



# StrictDoc project

- Project overview
- Case studies
- Motivation
- Implementation details
- Development plan



# StrictDoc — Development team

## StrictDoc core team:

Stanislav Pankevich, [s.pankevich@gmail.com](mailto:s.pankevich@gmail.com)

Maryna Balioura, [mettta@gmail.com](mailto:mettta@gmail.com)

## Documentation:

[StrictDoc project](#)

[StrictDoc roadmap](#)

## Get in touch with the community:

[StrictDoc GitHub issues](#)

[StrictDoc mailing list](#)

[StrictDoc office hours](#) — Every Tuesday, 17:00–18:00 CET



# Contents

- 1) StrictDoc — Project overview and screenshot tour.
- 2) StrictDoc — Case studies.
- 3) StrictDoc — Motivation behind the project.
- 4) StrictDoc — Implementation details.
- 5) StrictDoc — Development plan, roadmap, backlog.

---

**StrictDoc —**

**Project overview and screenshot tour**



# What is StrictDoc?

- StrictDoc is a documentation generator, editor, and converter.
- Supports requirements traceability and structured documentation workflows.
- Designed with safety-critical systems in mind.
- Open source software under the Apache 2.0 license.



## StrictDoc — Project goals

- Long-term vision: a free and open-source, but highly capable, tool that makes requirements work easy.
- Automate documentation and requirements work at all levels.
- 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.
- All target groups are considered:
  - Software, hardware
  - Systems, electrical, thermal, etc.
  - QA, Safety, management, non-technical, etc.



# StrictDoc — Open source tool

- Spare-time project for 2 core developers. 22 contributors so far.
- Created in 2019, inspired by the Doorstop project.
- Inspired by Doorstop's OSS approach to Git-based requirements management.
- Written in Python.
- Apache 2 license.
- 1.8K pull requests, 5K+ commits, 30K+ LOC.

## Key highlights:

- **2020-2022:** Documentation generator, HTML export, ReqIF, tracing source files to requirements, document grammar and custom fields, traceability graph validations.
- **2023:** Web-based user interface. The HTML-to-PDF feature for publishing documents.
- **2024:** Language-aware traceability to C/C++ and Python.
- **2025:** Extending traceability to test and coverage reports, preparing for safety-related qualification.



## How StrictDoc supports critical software development

- Create and manage technical documentation with requirements
- Traceability matrix for all artifacts
- Tracing requirements to source files, test results, test coverage
- Project statistics report
- Search query engine
- Diff and changelog
- Publishing standalone HTML and PDF documents
- ReqIF support for requirements exchange

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



## .SDoc format

- Starting point: Format to support text and metadata.
- YAML frontmatter does not scale to large documents.
- RST directives do not support nested metadata.
- JSON is less human-readable, and so are HTML/XML.
- Nesting content in a document with 4+ 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/closing tags for nested content
- ASN.1 – Capital letters.

```

drafts > requirements > 01_strictdoc > L1_Open_Requirements_Tool.sdoc
194
195 [REQUIREMENT]
196 UID: SDOC-SSS-3
197 TITLE: Documents (CRUD)
198 STATEMENT: >>>
199 The Requirements Tool shall provide the CRUD
    operations for document management:
200
201 - Create document
202 - Read document
203 - Update document
204 - Delete document.
205 <<<
206 RATIONALE: >>>
207 The CRUD operations are essential operations of
    document management. They are at the core of a
    documentation management tool.
208 <<<
209
210 [REQUIREMENT]
211 UID: SDOC-SSS-51
212 TITLE: Documents with nested sections/chapters
    structure
213 STATEMENT: >>>
214 The Requirements Tool shall allow management of
    documents with nested sections/chapters structure.
215 <<<
216


```

## Two ways to link code and requirements

- 1 **Relation markers:** Link code to requirements.  
Use when can control **source code**  directly.

```
class ProjectStatisticsGenerator:
    def export(
        self,
        project_config: ProjectConfig,
        traceability_index: TraceabilityIndex,
    ) -> Markup:
        """
        Export project statistics to an HTML page.

        @relation(SDOC-SRS-97, scope=function)
        """
```

- 2 **Forward relations:** Link requirements to source code.  
Use when **source code**  cannot be modified.

```
[REQUIREMENT]
UID: SDOC-SRS-97
TITLE: Project statistics generator
STATEMENT: StrictDoc shall generate project statistics.
RELATIONS:
- TYPE: File
  VALUE: strictdoc/generators/project_statistics.py
  FUNCTION: ProjectStatisticsGenerator.export
```



## Attach metadata to source code

- The comments are parsed as SDoc nodes.
- An auto-generated document is created from a provided template.

