
ELISA
Enabling Linux in
Safety Applications
WORKSHOP
ELISA Workshop
Munich, Germany
November 18-20, 2025
Co-hosted with Red Hat

Open Functional Safety: Safety-Qualified Lifecycle with Sphinx

WORKSHOP

About me

- Senior Functional Safety and Cybersecurity Consultant

- 3 years experience at Fraunhofer ESK as Embedded Systems Designer Hardware/Software
- More than 8 Jahre experience as an Assessor at TÜV SÜD for Functional Safety in Industrial and Automotive Domains for Generic Safety Components
- Main topics Operating Systems/Hypervisors, Middleware, Libraries, Tools and Semiconductors/IP
- Trainer for IEC 61508, ISO 26262 and ISO 13849 and member of German ISO 26262-6 (SW) committee

Agenda

- Challenge for Safety Compliance for small/medium and open source communities
- A possible solution: Lifecycle with Open Source Toolchain
- Classification and qualification

Challenges for Safety Compliance

Challenges for Safety Compliance

- Variety of standards
  - ISO 26262
  - IEC 61508
  - ISO 13849
  - IEC 62061
  - ….
- Different kinds of software
  - Firmware (C)
  - Libraries for Application Software (C, LVL)
  - Application Software (LVL, e.g. Function Blocks etc.)

Challenges for Safety Compliance

- Industrial vs. Automotive Functional Safety
  - Companies starting from 5 to 200 people
  - Sometimes only 1 or 2 SW Engineers
- As the teams/companies are small, sometimes really very limited processes
- When dealing with ISO 13849 (coming from first edition) not even a safety plan in safety projects
- Therefore also no clear processes on requirements, design etc.

Challenges for Safety Compliance

- What do we typically see?
- “Best case” Requirements are kept in some Word and Excel documents
- Traceability is done manually
- Maybe some VBA added…
- Or an Excel list….
- Or a PowerPoint
- Or a napkin…..

Challenges for Safety Compliance

- Company sizes do not allow for expensive tooling
- Same applies to Open Source Communities
- Requirements Mgmt. 10k€ per seat
- Design Tools similar dimension
- Compiler, test tools etc. another 20k€

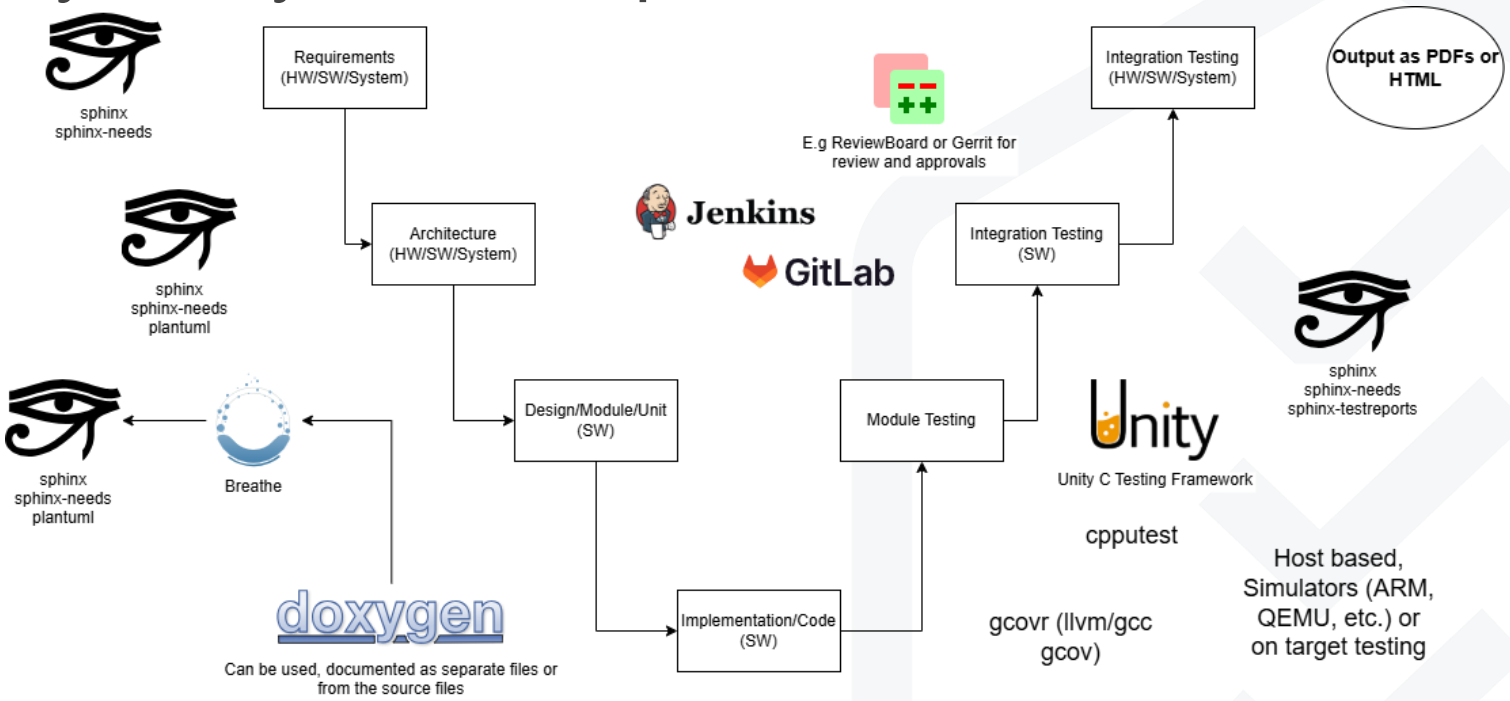
- But the company only has 2 developers.... Not feasible
- Same for Open Source, not all tools available

