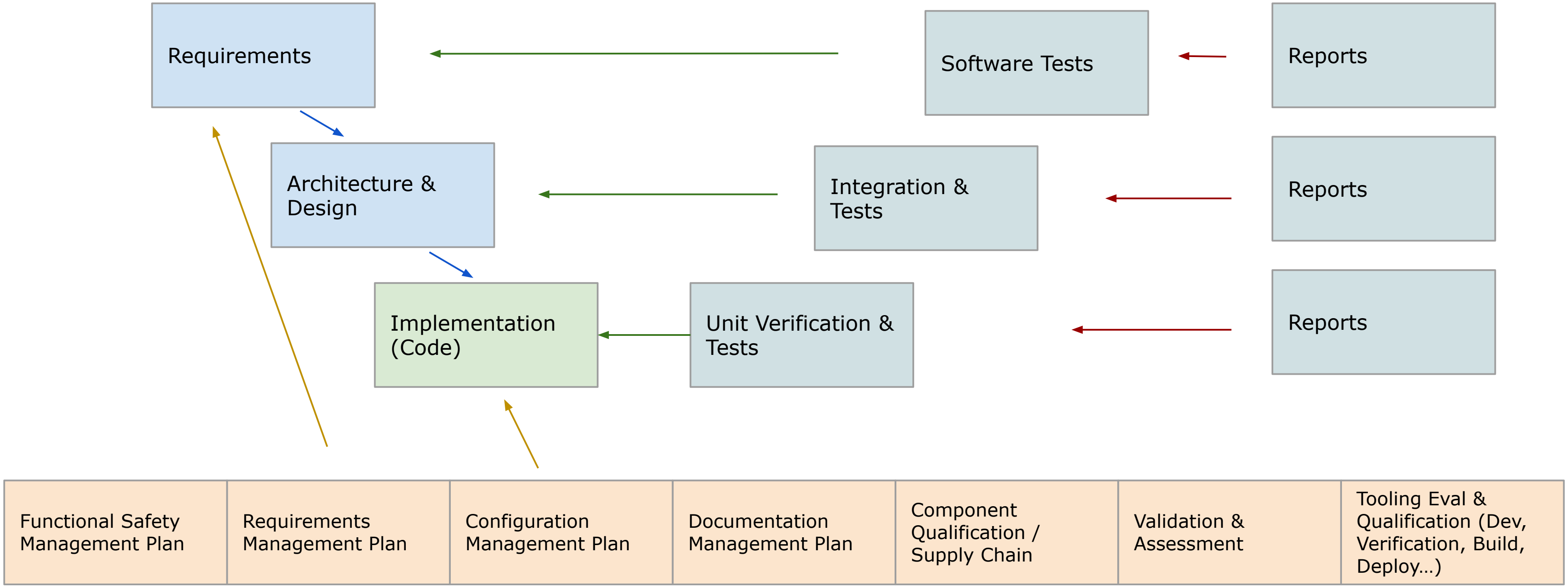
SPDX

What is FuSa aiming for?

Safety Architecture and Documentation



Dependencies in a FuSa Project



SPDX

Maintenance

After Applying a Vulnerability Fix

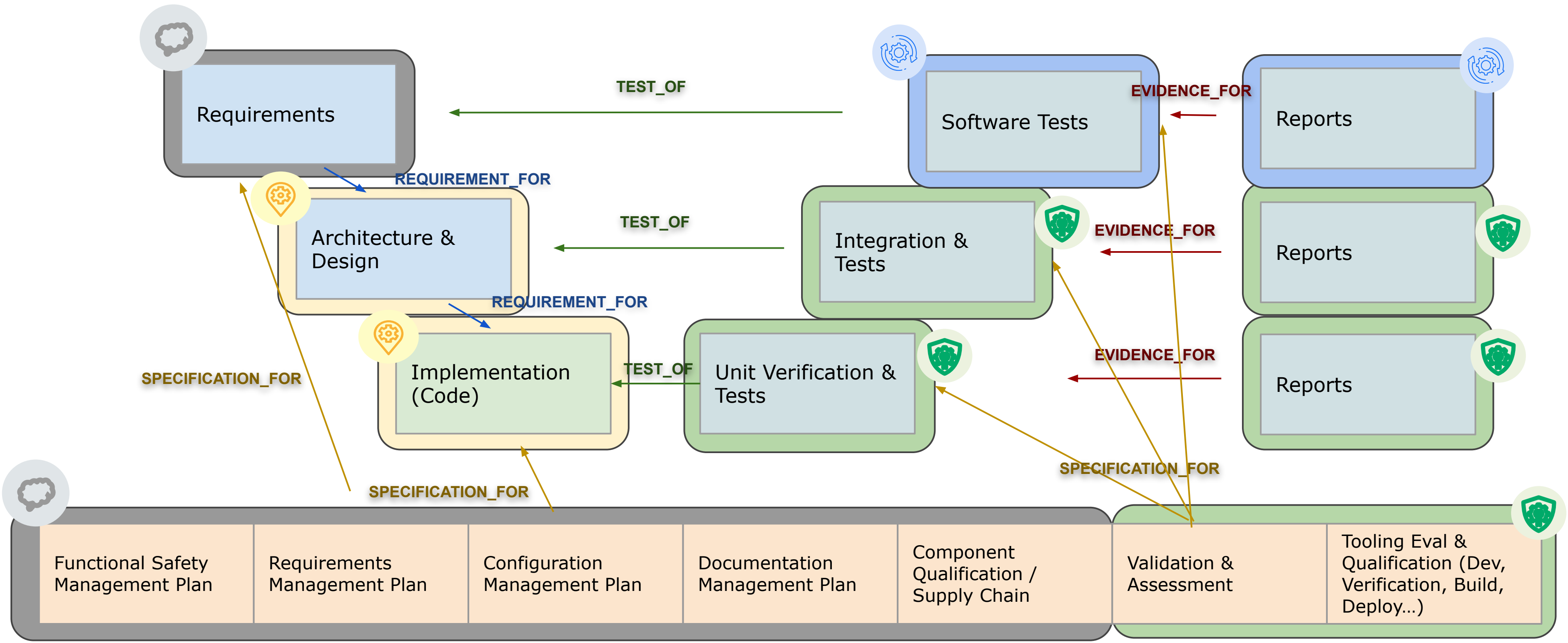
Requirements are needed to know you're "**done**" after applying a patch:

- 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 enables recording and connecting the sources, assessments, vulnerabilities & patches, build & calibration data, tests, requirements and evidence ⇒ **path to automation**

SPDX Safety Dependencies in a FuSa Project



Maintenance and Promotion of Safety Principles

SPDX SAFETY

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)

SPDX

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

Maintenance and Promotion of Safety Principles

SPDX SAFETY

Available in SBOM

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)

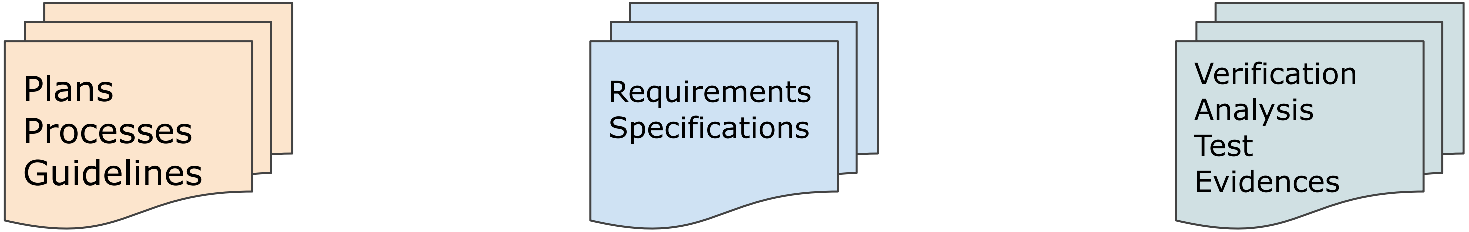
SPDX

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

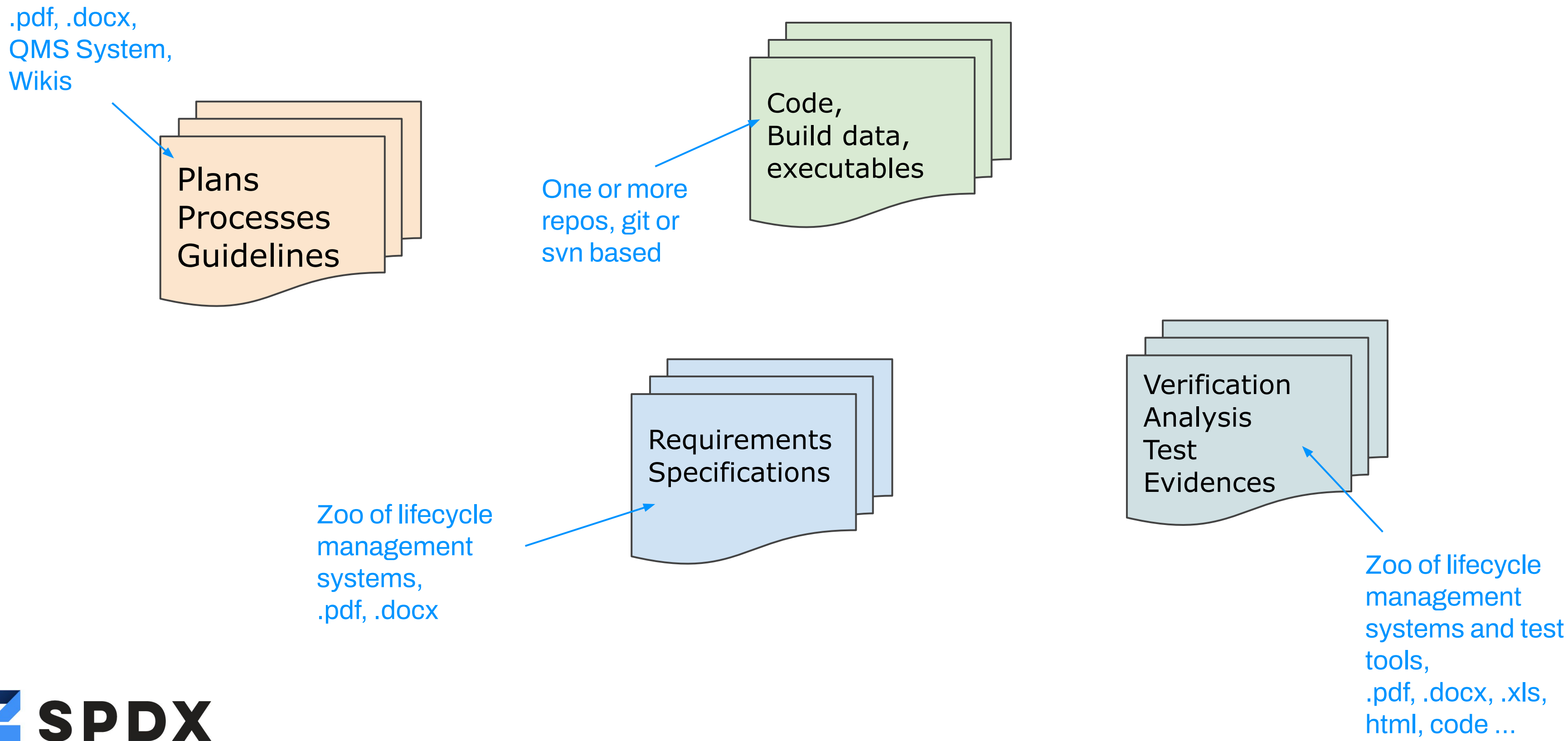
FuSa documentation structure

All FuSa related documentation is part of the Safety Case!