### Use case example:

- Generate a test specification from source code annotations.
- The test spec items are linked to requirements automatically.

```
/**
 * \brief Test example
 *
 * @relation(REQ-1, scope=function)
 *
 * INTENTION: ...
 *
 * INPUT: ...
 *
 * EXPECTED_RESULTS: ...
 */
TEST_CASE("Test example", testExample)
{
    ...
}
```

## StrictDoc — Two workflows

1

### Command-line / IDE workflow

- Generate documents using ``strictdoc`` command.
- The default export is static HTML.

```
strictdoc export .
```

2

### Web interface

- Work with editable documentation.
- Starts a web server.  
Static HTML that is editable.

```
strictdoc server .
```

/docs/

## User Guide

strictdoc\_01\_user\_guide.sdoc

## StrictDoc Feature Map

strictdoc\_02\_feature\_map.sdoc

## F.A.Q.

strictdoc\_03\_faq.sdoc

## Release Notes

strictdoc\_04\_release\_notes.sdoc

## Troubleshooting

strictdoc\_05\_troubleshooting.sdoc

## Contributing to StrictDoc

strictdoc\_10\_contributing.sdoc

## Developer Guide

strictdoc\_11\_developer\_guide.sdoc

## Requirements Tool Specification (L1)

strictdoc\_20\_L1\_Open\_Requirements\_Tool.s...

## StrictDoc Requirements Specification (L2)

strictdoc\_21\_L2\_StrictDoc\_Requirements.sd...

## Development Plan

strictdoc\_24\_development\_plan.sdoc

## Design Document

strictdoc\_25\_design.sdoc

## StrictDoc Backlog

strictdoc\_28\_Backlog.sdoc

## Credits

strictdoc\_30\_credits.sdoc

/docs\_extra/

## Technical Note: DO-178C requirements tool requirements

DO178\_requirements.sdoc

## Technical Note: Zephyr requirements tool requirements

Zephyr\_requirements.sdoc

## 1.5. Document model

MID:	f94419a390f2403daa3c4cd0e96f6cb6
UID:	SDOC-SRS-98
STATUS:	Active
STATEMENT:	StrictDoc's data model shall support modeling documents.
PARENT RELATIONS:	← <b>SDOC-SSS-64</b> Structuring requirements in documents
FILE RELATIONS:	</> strictdoc/backend/sdoc/models/document.py, lines: 1-277, entire file

## 1.6. Document metadata

MID:	435f856197d743c1b9b1dcbe91a22863
UID:	SDOC-SRS-110
STATUS:	Active
STATEMENT:	StrictDoc's data model shall support a Document metadata model including at least: <ul style="list-style-type: none"> <li>UID</li> <li>Document version</li> <li>Document classification</li> <li>Document authors.</li> </ul>
PARENT RELATIONS:	← <b>SDOC-SSS-53</b> Document meta information (UID, version, authors, signatures, etc) ← <b>SDOC-SSS-75</b> Document versioning
FILE RELATIONS:	</> strictdoc/backend/sdoc/models/document_config.py, lines: 1-191, entire file

- 1 SDoc data model
  - 1.1 Data model
  - 1.2 Requirement model
  - 1.3 Requirement model fields
  - 1.4 Requirement model default fields
  - 1.5 Document model
  - 1.6 Document metadata
  - 1.7 Document configuration
  - 1.8 Section model
  - 1.9 Free text
  - 1.10 Composeable document
  - 1.11 Requirement relations
  - 1.12 Requirement relation roles
  - 1.13 Inline links
  - 1.14 Inline anchors