A possible solution: Lifecycle with Open Source Toolchain

Safety Lifecycle with Open Source

- A possible lifecycle with a “preset” of Open Source tools in a complete toolchain
- Just a proposal of tools, many can be exchanged depending on the project
- Showing possible ways of qualification of the different tools

Safety Lifecycle with Open Source



Safety Lifecycle with Open Source

- Documentation is often time consuming and difficult to manage
- Therefore often inconsistent

- Selecting a base that allows for having a complete picture of the project
- Can serve as the overall documentation package for assessors as well

- Keeping it as close to the source code as possible
- X-as-Code

Safety Lifecycle with Open Source – Sphinx

- Why Sphinx?
- Quite established, HTML and PDF outputs

- reStructuredText is a middle ground regarding complexity
- Supported by pandoc allowing converting existing templates/documentation to reST
- Yet System/HW side usage is depending on users or additional (not yet existing…) GUI tooling
- Easily extended via plugins

- Example on tool side: Ferrocene, Safe Rust toolchain uses Sphinx for their safety documentation (https://github.com/ferrocene/ferrocene

Safety Lifecycle with Open Source – Sphinx-Needs

- Extends Sphinx with the ability to mange “needs” and their links
- Needs can be everything (user defined) from System Requirements to Detailed Design or Test Cases (called Roles)
- Now supports also Schema Validation with strict typing system
- Additional tooling for easier editing in VSCode available from useBlocks, live preview without building, navigating etc. (free for Open Source)

```
.. req:: User needs to login
   :id: ID123
   :status: open
   :tags: user;login
   :collapse: false
```