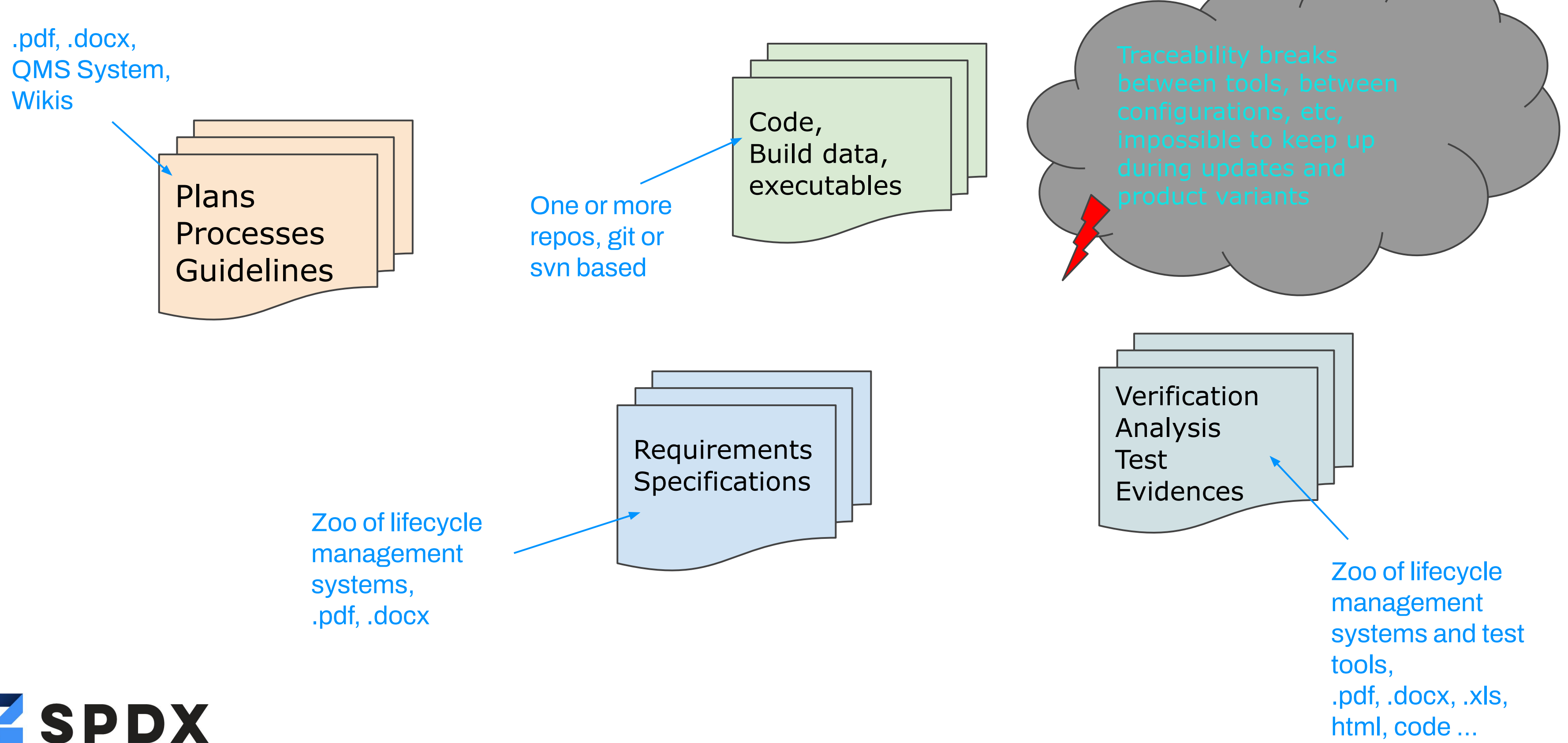
Think of all these documents as part of the release - each document is part of the Bill of Material, as is each screw, each microcontroller and each piece of software!



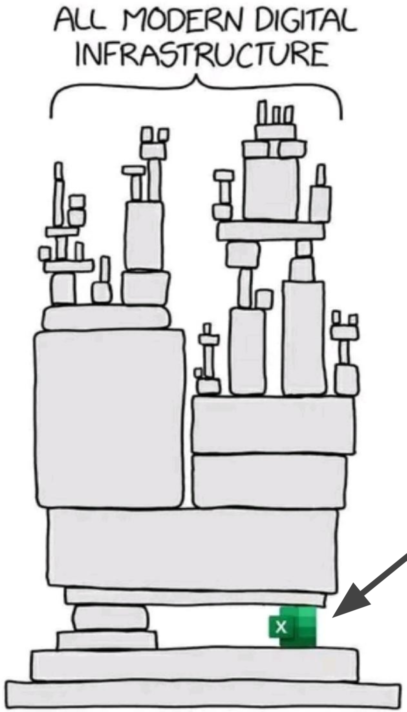
Data Structure of current FuSa projects...



Data Structure of current FuSa projects...

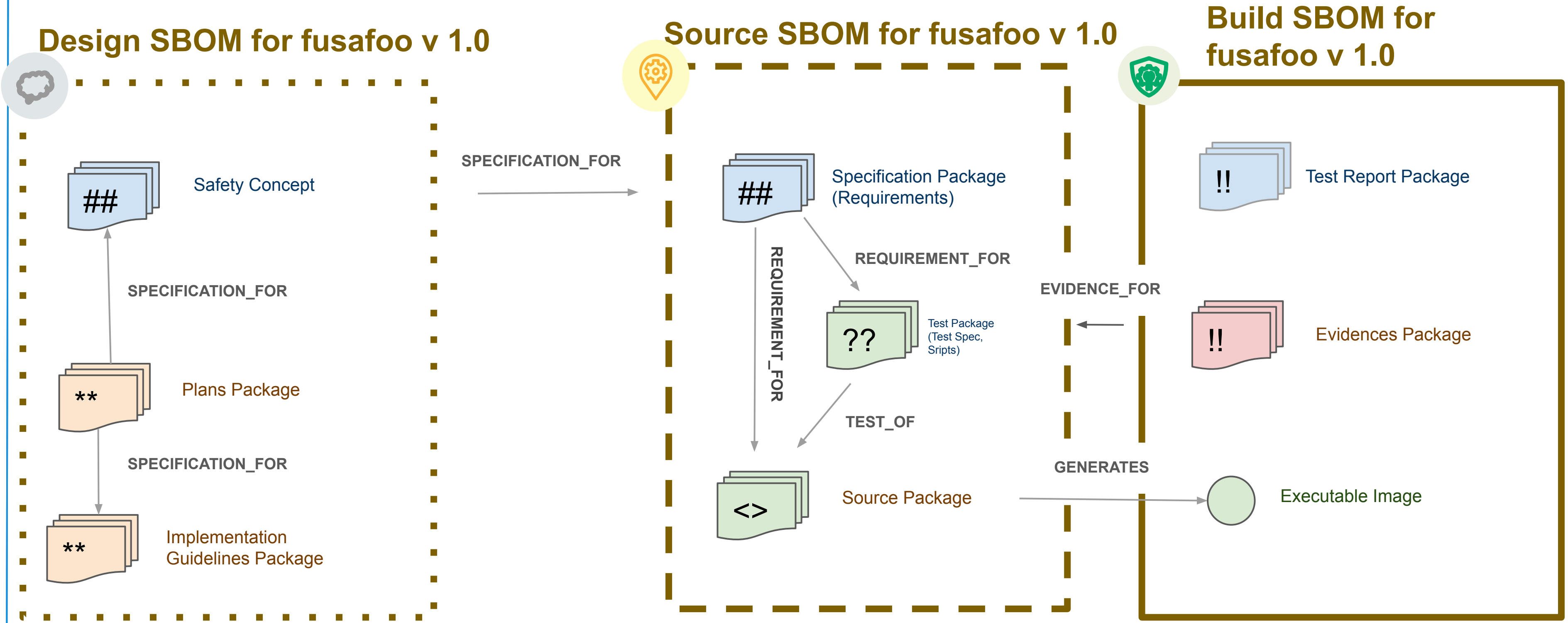


No 1 Safety Information Exchange Format



draft_2005TemplateSafetyCase_thisproject_final_forTraceingv06.xls

Generic Project View



Using the SPDX Safety Profile for the Zephyr Project

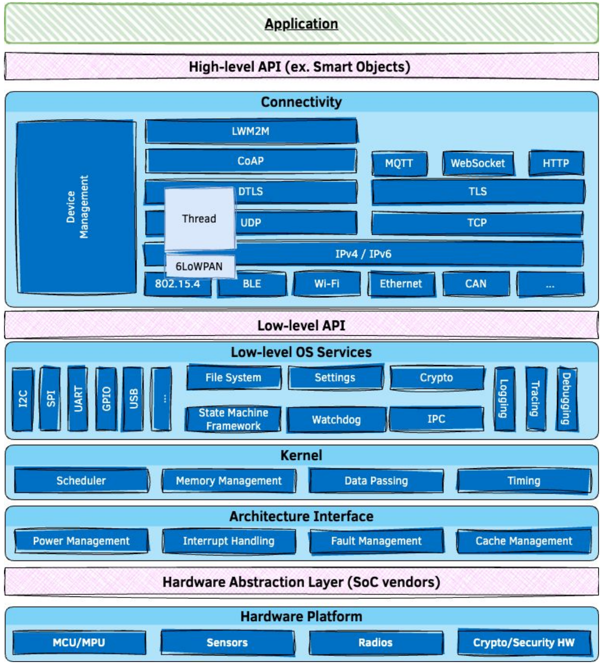
Software Architectural Element

Zephyr Project:

- Embedded RTOS
- Build system
- Test cases & Test framework

Plus evidences for (safety) systematic capability:

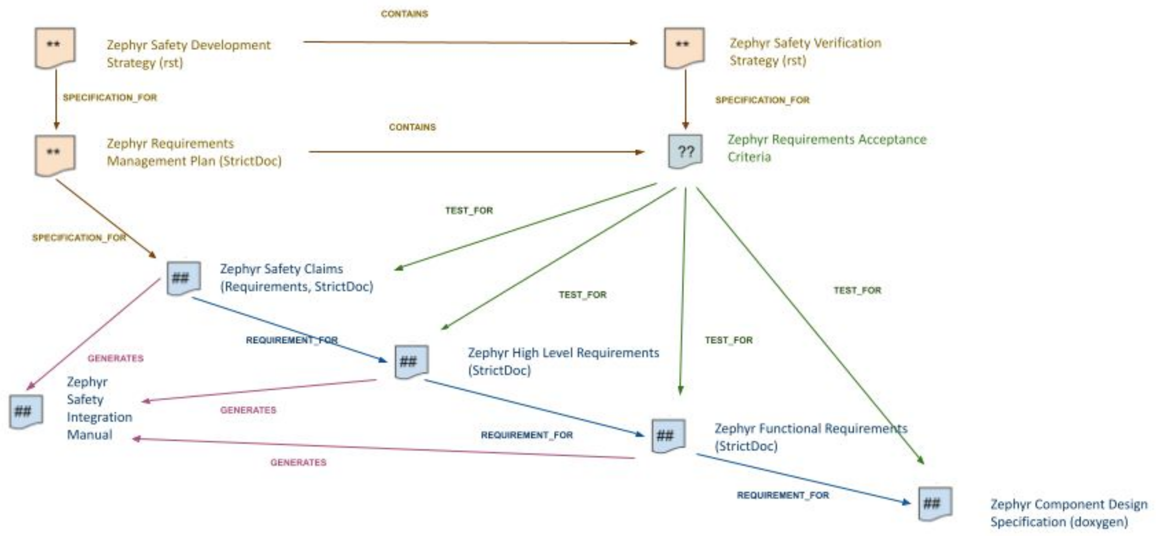
- (Safety) Requirements
- Functional Safety Management plans
- Safety Analysis
- Completeness, Compliance & (Test & Analysis) Coverage Evidences



Zephyr Requirements Management

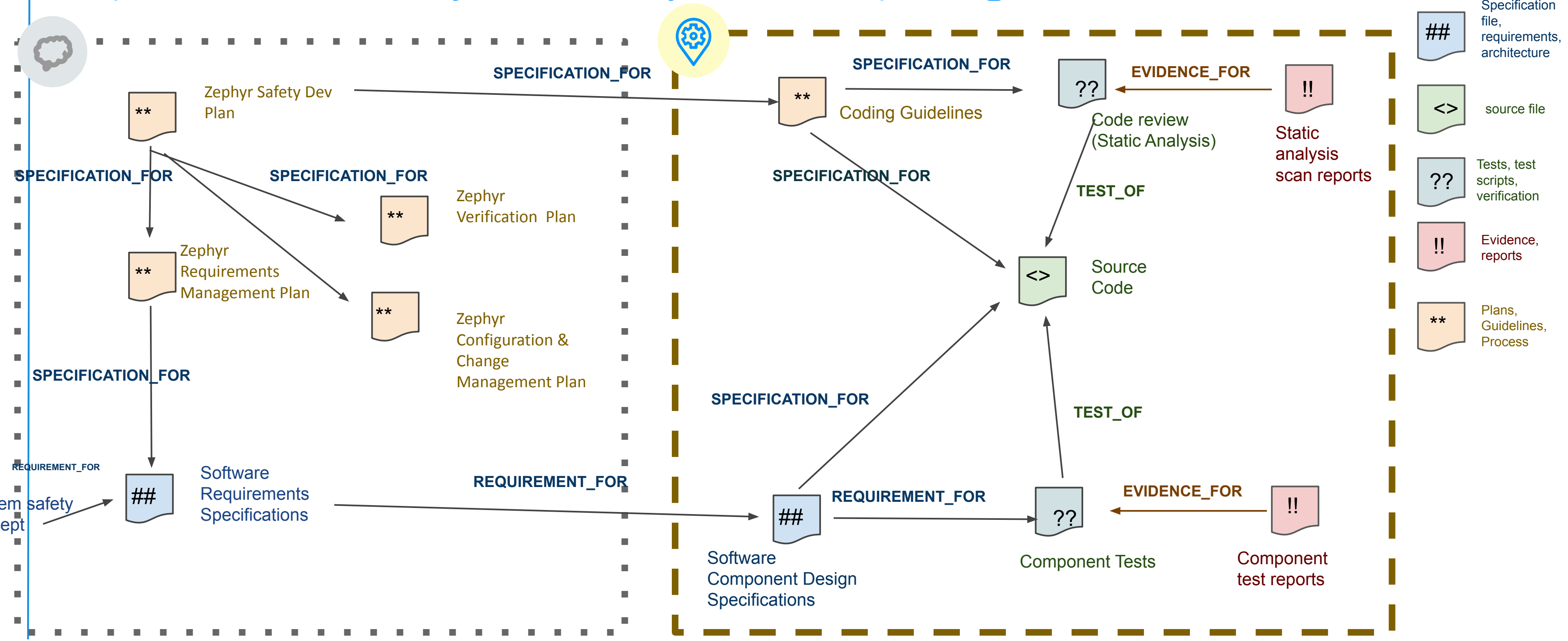
Requirements Management Knowledge Model

Safety Working Group View & Verification



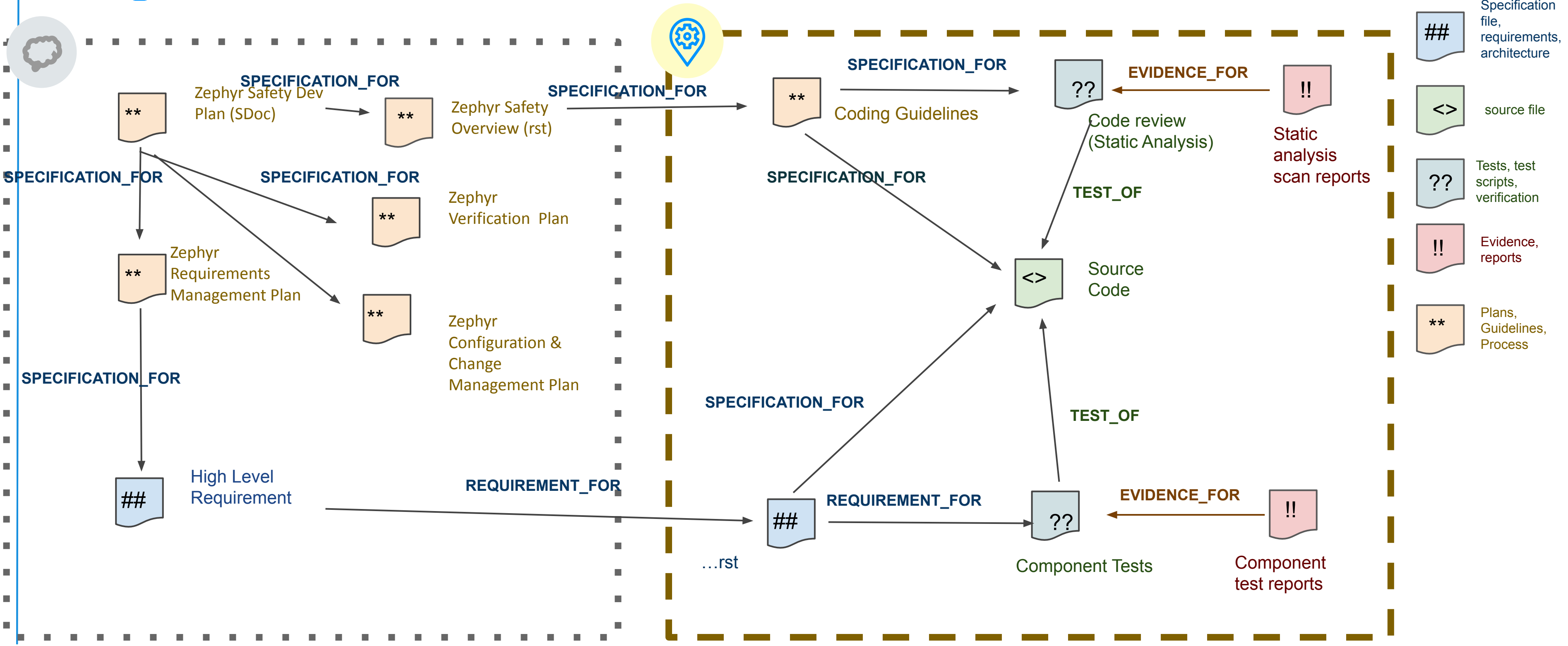
Zephyr Safety:

Dependencies of Safety Plan, Safety Claim, Req, Design and Code



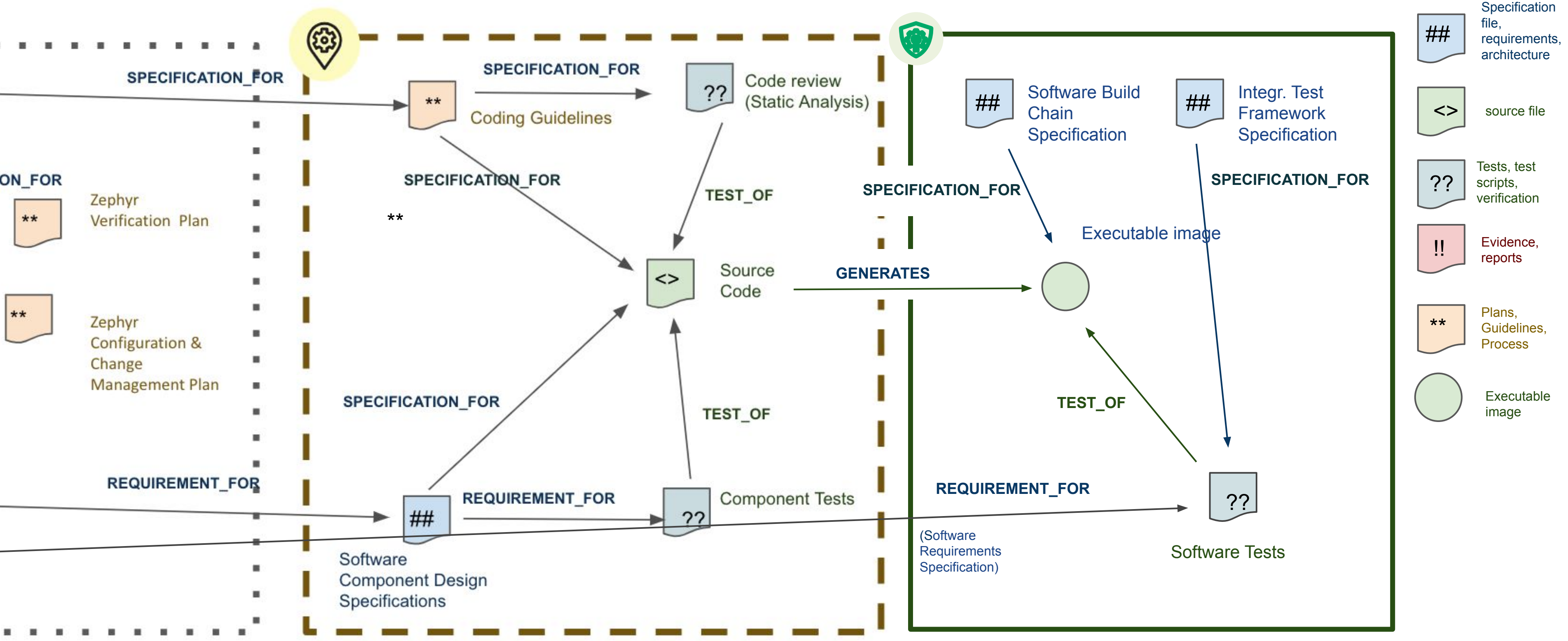
Zephyr Safety:

Design SBOM to Source SBOM

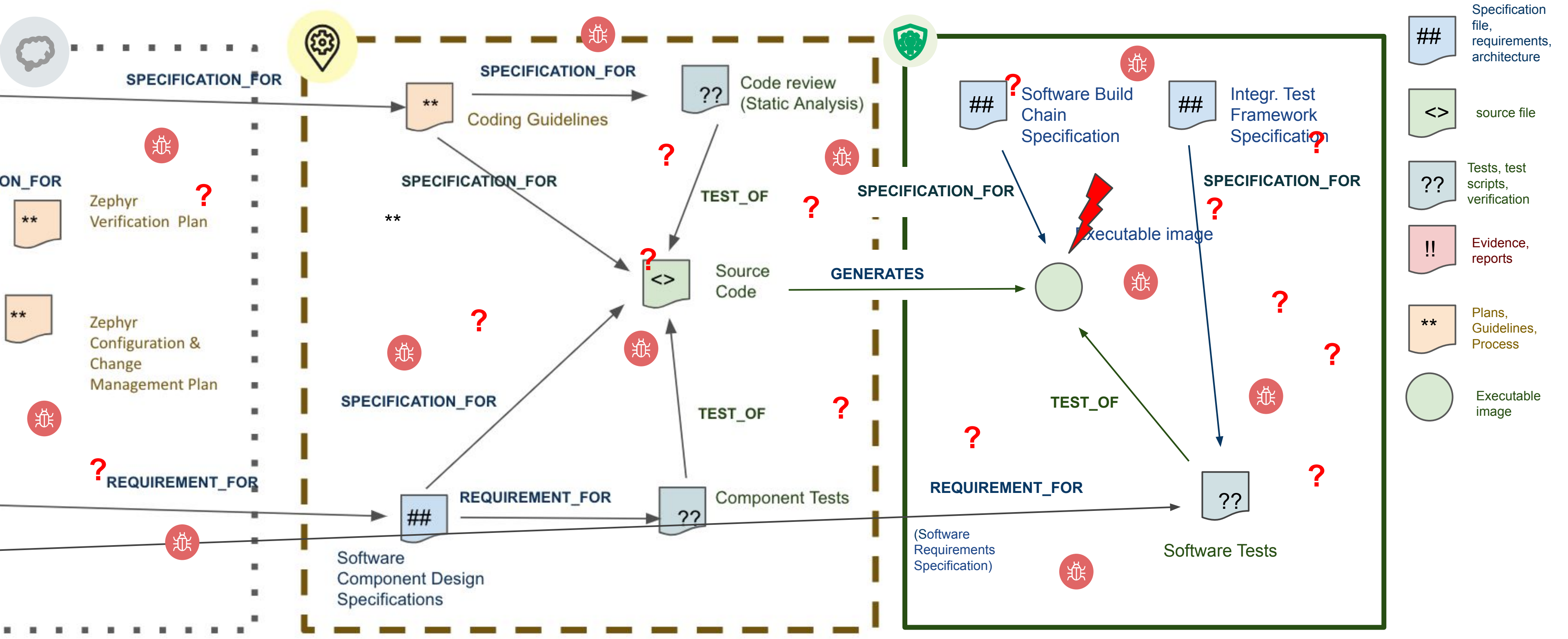


Zephyr Safety

Source SBOM to Build SBOM

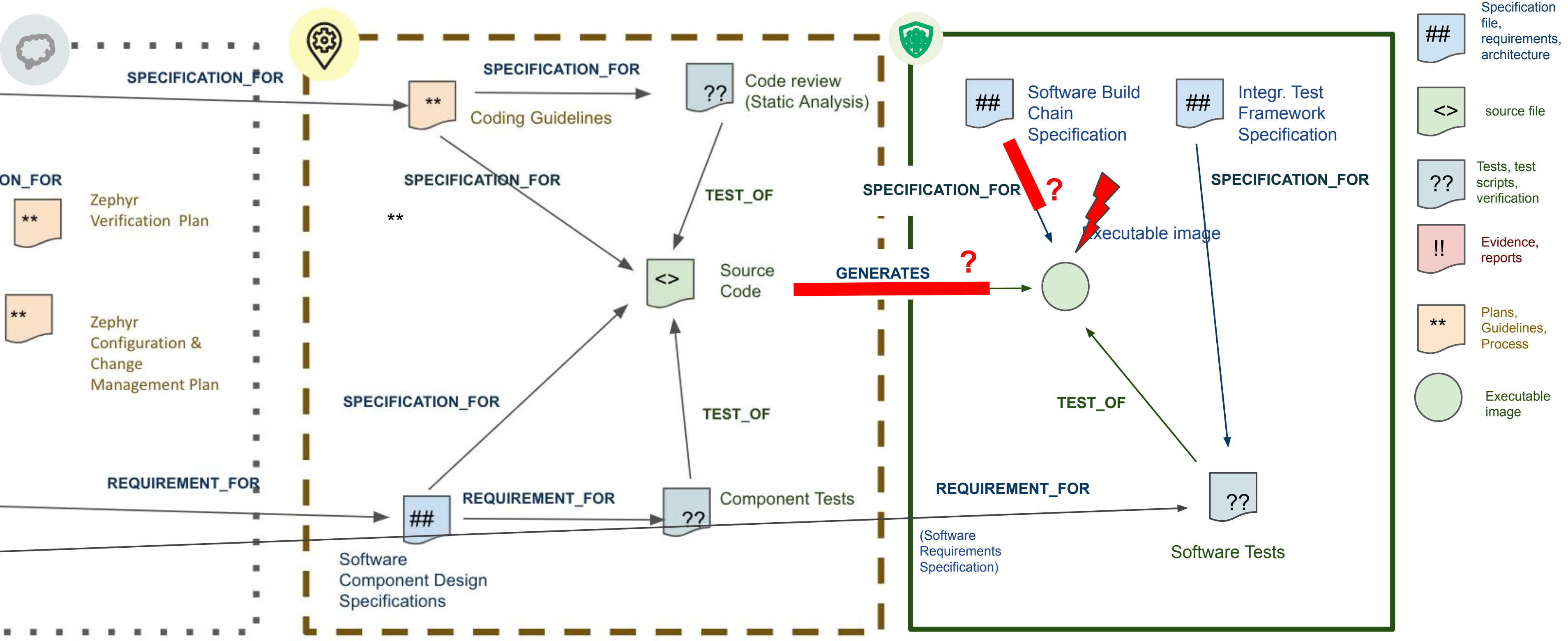


Dependency Identification on Component Level

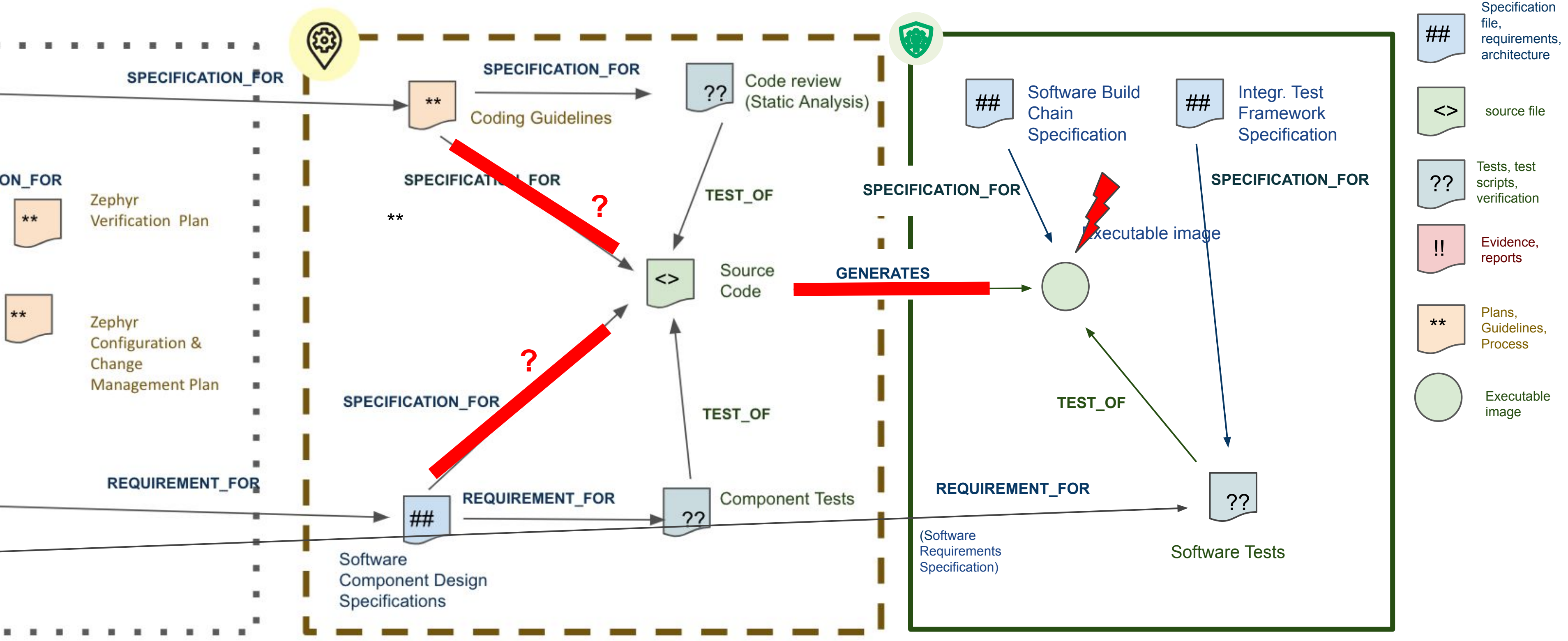


SPDX

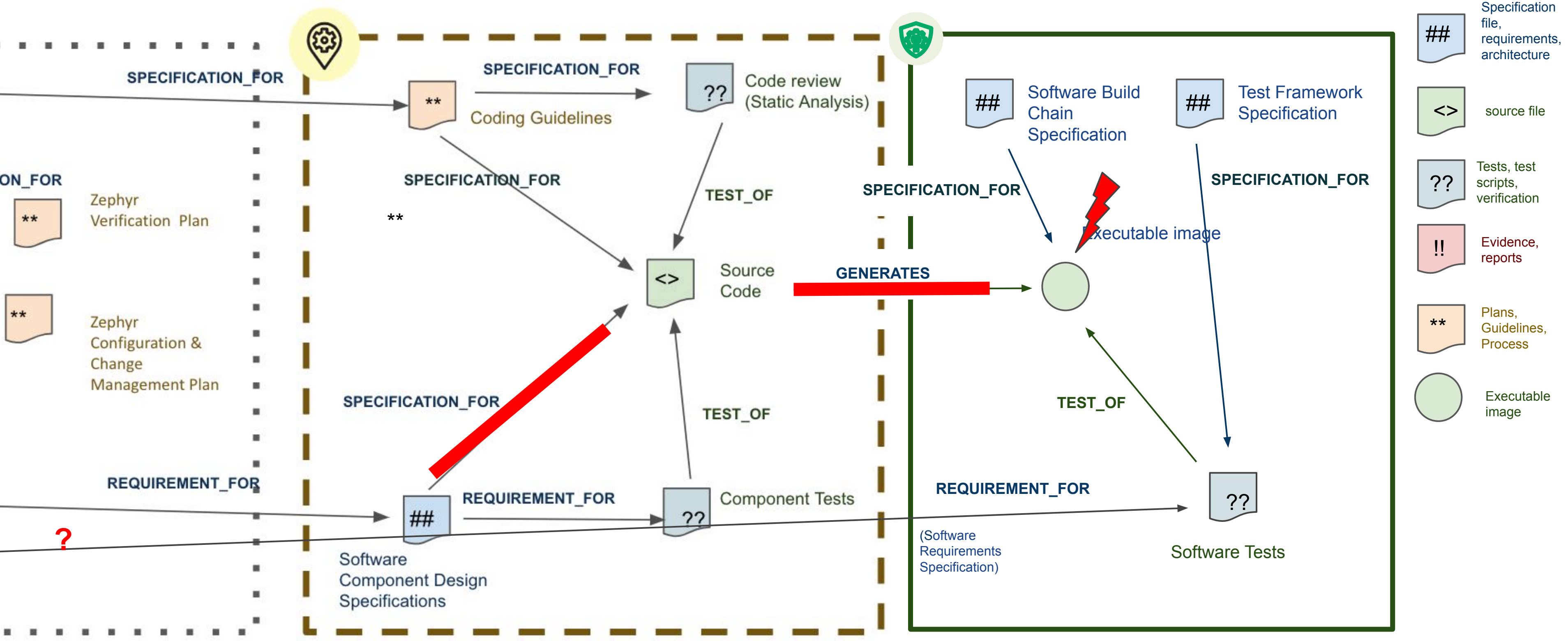
Dependency Identification on Component Level



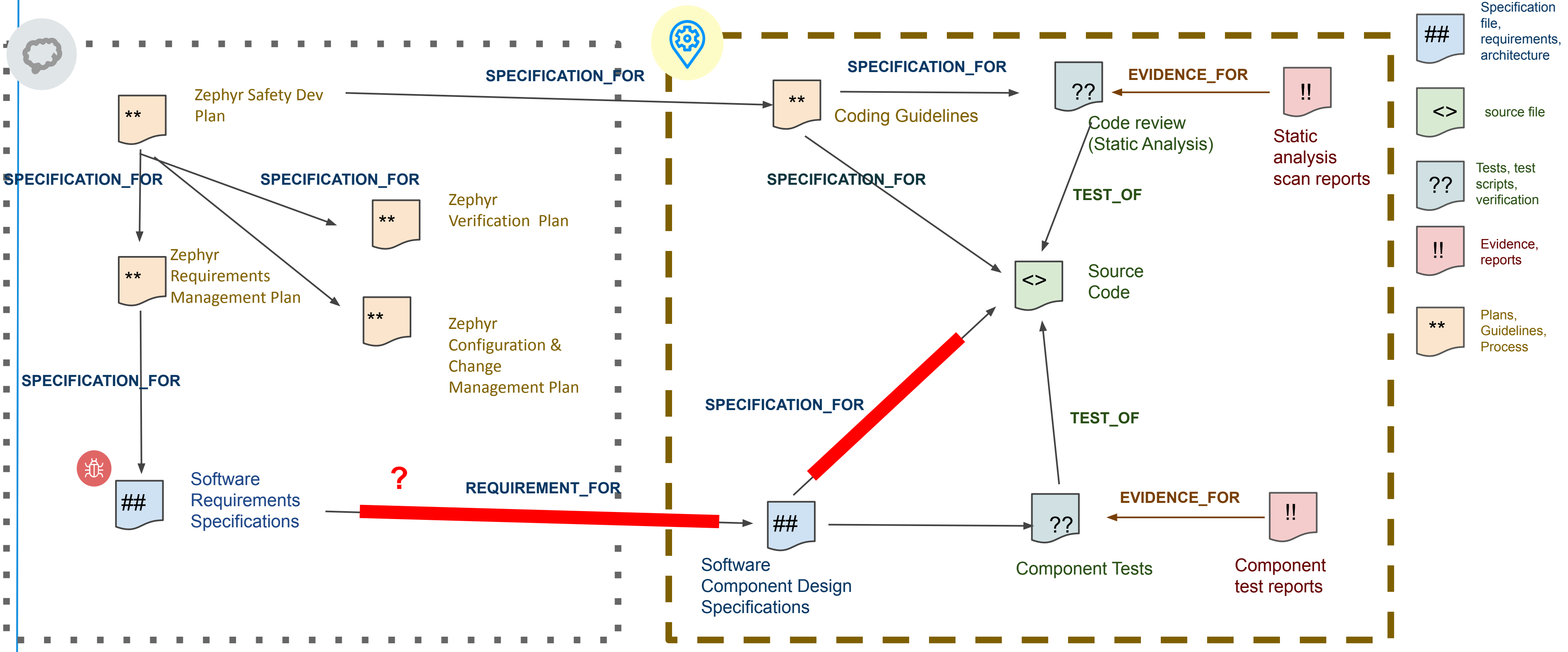
Dependency Identification on Component Level



Dependency Identification on Component Level



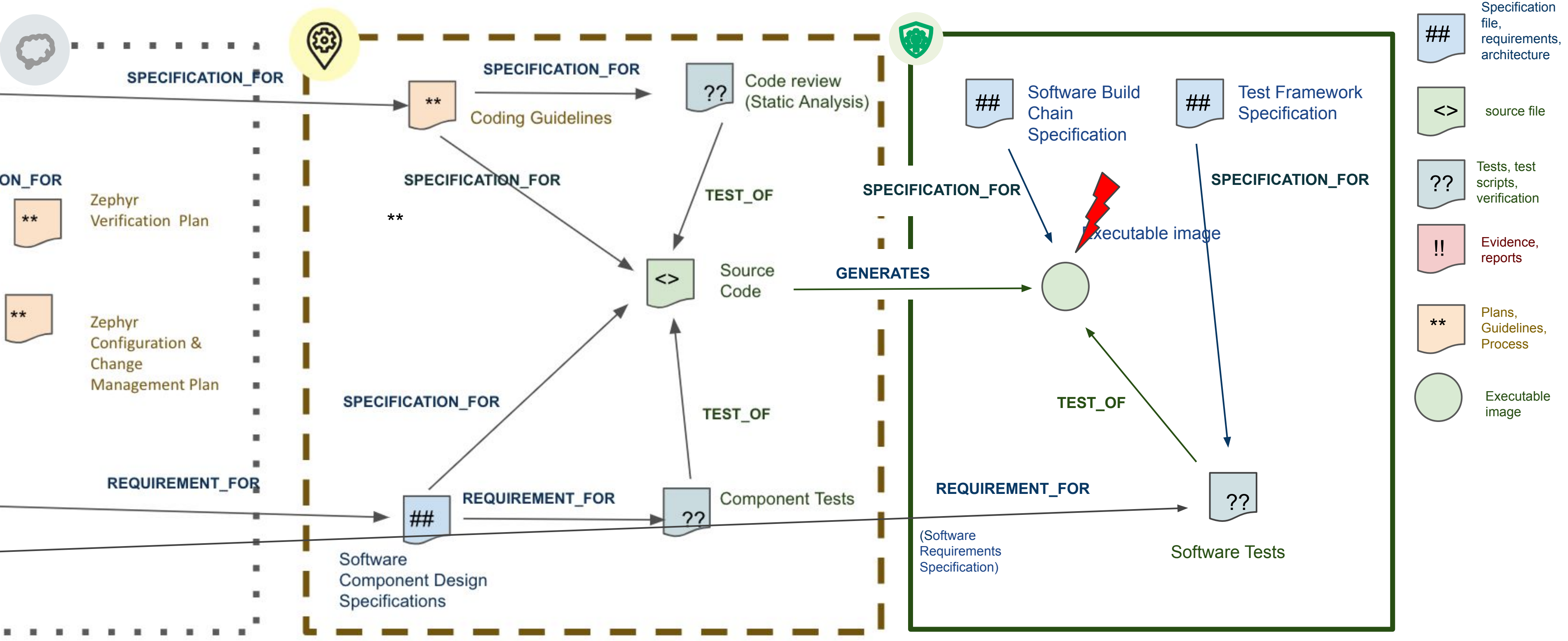
Dependency Identification on Component Level



SPDX SAFETY

SPDX

Dependency Identification on Component Level



Issues in Requirements Engineering

- Commercial requirements tools can be expensive
  - How to build a working group with several organizations collaborating?
- Exchanging requirements
  - What if organizations use different tools and formats?
- Requirements and software worlds are often not connected
  - An initial Word/Excel document gets forgotten in the implementation
- Requirements and open source software are mostly not connected
  - Waterfall model struggles with OSS's rapid and decentralized development
  - Very few OSS projects are developed according to requirements
- But everything is changing (slowly)!
  - GitHub: Over 12 OSS requirements tools with various degrees of maturity
- <u>Key question: How to make requirements useful for open source software?</u>

StrictDoc – FOSS requirements tool

- Created in 2019
- Spare-time project for two core developers
- 1.6K pull requests, 3.4K commits, 30K+ LOC, Apache 2 license
- Inspired by Doorstop's OSS approach to requirements management
- 2020-2022:
  - Documentation generator, HTML export, ReqIF, tracing source files to requirements, custom fields, traceability graph validations
- 2023:
  - A year of the web-based user interface. The HTML-to-PDF feature for publishing documents.

