Why do we Need SPDX for Functional Safety?

Nicole Pappler  
CTO,  
AlektoMetis.com

Kate Stewart  
VP Dependable Embedded Systems,  
The Linux Foundation
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.
So in Functional Safety (FuSa)...

- We talk a lot about safety, risk and harm
- With safety being the freedom from unacceptable risk
- Risk is the combination of the probability of harm done and the severity of that harm
- With harm we always mean the injury or death of one or more persons, impact on the environment, damage of goods etc.

- Overall goal in FuSa: Minimise the risk to an acceptable level
- Acceptable risk? Risk accepted by society, e.g. risk that is considered to be part of everyday life
What is FuSa aiming for?

Safety Architecture and Documentation

Suitable, robust system concept and architecture

Analysis, Reviews and Tests

Processes for development, verification, build, deployment and maintenance (according to Safety Standards like IEC 61508)

Loads of documentation and evidences

Safety Plan
Verification Plan
Requirements
SW Architecture & Design
Coding Guidelines
Test Cases
Test Reports
Code
Calibration Data
Functional Safety - systematic capability

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
Functional Safety - Standards

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!
Dependencies in a FuSa Project
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
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)

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**

How can SPDX support FuSa projects?

**Engineers like to engineer!**
- Creating a fantastic system
- Maintaining the fantastic system
- Applying a process to do so
- Ensuring all documentation and evidences are consistent
How can SPDX support FuSa projects?

**Engineers like to engineer!**

- Creating a fantastic system :-)  
- Maintaining the fantastic system :-)  
- Applying a process to do so :-)  
- Ensuring all documentation and evidences are consistent *no-no-no-no*
How can SPDX support FuSa projects?

Engineers like to engineer!
- Creating a fantastic system :-) 
- Maintaining the fantastic system :-) 
- Applying a process to do so :-/ 
- Ensuring all documentation and evidences are consistent *no-no-no-no*
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…

- Plans
- Processes
- Guidelines
- Requirements
- Specifications
- Code, Build data, executables
- Verification
- Analysis
- Test
- Evidences

One or more repos, git or svn based

Zoo of lifecycle management systems, .pdf, .docx

Zoo of lifecycle management systems and test tools, .pdf, .docx, .xls, html, code...
Data Structure of current FuSa projects...

- Plans
- Processes
- Guidelines
  - .pdf, .docx, QMS System, Wikis

- Requirements Specifications
  - Zoo of lifecycle management systems, .pdf, .docx

- Code, Build data, executables
  - One or more repos, git or svn based

- Verification Analysis Test Evidences
  - Zoo of lifecycle management systems and test tools, .pdf, .docx, .xls, html, code...

- Traceability breaks between tools, between configurations, etc, impossible to keep up during updates and product variants
Generate SBOMS when the data is known

- Source SBOM
- Design SBOM
- Runtime SBOM
- Build SBOM
- Deployed SBOM
<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>

Source: Types of Software Bills of Materials (SBOM) published by CISA on 2023/4/21
## Managing set of relevant items with SBOMs

<table>
<thead>
<tr>
<th>SBOM Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Design SBOM</strong></td>
<td>Functional Safety Management (Plans) and Safety Concept</td>
</tr>
<tr>
<td><strong>Source SBOM</strong></td>
<td>Requirements, Design, Safety Analysis, Source Code, Test Cases</td>
</tr>
<tr>
<td><strong>Build SBOM</strong></td>
<td>Build Framework, Build configuration and environment data, Test Framework, Executable, Test Reports</td>
</tr>
<tr>
<td><strong>Deploy SBOM</strong></td>
<td>Deployed configuration and environment data, Hardware architecture specific information and data, deployment tests and reports</td>
</tr>
<tr>
<td><strong>Runtime SBOM</strong></td>
<td>Runtime relevant data (configuration data), training data, error logging data</td>
</tr>
</tbody>
</table>
## SPDXX Relationships to Clarify Dependencies