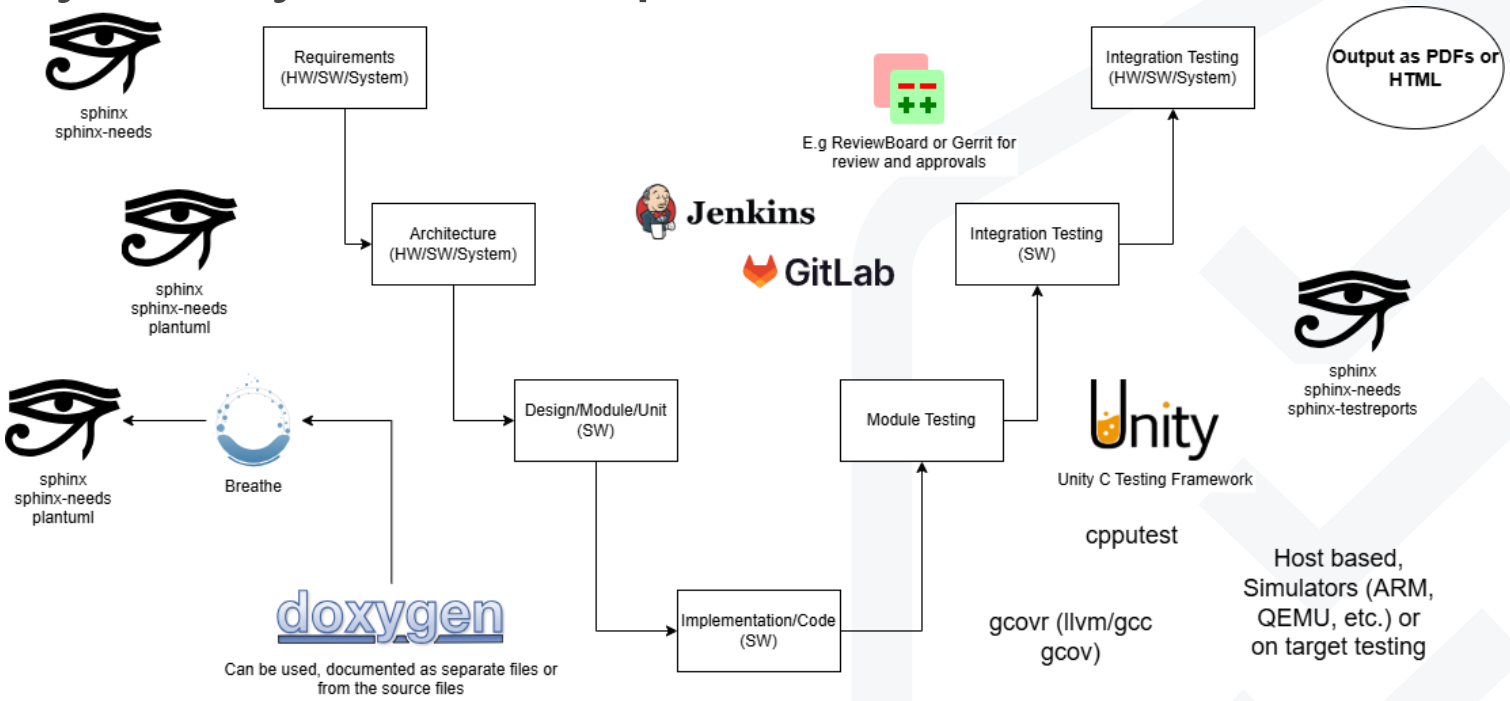
ELISA Enabling Linux in Safety Applications

WORKSHOP

Safety Lifecycle with Open Source – Automation

- To always have up to date information everything should be automated
- Everything lives in version control / configuration management, e.g. git or svn
- Automation environment can be GitHub, GitLab or Jenkins
- Depending on that selection, additional tooling for review and merging might be required
- ReviewBoard or gerrit not only for code review, but also requirements/etc.

