The SPDX Safety Profile

Nicole Pappler
CTO,
AlektoMetis.com

Stanislav Pankevich
Software Architect,
Reflex Aerospace GmbH
Safety Analysis is Performed on Systems

Safety Critical Application (e.g. from CIP, AGL, Energy, etc.)

Other RTOS (e.g. Zephyr)

Linux

RTOS (e.g. Zephyr)

HW-Virtualization (e.g. Xen)

μC

μP

Tooling (e.g. Yocto)
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

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
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!
What is FuSa aiming for?
Safety Architecture and Documentation

Suitable, robust system concept and architecture

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

Analysis, Reviews and Tests

Loads of documentation and evidences

Safety Plan
Verification Plan
Requirements
SW Architecture & Design
Coding Guidelines
Test Cases
Test Reports
Code
Calibration Data
## Dependencies in a FuSa Project

<table>
<thead>
<tr>
<th>REQUIREMENTS MANAGEMENT PLAN</th>
<th>REQUIREMENTS MANAGEMENT PLAN</th>
<th>CONFIGURATION MANAGEMENT PLAN</th>
<th>DOCUMENTATION MANAGEMENT PLAN</th>
<th>COMPONENT QUALIFICATION / SUPPLY CHAIN</th>
<th>VALIDATION &amp; QUALIFICATION</th>
<th>TOOLING EVAL &amp; QUALIFICATION (DEV, VERIFICATION, BUILD, DEPLOY...)</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQUIREMENTS MANAGEMENT PLAN</td>
<td>REQUIREMENTS MANAGEMENT PLAN</td>
<td>CONFIGURATION MANAGEMENT PLAN</td>
<td>DOCUMENTATION MANAGEMENT PLAN</td>
<td>COMPONENT QUALIFICATION / SUPPLY CHAIN</td>
<td>VALIDATION &amp; QUALIFICATION</td>
<td>TOOLING EVAL &amp; QUALIFICATION (DEV, VERIFICATION, BUILD, DEPLOY...)</td>
</tr>
</tbody>
</table>

### Dependencies

- **Requirements**
- **Architecture & Design**
- **Implementation (Code)**
- **Unit Verification & Tests**
- **Software Tests**
- **Integration & Tests**
- **Reports**
- **Functional Safety Management Plan**
- **Configuration Management Plan**
- **Documentation Management Plan**
- **Component Qualification / Supply Chain**
- **Validation & Assessment**
- **Tooling Eval & Qualification (Dev, Verification, Build, Deploy...)**
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 enable 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

- **Requirements**
- **Architecture & Design**
- **Implementation (Code)**
- **Unit Verification & Tests**
- **Integration & Tests**
- **Software Tests**
- **Reports**
- **Functional Safety Management Plan**
- **Requirements Management Plan**
- **Configuration Management Plan**
- **Documentation Management Plan**
- **Component Qualification / Supply Chain**
- **Validation & Assessment**
- **Tooling Eval & Qualification (Dev, Verification, Build, Deploy...)**
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)

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**

**Maintenance and Promotion of Safety Principles**

- **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)

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

- Code, Build data, executables

- Requirements Specifications

- 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
- Code, Build data, executables
- Requirements
- Specifications
- Verification
- Analysis
- Test
- Evidences
- Zoo of lifecycle management systems
- .pdf, .docx
- QMS System
- Wikis
- One or more repos, git or svn based
- Traceability breaks between tools, between configurations, etc, impossible to keep up during updates and product variants
No 1 Safety Information Exchange Format

draft_2005TemplateSafetyCase_thisproject_final_forTraceingv06.xls
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 Dev Plan
- Zephyr Verification Plan
- Zephyr Configuration & Change Management Plan
- Software Requirements Specifications
- Software Component Design Specifications
- Coding Guidelines

Specifications:
- Code review (Static Analysis)
- Source Code
- Component Tests

Evidence:
- Component test reports
- Static analysis scan reports
- Specification file, requirements, architecture
- Tests, test scripts, verification
- Plans, Guidelines, Process

Requirements:
- Safety Requirements Specifications
- Safety Component Design Specifications

Licensed under CC-BY-SA-3.0
Zephyr Safety:
Design SBOM to Source SBOM

Zephyr Safety Dev Plan (SDoc)
Zephyr Safety Overview (rst)
Zephyr Verification Plan
Zephyr Configuration & Change Management Plan
High Level Requirement

Coding Guidelines
Code review (Static Analysis)
Static analysis scan reports
Source Code

Component Tests
Component test reports

Requirements, architecture
Source file
Tests, test scripts, verification
Evidence, reports
Plans, Guidelines, Process