<table>
<thead>
<tr>
<th>DESCRIBES</th>
<th>DEPENDENCY_OF</th>
<th>PREREQUISITE_FOR</th>
<th>GENERATES</th>
<th>VARIANT_OF</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESCRIBED_BY</td>
<td>RUNTIME_DEPENDENCY_OF</td>
<td>HAS_PREREQUISITE</td>
<td>TEST_OF</td>
<td>FILE_ADDED</td>
</tr>
<tr>
<td>CONTAINS</td>
<td>BUILD_DEPENDENCY_OF</td>
<td>ANCESTOR_OF</td>
<td>TEST_TOOL_OF</td>
<td>FILE_DELETED</td>
</tr>
<tr>
<td>CONTAINED_BY</td>
<td>DEV_DEPENDENCY_OF</td>
<td>DESCENDENT_OF</td>
<td>TEST_CASE_OF</td>
<td>FILE_MODIFIED</td>
</tr>
<tr>
<td>DYNAMIC_LINK</td>
<td>OPTIONAL_DEPENDENCY_OF</td>
<td>DOCUMENTATION_OF</td>
<td>EXAMPLE_OF</td>
<td>PATCH_FOR</td>
</tr>
<tr>
<td>STATIC_LINK</td>
<td>PROVIDED_DEPENDENCY_OF</td>
<td>BUILD_TOOL_OF</td>
<td>METAFILE_OF</td>
<td>PATCH_APPLIED</td>
</tr>
<tr>
<td>AMENDS</td>
<td>TEST_DEPENDENCY_OF</td>
<td>EXPANDED_FROM_ARCHIVE</td>
<td>PACKAGE_OF</td>
<td>REQUIREMENT_FOR</td>
</tr>
<tr>
<td>COPY_OF</td>
<td>OPTIONAL_COMPONENT_OF</td>
<td>DISTRIBUTION_ARTIFACT</td>
<td>DATA_FILE_OF</td>
<td>SPECIFICATION_FOR</td>
</tr>
<tr>
<td>DEPENDS_ON</td>
<td>DEPENDENCY_MANIFEST_OF</td>
<td>GENERATED_FROM</td>
<td>DEV_TOOL_OF</td>
<td>OTHER</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/)
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
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
Zephyr Safety:
Design SBOM to Source SBOM
Zephyr Safety
Source SBOM to Build SBOM

**SPECIFICATION FOR**
- Zephyr Verification Plan
- Zephyr Configuration & Change Management Plan

**REQUIREMENT FOR**
- Software Component Design Specifications

**SPECIFICATION FOR**
- Coding Guidelines
  - Code review (Static Analysis)

**SPECIFICATION FOR**
- Source Code
  - Test Of: Component Tests
  - Generates: Executable image

**SPECIFICATION FOR**
- Executable image
  - Test Of: Software Tests

**SPECIFICATION FOR**
- Software Build Chain Specification
  - Specification file, requirements, architecture

**SPECIFICATION FOR**
- Integr. Test Framework Specification
  - Tests, test scripts, verification

**SPECIFICATION FOR**
- Evidence, reports

**SPECIFICATION FOR**
- Plans, Guidelines, Process

**SPECIFICATION FOR**
- Source file

**SOURCE SBOM TO BUILD SBOM**

Source SBOM to Build SBOM

**LICENSED UNDER CC-BY-SA-3.0**
Dependency Identification on Component Level

- **Specification file**, requirements, architecture
- **Source file** <-> **Tests, test scripts, verification**
- **Evidence, reports**
- **Plans, Guidelines, Process**
- **Executable image**

**Software Build Chain Specification**

**Integr. Test Framework Specification**

**Executable image**

**Software Tests**

**Software Requirements Specification**

**Component Tests**

**Source Code**

**Code review (Static Analysis)**

**Zephyr Verification Plan**

**Zephyr Configuration & Change Management Plan**

**Software Component Design Specifications**

**SPDX**

Licensed under CC-BY-SA-3.0
Dependency Identification on Component Level

- **Specification_file**, requirements, architecture
- Source file
- Tests, test scripts, verification
- Evidence, reports
- Plans, Guidelines, Process
- Executable image