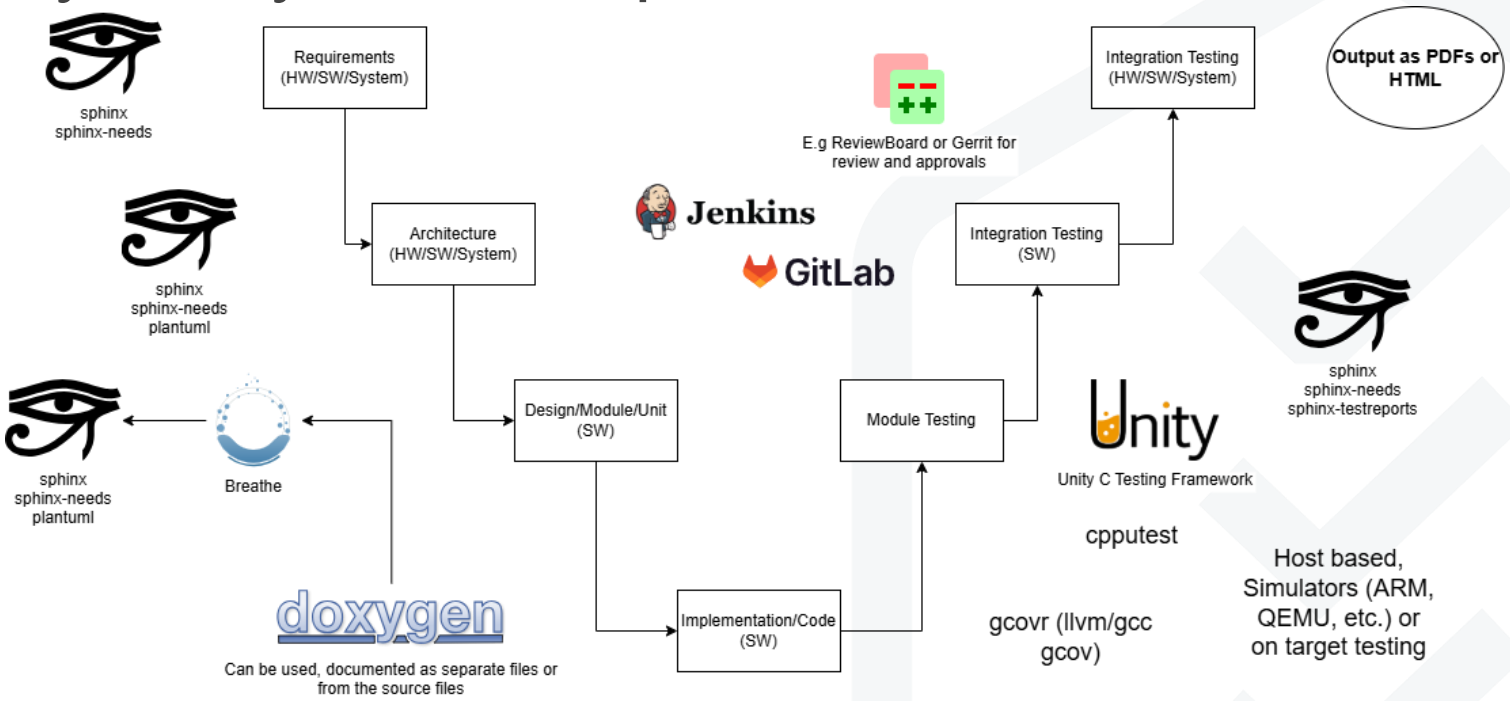
Safety Lifecycle with Open Source



Safety Lifecycle with Open Source – Unit/Source

- Specification on Unit Level either as part of the source files or separate files
- If together with the source, also (maybe already familiar) Doxygen Syntax can be used and extracted
- Breathe or sphinx-codelinks
- Traceability to unit design by module and function name