StrictDoc – Project goals

- Long-term vision: a free and open-source, but highly capable, tool that makes requirements work easy and enjoyable
- Automate requirements work at all levels
- All target groups are considered:
  - Software, hardware
  - Systems, electrical, thermal, etc.
  - QA, Safety, management, non-technical, etc.
- Usable on both individual laptops (pip install) and eventually on cloud
- Start creating requirements in 5 minutes, scale to large documents
- Open data: easy way to get data in and out
- Synergies with other tools, e.g., everything Python, Capella MBSE, SPDX, etc.

.SDoc format

- Starting point: Format to support both text and metadata
- YAML frontmatter does not scale
- RST directives do not support nested metadata
- JSON is less human-readable, and so are HTML/XML
- Nesting content in a document with 8+ chapter levels does not scale visually

- SDoc ('strict-doc') is a practical compromise inspired by:
    - YAML – nested meta information fields
    - TOML – keys in square brackets
    - XML/HTML – opening and closing tags for nested content
    - ASN.1 – Capital letters
- StrictDoc's implementation is not hard-coded to .SDoc

drafts > requirements > 01_strictdoc > L1_Open_Requirements_Tool.sdoc

```
[REQUIREMENT]
UID: SDOC-SSS-3
TITLE: Documents (CRUD)
STATEMENT: >>>
The Requirements Tool shall provide the CRUD operations for document management:

- Create document
- Read document
- Update document
- Delete document.
<<<
RATIONALE: >>>
The CRUD operations are essential operations of document management. They are at the core of a documentation management tool.
<<<

[REQUIREMENT]
UID: SDOC-SSS-51
TITLE: Documents with nested sections/chapters structure
STATEMENT: >>>
The Requirements Tool shall allow management of documents with nested sections/chapters structure.
<<<

```

SPDX SAFETY

SPDX

Zephyr, SPDX and StrictDoc

- FOSDEM 2023 - Using SPDX for functional safety
- Collaboration with the Zephyr Safety Working Group since 2023 Q2
- Zephyr's requirements are written using StrictDoc
- The group is working on understanding and structuring the requirements, relating them to the source code and other artifacts of Zephyr

- StrictDoc interfaces to Zephyr:
  - SDoc files and Zephyr design documentation
  - SDoc files and Zephyr source files (under discussion)
  - StrictDoc-produced SPDX file that connects to the parent Zephyr SPDX

Live demo

- StrictDoc
- Zephyr requirements

SPDX SAFETY

SPDX

How StrictDoc supports Safety

- Create and manage technical documentation with requirements
- Traceability matrix for all artifacts
- Tracing requirements to source files
- Project statistics report
- Search query engine
- Diff and changelog
- Publishing standalone HTML and PDF documents
- ReqIF support for requirements exchange
- SPDX interface (joined the SPDX FuSa working group)

And other features, see [StrictDoc's Roadmap (SVG)].

SPDX SAFETY

Backup: StrictDoc – Technical details

- Requirements are stored in text files
- Git-controlled storage of requirements and source code
- The SDoc language is constructed using textX grammar
- Text markup – RST (other formats planned)
- Arbitrary nodes are supported (Requirement, Test, Assumption, etc.)
- Extensible document grammars, custom fields and relations
- The static HTML export and the dynamic web UI use the same templates
- ReqIF library is a satellite project of StrictDoc
- The software stack is lightweight
- Make maximum use of Git but also explore graph databases

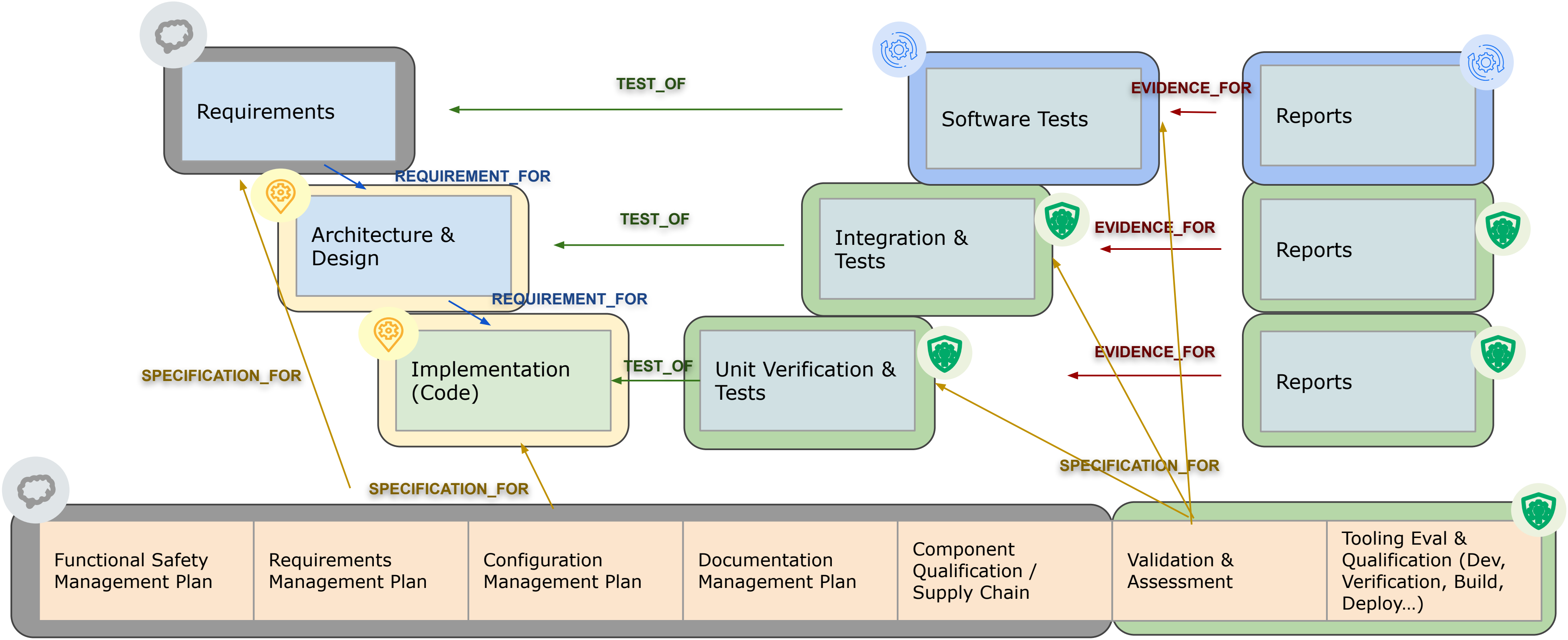
SPDX

Conclusions

Using a SPDX Safety Profile

- Provides a complete model of dependencies in a safety related project
- Standardized exchange format for a safety case
- Supports effective impact analysis methodologies (input information for FMEA, Ishikawa Analysis, GSN/SACM etc.)
- Provides reproducible results in both impact analysis and evidence generation
- Formal way to demonstrate completeness after project tailoring and for different scopes
- ...
- ...
- ...

SPDX Safety Dependencies in a FuSa Project



SPDX SAFETY

SPDX

Questions?

To join in evolving SPDX safety profile:

- Subscribe to: https://lists.spdx.org/g/spdx-fusa
- StrictDoc: https://github.com/strictdoc-project/strictdoc

Contact:

- Nicole Pappler - nicole@alektometis.com
- Stanislav Pankevich - s.pankevich@gmail.com

BACKUP SLIDES - MAYBE TO BE USED TO EXPLAIN THE DOCUMENTATION STRUCTURE

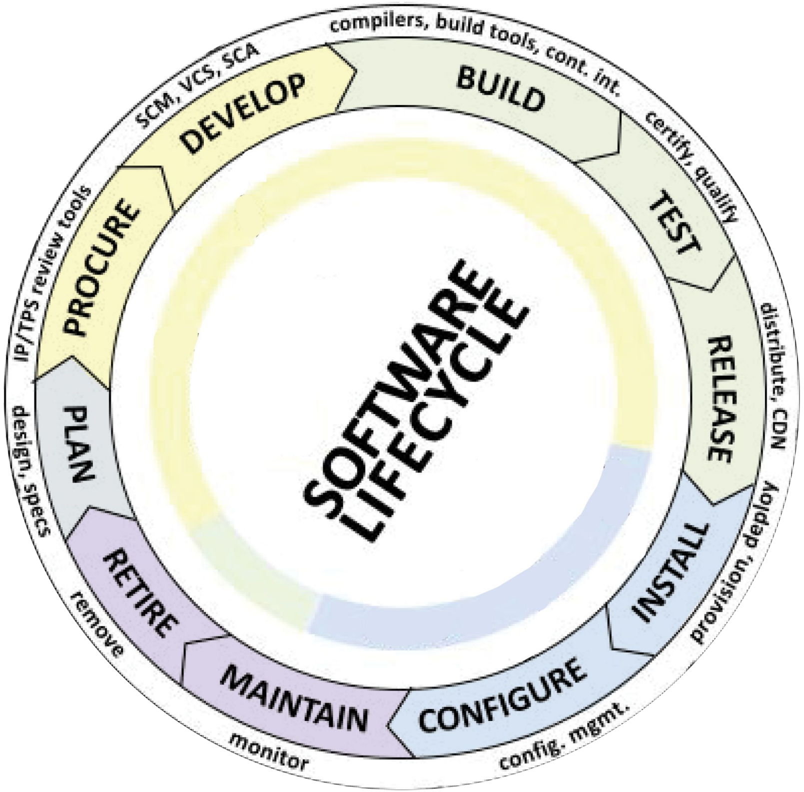
Generate SBOMS when the data is known

SPDX SAFETY

Source SBOM

Design SBOM

Runtime SBOM



Build SBOM

Deployed SBOM

SPDX

SBOM Types - manage your work products

SPDX SAFETY

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.