- 2 SDoc text markup
  - 2.1 SDoc markup language
  - 2.2 Identical SDoc content by import/export roundtrip
  - 2.3 SDoc and Git storage
  - 2.4 SDoc file extension
  - 2.5 One document per one SDoc file
  - 2.6 Fixed grammar
  - 2.7 Default grammar fields
  - 2.8 Custom grammar / fields
  - 2.9 Importable grammars

## Nodes Ranges

**SDOC-SRS-109**  
Composeable document

[ 1-277 ] strictdoc/backend/sdoc/models/document.py,  
entire file

[ 1-102 ]  
strictdoc/backend/sdoc/models/document\_from\_file.py,  
entire file  
[ 749-802 ] strictdoc/core/traceability\_index\_builder.py,  
range

**SDOC-SRS-98**  
Document model

[ 1-277 ] strictdoc/backend/sdoc/models/document.py,  
entire file

Path:	strictdoc/backend/sdoc/models/document.py
Lines:	277
Non-empty lines:	225
Non-empty lines covered with requirements:	225 / 225 (100.0%)
Functions:	31
Functions covered by requirements:	31 / 31 (100.0%)

1 - 277 | entire file

 ⚡ "Document model" (REQUIREMENT)  
⚡ "Composeable document" (REQUIREMENT)

```
1  """
2  @relation(SDOC-SRS-98, SDOC-SRS-109, scope=file)
3  """
4
5  from typing import Generator, List, Optional
6
7  from strictdoc.backend.sdoc.document_reference import DocumentReference
8  from strictdoc.backend.sdoc.models.document_config import DocumentConfig
9  from strictdoc.backend.sdoc.models.document_grammar import (
10     DocumentGrammar,
11 )
12 from strictdoc.backend.sdoc.models.document_view import DocumentView
13 from strictdoc.backend.sdoc.models.grammar_element import (
14     GrammarElement,
15     GrammarElementField,
```

# SDoc format and grammar

```
strictdoc_02_feature_map.sdoc x
```

```

56
57 [FEATURE]
58   UID: SDOC-FEAT-1
59   STATUS: Stable
60   TITLE: SDoc text markup
61   STATEMENT: >>>
62   The SDoc markup language is a hybrid format inspired by TOML, YAML, ASN.1, and HTML/XML,
    designed specifically for structuring technical documents with large volumes of
    requirements. It aims to encode documents that span up to several hundred or even a few
    thousand A4-printed pages, while keeping the markup noise minimal to maintain readability.
    The format supports both shallow and deeply nested document structures, accommodating up to
    9-10 levels of chapter nesting, and allows for multiple meta-information fields around
    each requirement.
63   <<<
64   DOCUMENTATION: >>>
65   - [LINK: SECTION-UG-SDoc-syntax]
66   <<<
67
68 [SECTION]
69   TITLE: Use case
70
71 [TEXT]
72   STATEMENT: >>>
73   The main use case for SDoc is to model a structure of a technical document that consists of
    tens and hundreds of technical requirements. The following high-level requirements for the
    markup are therefore relevant:

```

```
strictdoc_02_feature_map.sdoc x
```

```

1  [DOCUMENT]
2  TITLE: Feature Map
3  UID: SDOC_FEATURE_MAP
4  REQ_PREFIX: SDOC-FEAT-
5  OPTIONS:
6    REQUIREMENT_STYLE: Inline
7    REQUIREMENT_IN_TOC: True
8
9  [GRAMMAR]
10 ELEMENTS:
11 - TAG: TEXT
12   FIELDS:
13   - TITLE: UID
14     TYPE: String
15     REQUIRED: False
16   - TITLE: STATEMENT
17     TYPE: String
18     REQUIRED: False
19 - TAG: FEATURE
20   FIELDS:
21   - TITLE: MID
22     TYPE: String
23     REQUIRED: False
24   - TITLE: UID
25     TYPE: String
26     REQUIRED: False
27   - TITLE: STATUS
28     TYPE: String
29     REQUIRED: False
30   - TITLE: TITLE