Safety Lifecycle with Open Source

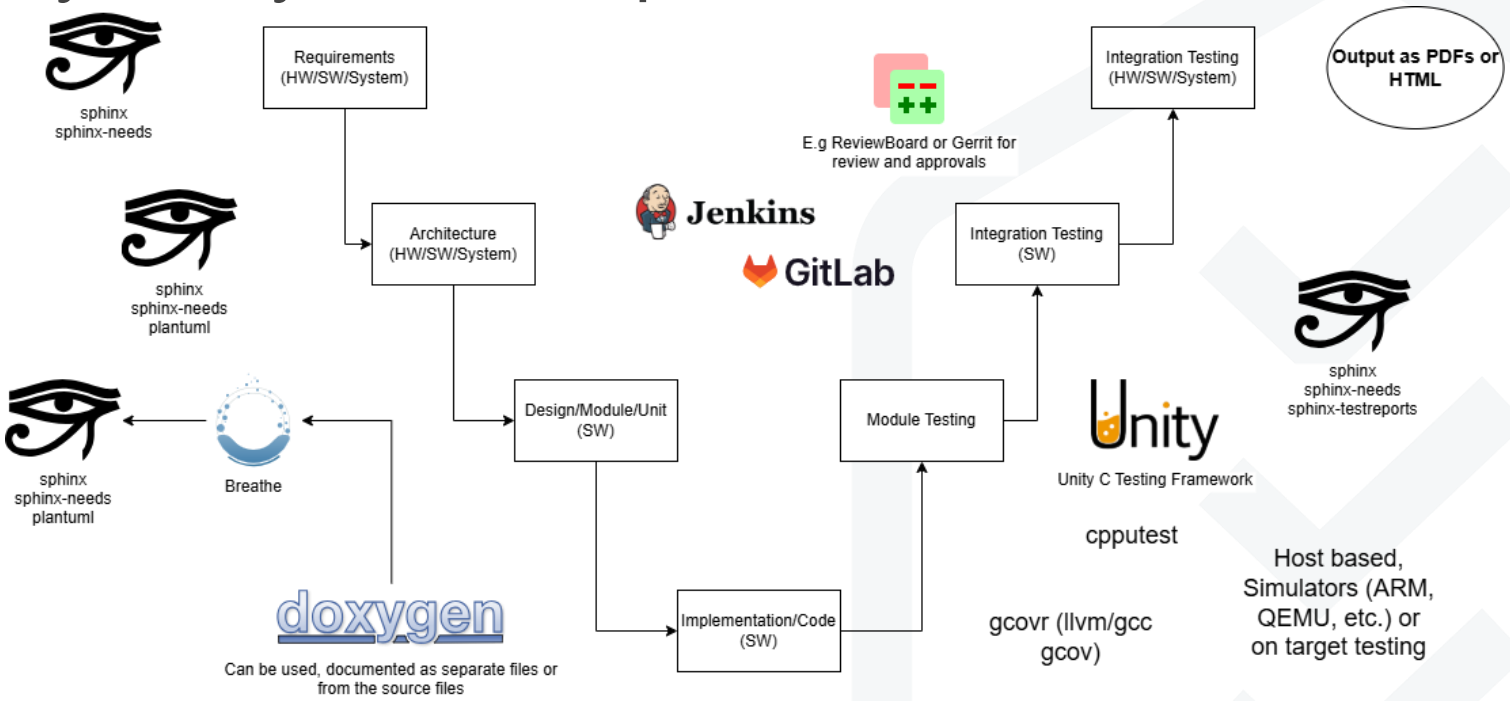


Safety Lifecycle with Open Source – Static Analysis

- Not that easy… not many open source tools available

- CPPCheck used in our demo/preset
- Ruleset depends on industry (MISRA or custom)

Safety Lifecycle with Open Source



Safety Lifecycle with Open Source – Compiler

- Very project dependent
- Sometimes controller specific tools must be used
- Sometimes companies already familiar with their existing commercial toolchain

- For unit/integration level testing also different compilers could be used (x86/host based)

- Otherwise either gcc or llvm typically used
- Small to medium projects often do not require libraries, otherwise thinking about the qualification of these as well!

Safety Lifecycle with Open Source – Dynamic Tests

- Many open source frameworks are available
- Often CPPUTest for C++ projects, many developers already familiar with it
- A bit more difficult to port for on-target execution

- Unity Test Framework for C
- Similar Syntax, more lightweight (only 2 files) and therefore easier to port

- Both can output JUNIT XML results for integration into automation

Safety Lifecycle with Open Source – Test Env.

- Heavily depend on the controllers/env.

- Host based (x86)
- QEMU
- Vendor tooling for specific controllers
- On-target testing

Safety Lifecycle with Open Source – Coverage

- Code coverage during dynamic testing

- Often commercial compiler/IDE integrates code coverage collection as well
- Open Source side typically integrated with compiler (gcc or llvm, e.g. gcov)

- Easy to collect during host based testing
- But on-target or at least during simulator would provide justifications for the compiler qualification later on
- Coverage can be collected via JTAG probes

Safety Lifecycle with Open Source – Test Results

- Sphinx-testreports
- Adds the ability to link results from JUNIT XML tests (unit/integration)
- As well as custom JSON test result files from other level
- Integration Tests / etc. depend again heavily on type of development
- Often existing tooling for test mgmt. is present which can be bridged via the JSON import