SPDX

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

Managing set of relevant items with SBOMs

SPDX SAFETY

	Design SBOM	Functional Safety Management (Plans) and Safety Concept
	Source SBOM	Requirements, Design, Safety Analysis, Source Code, Test Cases
	Build SBOM	Build Framework, Build configuration and environment data, Test Framework, Executable, Test Reports
	Deploy SBOM	Deployed configuration and environment data, Hardware architecture specific information and data, deployment tests and reports
	Runtime SBOM	Runtime relevant data (configuration data), training data, error logging data

SPDX

SPDX Relationships to Clarify Dependencies

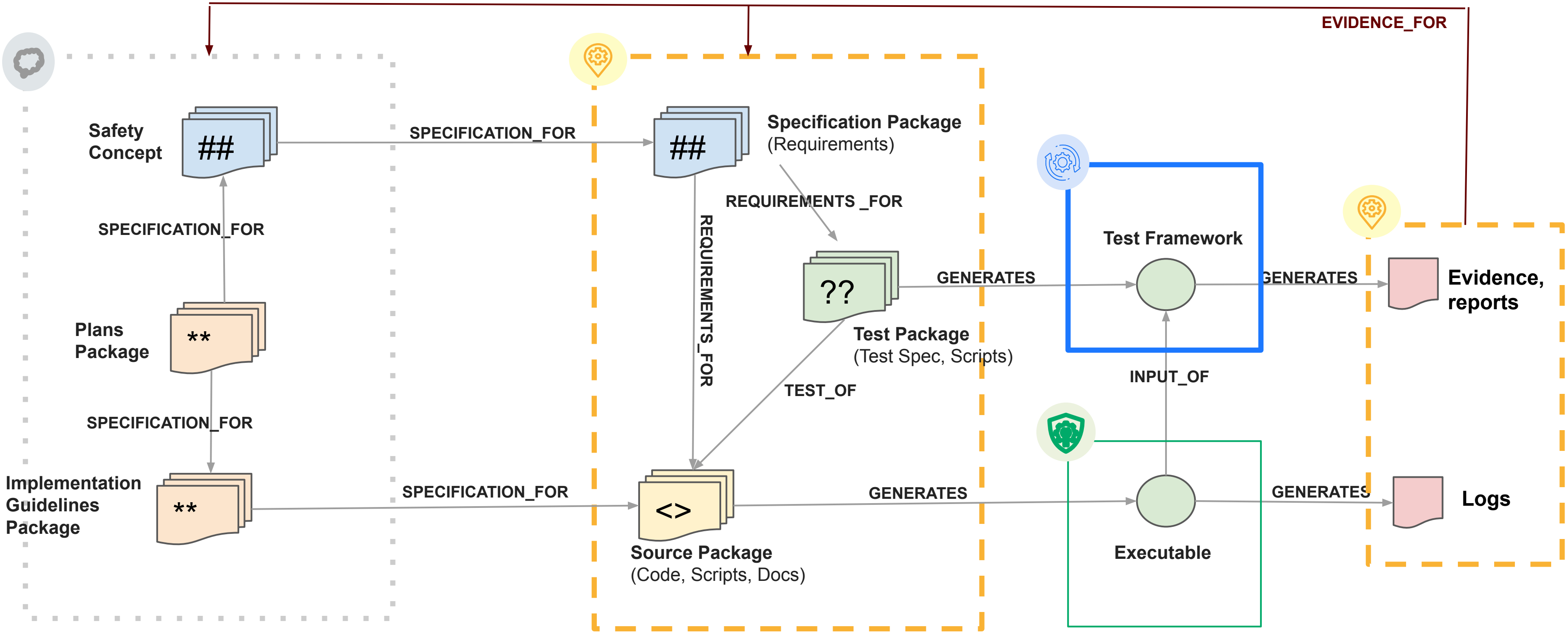
SPDX SAFETY

DESCRIBES	DEPENDENCY_OF	PREREQUISITE_FOR	GENERATES	VARIANT_OF
DESCRIBED_BY	RUNTIME_DEPENDENCY_OF	HAS_PREREQUISITE	TEST_OF	FILE_ADDED
CONTAINS	BUILD_DEPENDENCY_OF	ANCESTOR_OF	TEST_TOOL_OF	FILE_DELETED
CONTAINED_BY	DEV_DEPENDENCY_OF	DESCENDENT_OF	TEST_CASE_OF	FILE_MODIFIED
DYNAMIC_LINK	OPTIONAL_DEPENDENCY_OF	DOCUMENTATION_OF	EXAMPLE_OF	PATCH_FOR
STATIC_LINK	PROVIDED_DEPENDENCY_OF	BUILD_TOOL_OF	METAFILE_OF	PATCH_APPLIED
AMENDS	TEST_DEPENDENCY_OF	EXPANDED_FROM_ARCHIVE	PACKAGE_OF	REQUIREMENT_FOR
COPY_OF	OPTIONAL_COMPONENT_OF	DISTRIBUTION_ARTIFACT	DATA_FILE_OF	SPECIFICATION_FOR
DEPENDS_ON	DEPENDENCY_MANIFEST_OF	GENERATED_FROM	DEV_TOOL_OF	OTHER

SPDX

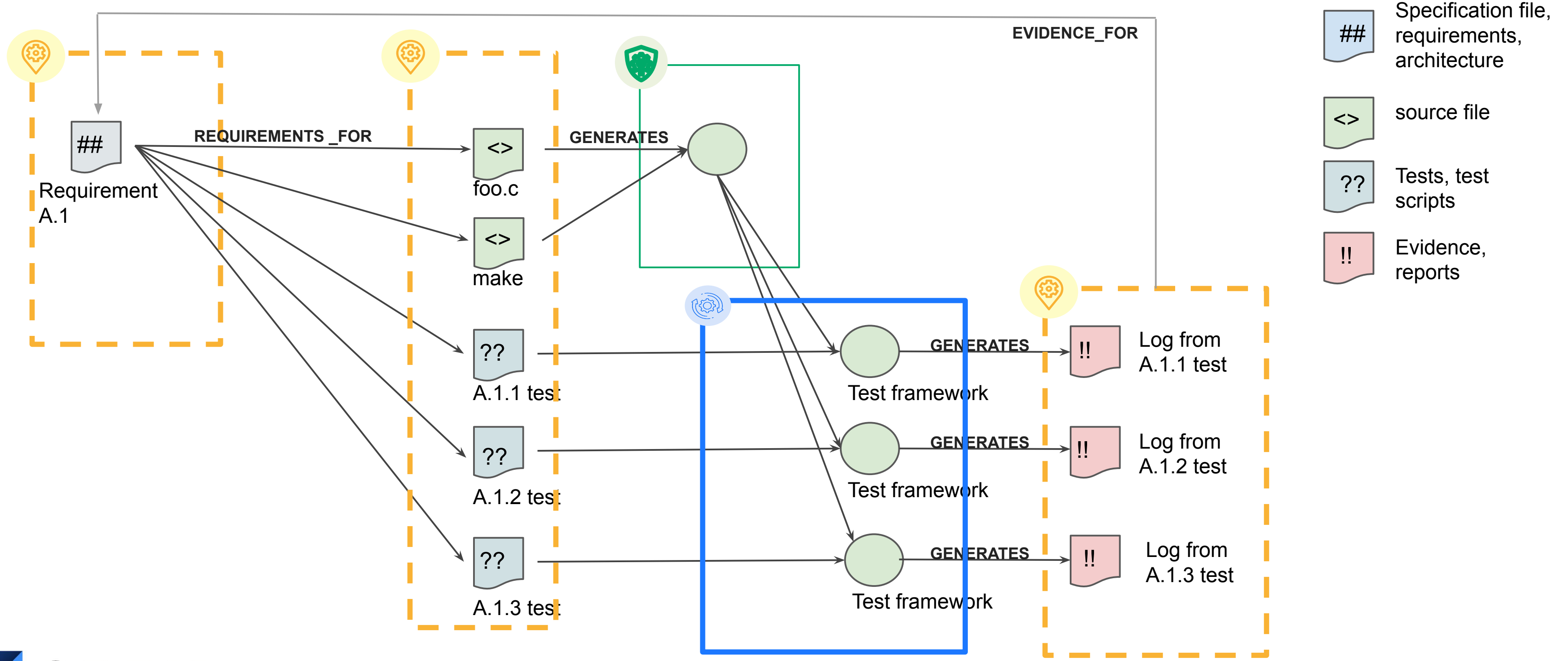
For more details see: https://spdx.github.io/spdx-spec/v2.3/relationships-between-SPDX-elements/