**Software Tests**

- **Software Build Chain Specification**
- **Integr. Test Framework Specification**
Dependency Identification on Component Level

**REQUIREMENT FOR**
- Specification file
- Requirements
- Architecture

**source file <>**
- Tests, test scripts
- Verification
- Evidence, reports

**Evidence, reports**
- Plans, Guidelines, Process
- Executable image

**SPECIFICATION FOR**
- Coding Guidelines
- Code review (Static Analysis)
- Integr. Test Framework Specification
- Software Build Chain Specification

**TEST OF**
- Executable image
- Software Tests

**EXECUTABLE IMAGE**
- Software Generation
- Component Tests
- Component Design Specifications
- Zephyr Configuration & Change Management Plan
- Zephyr Verification Plan

**License:** Licensed under CC-BY-SA-3.0
Dependency Identification on Component Level

- REQUIREMENT FOR Specification file, requirements, architecture
- Source file
- Tests, test scripts, verification
- Evidence, reports
- Plans, Guidelines, Process
- Executable image

- SPECIFICATION FOR Software Tests
- SPECIFICATION FOR Executable image
- SPECIFICATION FOR Test Framework Specification
- SPECIFICATION FOR Software Build Chain Specification

- TEST_OF Executable image
- TEST_OF Component Tests
- TEST_OF Source Code
- TEST_OF Code review (Static Analysis)
- TEST_OF Coding Guidelines

- REQUIREMENT FOR Software Component Design Specifications
- REQUIREMENT FOR Zephyr Verification Plan
- REQUIREMENT FOR Zephyr Configuration & Change Management Plan

- SPECIFICATION FOR (Software Requirements Specification)

Licensed under CC-BY-SA-3.0
Dependency Identification on Component Level

**Requirement**

- Specification file
- Requirements, architecture
- Source file
- Tests, test scripts, verification
- Evidence, reports
- Plans, guidelines, process
- Executable image

**Specification Chain**

- Software Requirements Specification
- Test Framework Specification
- Software Build Chain Specification
- Executable image

**Generation**

- Software Tests
- (Software Requirements Specification)

**Dependencies**

- Code review (Static Analysis)
- Requirement_for: Zephyr Verification Plan
- Requirement_for: Zephyr Configuration & Change Management Plan
- Requirement_for: Software Component Design Specifications
- Requirement_for: Component Tests
- Requirement_for: Source Code
- Requirement_for: Coding Guidelines
Requirement Traceability
Traceability
Requirement to Code to Tests to Evidence

**Requirement A.1**

- Requirement A.1
- foo.c
- make

**Test Framework**

- A.1.1 test
- A.1.2 test
- A.1.3 test

**Log from**

- A.1.1 test
- A.1.2 test
- A.1.3 test

**Evidence, reports**
Traceability
Code to Requirements to Tests to Evidence

- **Requirement A.1**: source file foo.c
- **Requirement B.1**: make

**Tests, test scripts**

**Traceability Chart**

- **Requirement A.1**: GENERATES REQUIREMENTS_FOR A.1.1 test
- **Requirement B.1**: GENERATES REQUIREMENTS_FOR B.1.1 test
- **Requirement B.1**: GENERATES REQUIREMENTS_FOR B.1.2 test
- **Requirement B.1**: GENERATES REQUIREMENTS_FOR B.1.3 test

**Test framework**

- **Log from A.1.1 test**: GENERATES EVIDENCE_FOR
- **Log from A.1.2 test**: GENERATES EVIDENCE_FOR
- **Log from B.1.1 test**: GENERATES EVIDENCE_FOR
- **Log from B.1.2 test**: GENERATES EVIDENCE_FOR
- **Log from B.1.3 test**: GENERATES EVIDENCE_FOR

**Evidence, reports**
Conclusions

Using a SPDX Safety Profile

- Provides a complete model of dependencies in a safety related project
- 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
Questions?

To join in evolving SPDX safety profile:

- Subscribe to: https://lists.spdx.org/g/spdx-fusa

Contact:

- Nicole Pappler - nicole@alektometis.com
- Kate Stewart - kstewart@linuxfoundation.org