Classification and Qualification

Classification and Qualification

- ISO 13849: “Tools with confidence from use”
  - References to the long-term existence of the projects
  - Usage examples, success stories and development history
  - “e.g. programmers are trained to use the tools” via providing training, user manual, templates etc. to the customer
- So in ISO 13849 only projects, this would already be “enough”

Safety Lifecycle with Open Source

- IEC 61508: Classification and Validation
- Classification of tools in the second edition still based on static classes:
  - T1 Editors, **Requirement Management Tools** etc.
  - T2 could fail to reveal faults in the software
  - T3 can directly/indirectly influence the final binary

- For T1 (Sphinx, Sphinx-Needs) no further action required
- But…

Safety Lifecycle with Open Source

- IEC 61508: Classification and Validation
- 3rd Edition could take a more ISO 26262 based approach
- Impact Classification: TI1 lowest -> No qualification required (“negligible impact compared to human errors”)
- But even if TI2 (traceability checks etc.) is chosen, maximum TIL2 for SIL3 (regardless of TD)
- Main method is then “Tool integration and validation in usage context (1.3.4)”

Safety Lifecycle with Open Source – Sphinx

- Increased confidence from use
  - …. Not really useful here, not flexible enough, etc.
- Development in accordance with a safety standard
  - …. Fully applying ISO 26262-6 or other standards not feasible, open source development
- Evaluation of the tool development process
  - Yes, one argument, community guidelines etc. can be evaluated, but alone not enough
- Validation of the software tool
  - Only one left ;)

Safety Lifecycle with Open Source – Sphinx

- Two Part Approach based on the ISO 26262 / Possible IEC 61508 3rd Edition
- “Tool Classification” as Open Source
  - Corresponds to the ISO 26262 Evaluation Phase
  - Planning of tool usage, use cases and analysis
  - “Software Tool Criteria Evaluation Report” without mitigations
  - https://github.com/useblocks/sphinx-safety
- “Tool Qualification” as Closed Source
  - Linking the faults identified in the analysis to test/restrictions/etc.
  - Linking of tests to the features
  - Analysis of verification results. Including gaps in code coverage
  - Creation of the safety manual summarizing the usage restrictions etc.
  - Resulting in Software Tool Qualification Report

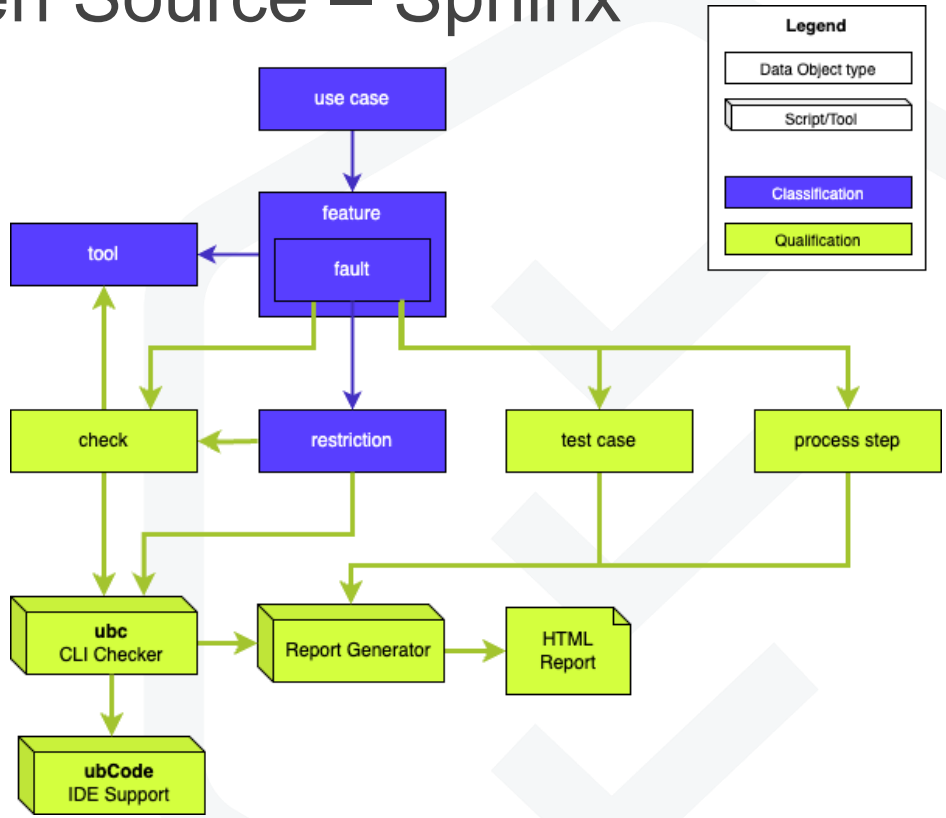
Safety Lifecycle with Open Source – Sphinx