Requirement Traceability



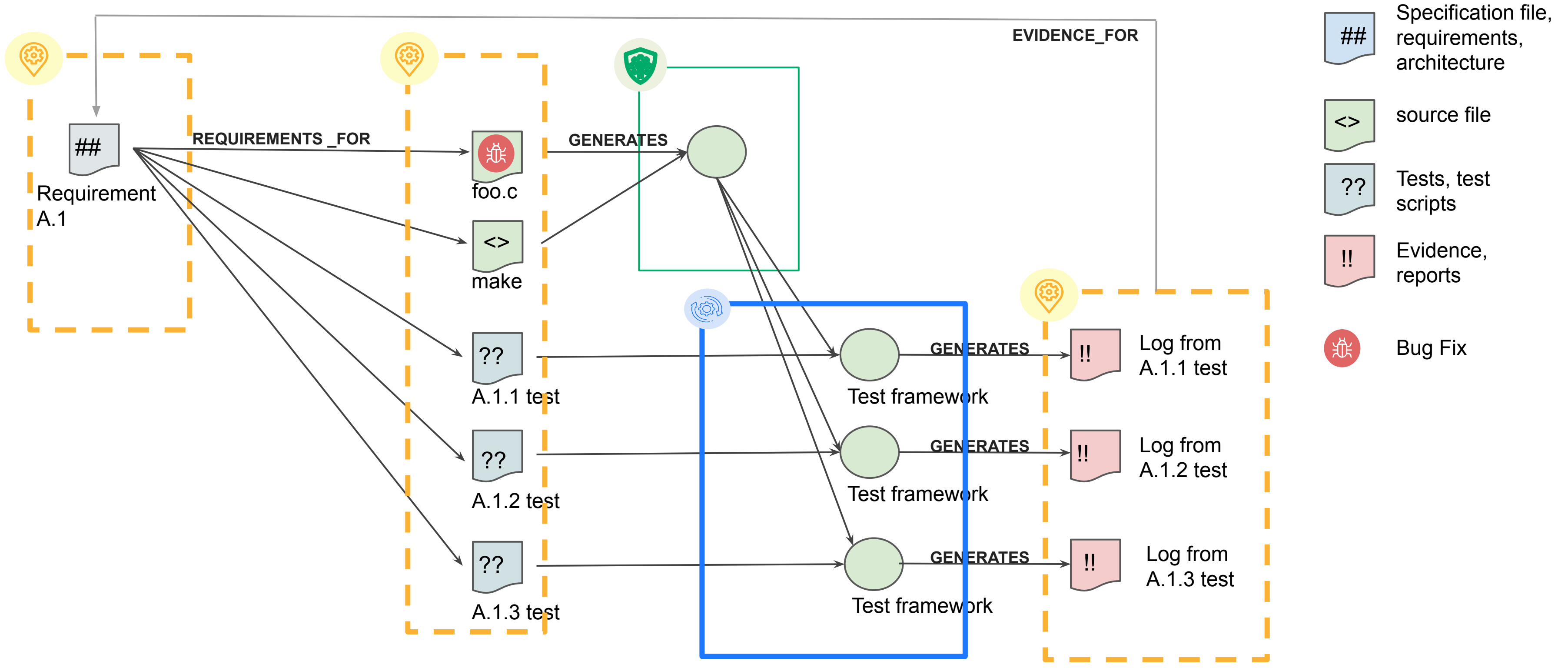
Traceability

Requirement to Code to Tests to Evidence



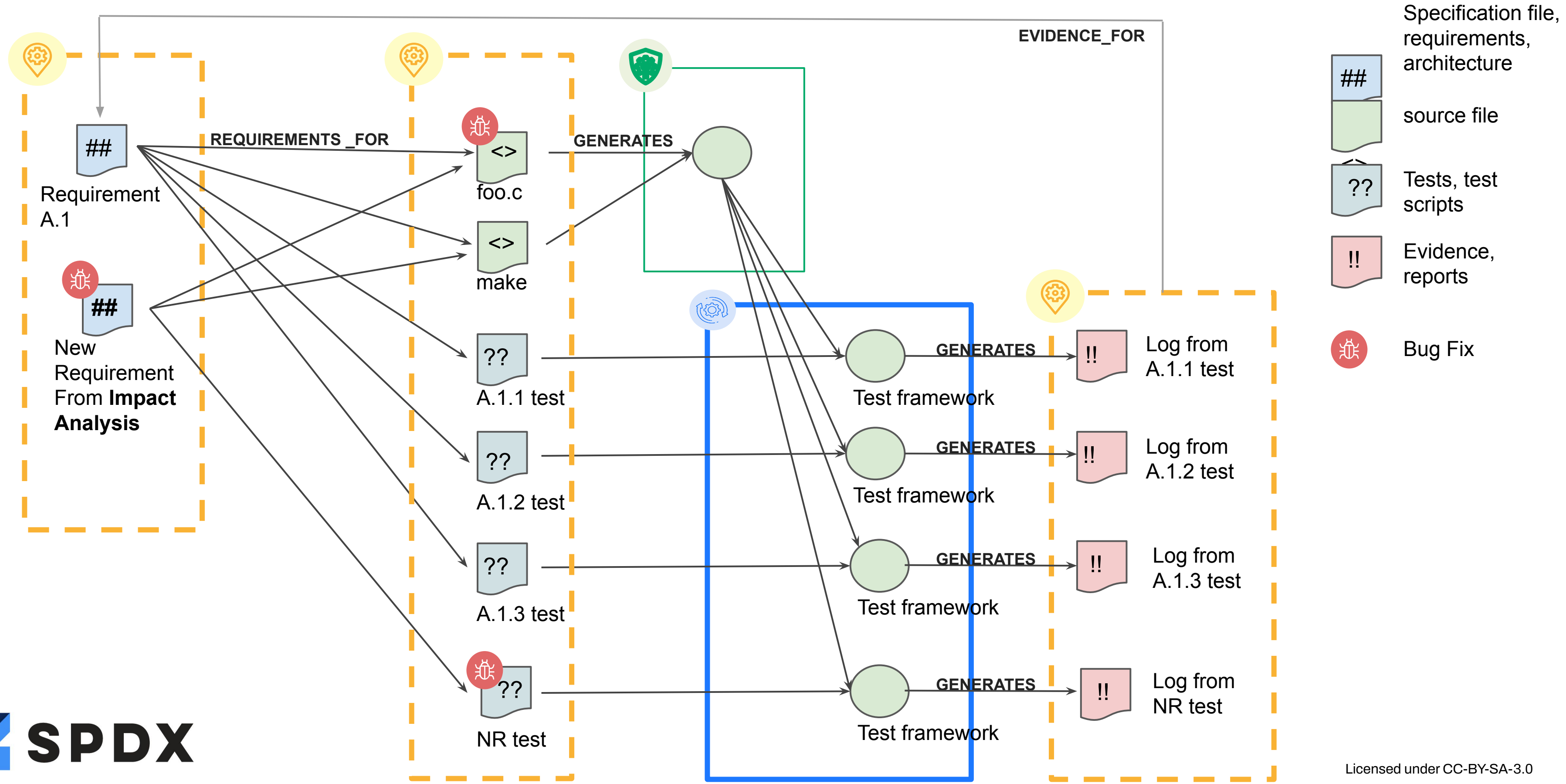
Traceability

Requirement to Code to Tests to Evidence



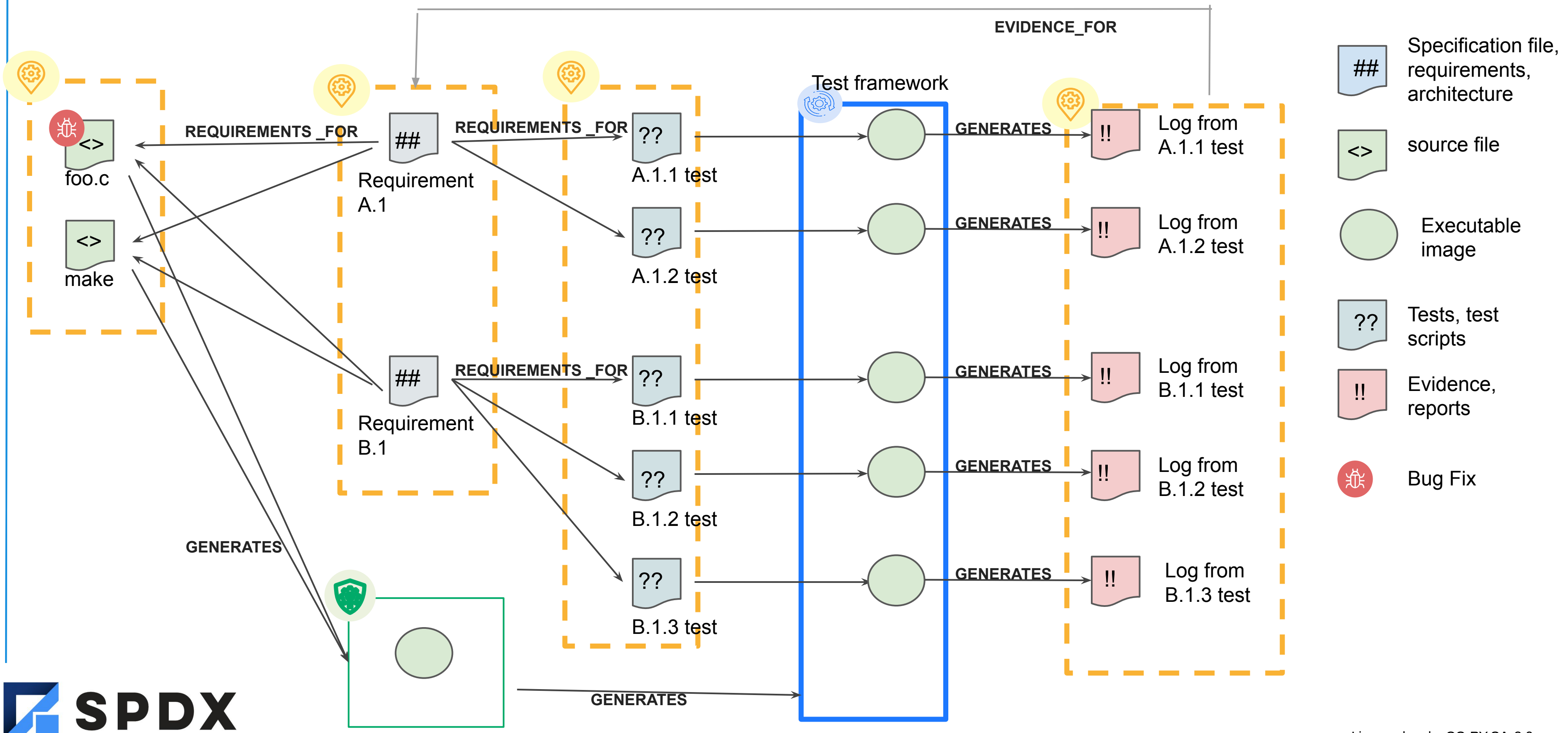
Traceability

New Requirement to Code to Tests to Evidence



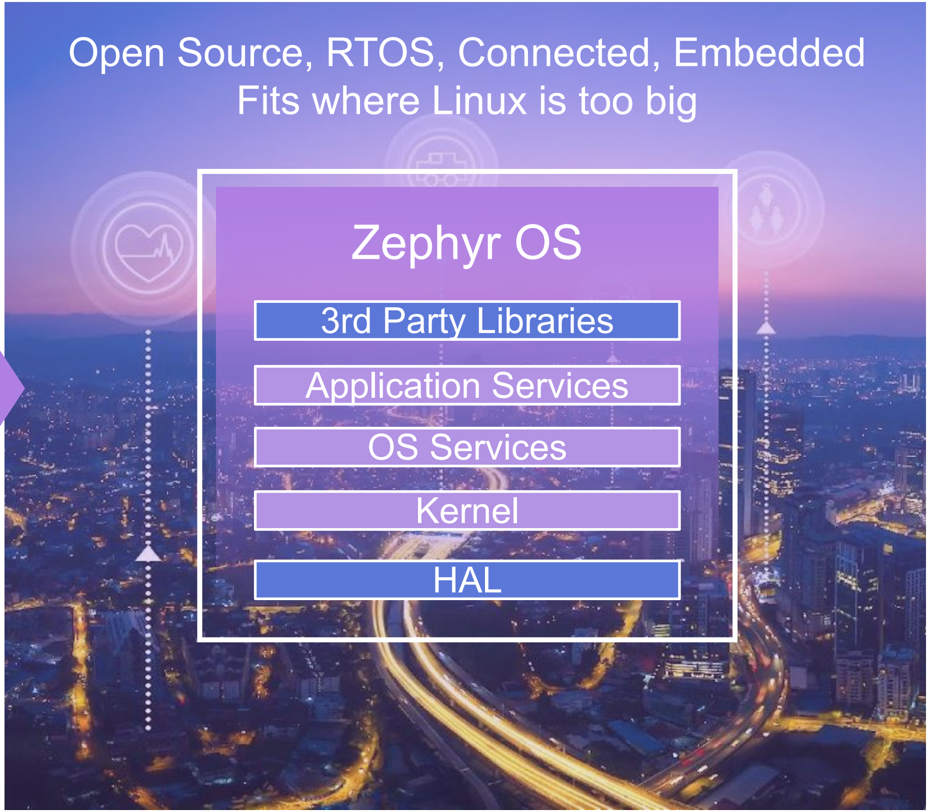
Traceability

Code to Requirements to Tests to Evidence



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



THE LINUX FOUNDATION PROJECTS

Zephyr Project

Software Architectural Element

Zephyr Project:

- Embedded RTOS
- Build system
- Test cases & Test framework

Plus evidences for (safety) systematic capability:

- Functional Safety Management plans
- Safety Analysis
- Completeness, Compliance & (Test & Analysis) Coverage Evidences