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

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

**SPECIFICATION FOR**
- Software Tests
- Integr. Test Framework Specification
- Software Build Chain Specification
- Code review (Static Analysis)
- Source Code
- Component Tests
- Executable image

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

**SOFTWARE COMPONENT DESIGN SPECIFICATIONS**

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

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

<< SPECIFICATION FOR >>
** Coding Guidelines
?? Code review (Static Analysis)

<< SPECIFICATION FOR >>
** Source Code
<< TEST_OF >>
Component Tests

<< SPECIFICATION FOR >>
?? Executable image

<< SPECIFICATION FOR >>
## Software Build Chain Specification
## Integr. Test Framework Specification

<< SPECIFICATION FOR >>
** Software Tests

<< REQUIREMENT FOR >>
Zephyr Verification Plan

<< REQUIREMENT FOR >>
Zephyr Configuration & Change Management Plan

<< REQUIREMENT FOR >>
Software Component Design Specifications

<< REQUIREMENT FOR >>
(Software Requirements Specification)
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 Test Framework Specification

<< source file

?? Tests, test scripts, verification

?? Evidence, reports

## SPECIFICATION_FOR Software Build Chain Specification

<< Executable image

## SPECIFICATION_FOR Component Tests

<< Software Component Design Specifications

## SPECIFICATION_FOR Code review (Static Analysis)

<< Coding Guidelines

<< Zephyr Verification Plan

<< Zephyr Configuration & Change Management Plan

## SPECIFICATION_FOR executable image

<< Software Build Chain Specification

<< Test Framework Specification

<< Component Tests

<< Source Code

<< Specification file, requirements, architecture
Dependency Identification on Component Level

- Zephyr Safety Dev Plan
- Zephyr Verification Plan
- Zephyr Requirements Management Plan
- Zephyr Configuration & Change Management Plan
- Software Requirements Specifications
- Software Component Design Specifications

- Specification: Coding Guidelines
  - Requirements: Code review (Static Analysis)
  - Evidence: Static analysis scan reports

- Specification: Source Code
  - Test: Component Tests
    - Evidence: Component test reports
Dependency Identification on Component Level

** REQUIREMENT FOR **
- Software Requirements Specifications

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

** TEST OF **
- Code review (Static Analysis)
  - Executable image
- Component Tests

** GENERATES **
- Executable image
- Software Tests
- Software Build Chain Specification
- Test Framework Specification

** SOURCE CODE **
- Coding Guidelines
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
- Key question: How to make requirements useful for open source software?
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
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
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).
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
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

Requirements

Architecture & Design

Implementation (Code)

Unit Verification & Tests

Integration & Tests

Software Tests

Reports

Functional Safety Management Plan

Requirements Management Plan

Configuration Management Plan

Documentation Management Plan

Component Qualification / Supply Chain

Validation & Assessment

Tooling Eval & Qualification (Dev, Verification, Build, Deploy...)
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

- Source SBOM
- Design SBOM
- Build SBOM
- Deployed SBOM
- Runtime SBOM
# SBOM Types - manage your work products

<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>SBOM 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>

## Managing set of relevant items with SBOMs

<table>
<thead>
<tr>
<th>SBOM Type</th>
<th>Items Included</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design SBOM</td>
<td>Functional Safety Management (Plans) and Safety Concept</td>
</tr>
<tr>
<td>Source SBOM</td>
<td>Requirements, Design, Safety Analysis, Source Code, Test Cases</td>
</tr>
<tr>
<td>Build SBOM</td>
<td>Build Framework, Build configuration and environment data, Test Framework, Executable, Test Reports</td>
</tr>
<tr>
<td>Deploy SBOM</td>
<td>Deployed configuration and environment data, Hardware architecture specific information and data, deployment tests and reports</td>
</tr>
<tr>
<td>Runtime SBOM</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/)
Traceability

Requirement to Code to Tests to Evidence

---

**Requirement:**

A.1

**Code:**

foo.c

**Tests:**

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

**Test Frameworks:**

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

**Evidence:**

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

---
Traceability
Requirement to Code to Tests to Evidence

- Requirement A.1
- foo.c
- make
- A.1.1 test
- A.1.2 test
- A.1.3 test
- Test framework
- Test framework
- Test framework
- Log from A.1.1 test
- Log from A.1.2 test
- Log from A.1.3 test
Traceability
New Requirement to Code to Tests to Evidence

Requirement A.1

New Requirement From Impact Analysis

Requirement A.1

foo.c
make

Test framework

GENERATES

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

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
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