- Defining the scope of the efforts
  - Which tools?
    Sphinx, Sphinx-Needs, Sphinx-testreports, sphinx-codelinks

- Defining the use cases
  - Writing Requirements
  - Document SW APIs / Unit Design
  - Integrate Test Results
  - Generate documentation packages

Safety Lifecycle with Open Source – Sphinx

- Based on the Use Cases, identify the features of the different tools needed
- Analyze these features through a HAZOP including Tool Impact
- Determining the faults for the features
- Up to this step, all included in the open source repo



Safety Lifecycle with Open Source – Sphinx

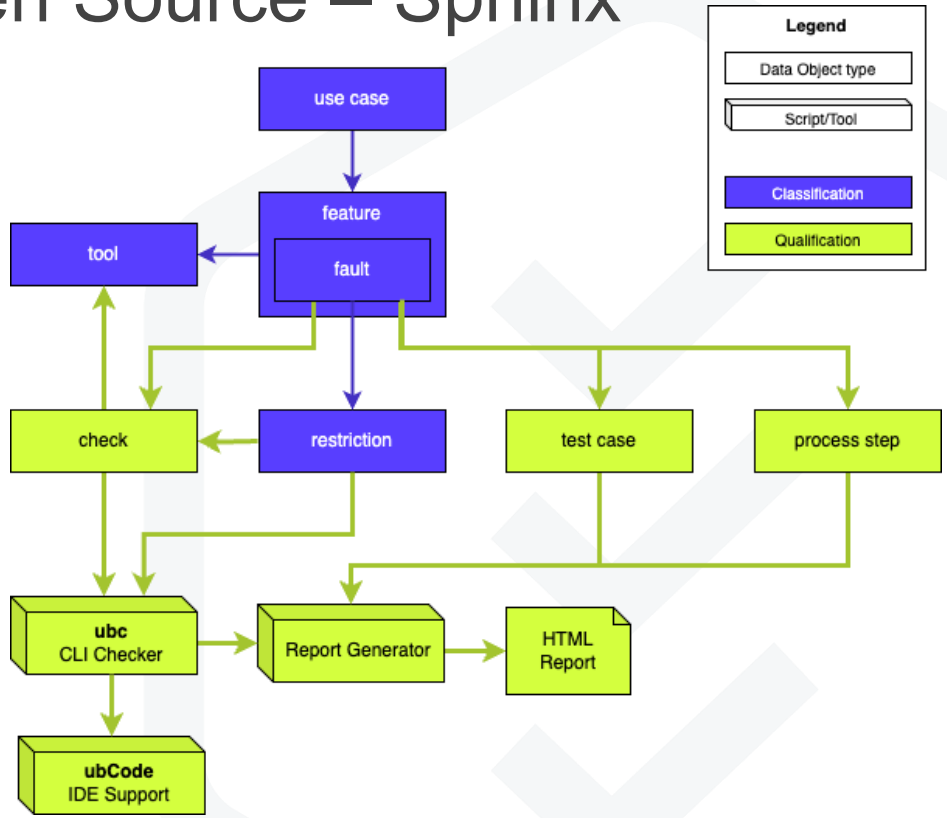
- Now linking the faults to either of four countermeasures

- Restrictions

- Test cases
- Process Steps

Or on higher level

- Redundancy in tools



Safety Lifecycle with Open Source – Sphinx

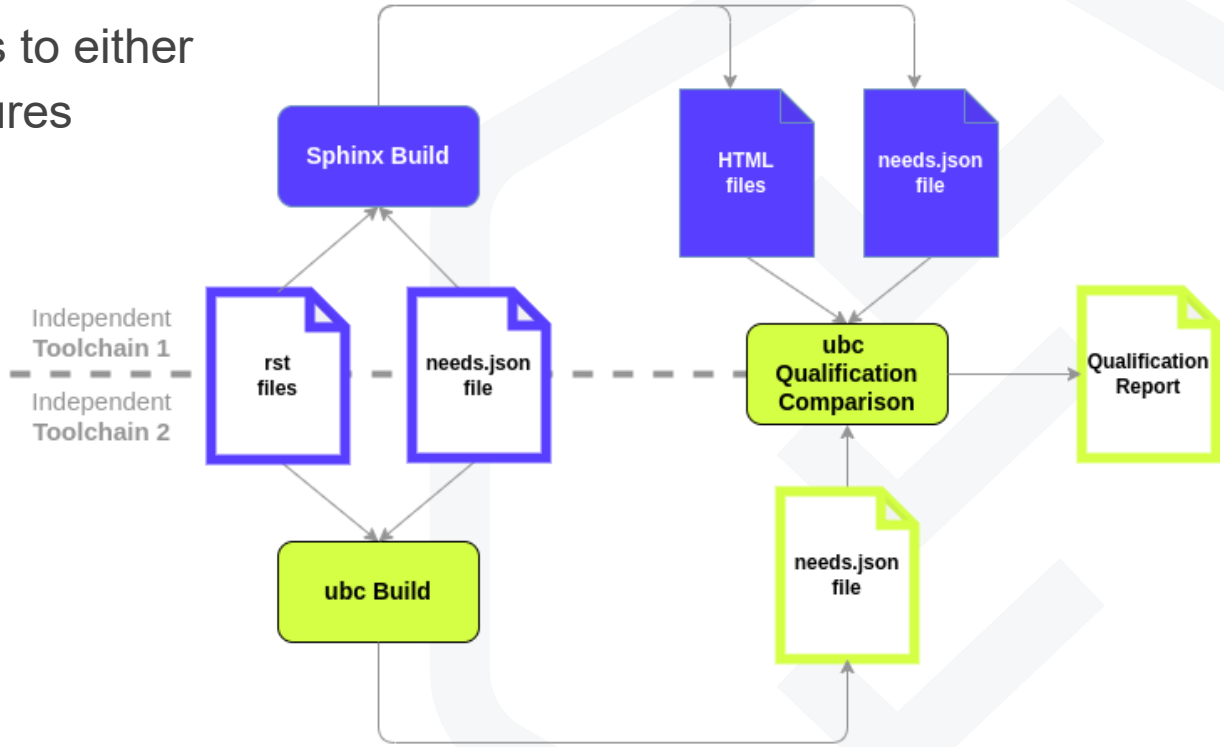
- Now linking the faults to either of four countermeasures

- Restrictions

- Test cases
- Process Steps

Or on higher level

- Redundancy in tools



Safety Lifecycle with Open Source – Sphinx

- Validation of the software tool
- Linking and reviewing the existing test cases to the features in the analysis
- Identifying gaps, missing tests? Enough tests? Not covered code? Adding new tests, will be part of the open source part
- The achieved code coverage during testing is analyzed, which is not required for tool qualification
- Target is 100% explained statement code coverage (with justifications of remaining gaps), same as for ASIL A/B Embedded Code

- This approach etc. (as it is part of the open source) can be used by other projects as a template

Safety Lifecycle with Open Source – Compiler

- “Tool integration and validation in usage context” “under their circumstances of use”
- If all stages of dynamic testing are executed on-target or at least in simulator with target architecture and include code coverage
- Pretty strong argument, that the whole output has been validated in usage context

Thank you!