```

# StrictDoc — Project tree

The screenshot displays the StrictDoc Project Index interface. On the left, a sidebar contains navigation icons. The main area is titled "StrictDoc Documentation / PROJECT INDEX". It features a "strictdoc/" folder containing a "docs/" subfolder. The "docs/" folder lists several documents: "User Guide" (strictdoc\_01\_user\_guide.sdoc), "StrictDoc Feature Map" (strictdoc\_02\_feature\_map.sdoc), "F.A.Q." (strictdoc\_03\_fa.q.sdoc), "Release Notes" (strictdoc\_04\_release\_notes.sdoc), "Troubleshooting" (strictdoc\_05\_troubleshooting.sdoc), "Contributing to StrictDoc" (strictdoc\_10\_contributing.sdoc), "Developer Guide" (strictdoc\_11\_developer\_guide.sdoc), "Requirements Tool Specification (L1)" (strictdoc\_20\_L1\_Open\_Requirements\_Tool.sdoc), and "StrictDoc Requirements Specification (L2)". On the right, a "Project tree configuration" panel shows the "Input paths" as "/Users/Stanislaw/workspace/projects/strictdoc-project/strictdoc" and "Document paths" as "docs/\*\*", "docs\_extra/\*\*", "docs/sphinx/\*\*", and "tests/\*\*". The "Source root path" is "/Users/Stanislaw/workspace/projects/strictdoc-project/strictdoc". The "Source paths" include "pyproject.toml", "tasks.py", "strictdoc/\*\*/\*.js", "strictdoc/\*\*/\*.py", "strictdoc/\*\*/\*.jinj.rst", "tests/integration/\*\*/\*.test", "tests/unit/\*\*/\*.py", "\*\*\_init\_.py", "build/\*\*", "output/\*\*", "strictdoc-project.github.io/\*\*", and "tests/unit/strictdoc/backend/sdoc\_source\_code\*\*". The bottom right corner indicates "Built with StrictDoc 0.2.0".

StrictDoc Documentation / PROJECT INDEX

strictdoc/

- docs/
  - User Guide  
strictdoc\_01\_user\_guide.sdoc
  - StrictDoc Feature Map  
strictdoc\_02\_feature\_map.sdoc
  - F.A.Q.  
strictdoc\_03\_fa.q.sdoc
  - Release Notes  
strictdoc\_04\_release\_notes.sdoc
  - Troubleshooting  
strictdoc\_05\_troubleshooting.sdoc
  - Contributing to StrictDoc  
strictdoc\_10\_contributing.sdoc
  - Developer Guide  
strictdoc\_11\_developer\_guide.sdoc
  - Requirements Tool Specification (L1)  
strictdoc\_20\_L1\_Open\_Requirements\_Tool.sdoc
  - StrictDoc Requirements Specification (L2)

Project tree configuration

Input paths: /Users/Stanislaw/workspace/projects/strictdoc-project/strictdoc

Document paths:

- ✓ docs/\*\*
- ✓ docs\_extra/\*\*
- ✗ docs/sphinx/\*\*
- ✗ tests/\*\*

Source root path: /Users/Stanislaw/workspace/projects/strictdoc-project/strictdoc

Source paths:

- ✓ pyproject.toml
- ✓ tasks.py
- ✓ strictdoc/\*\*/\*.js
- ✓ strictdoc/\*\*/\*.py
- ✓ strictdoc/\*\*/\*.jinj.rst
- ✓ tests/integration/\*\*/\*.test
- ✓ tests/unit/\*\*/\*.py
- ✗ \*\*\_init\_.py
- ✗ build/\*\*
- ✗ output/\*\*
- ✗ strictdoc-project.github.io/\*\*
- ✗ tests/unit/strictdoc/backend/sdoc\_source\_code\*\*

Built with StrictDoc 0.2.0



# StrictDoc — Web interface

The screenshot displays the StrictDoc web interface. At the top, the breadcrumb navigation shows 'StrictDoc Documentation / User Guide / Document'. To the right of the breadcrumb are two buttons: 'Edit grammar' and 'Export to PDF'. On the left side, there is a sidebar with a list of documents under the '/docs' directory. The main content area is titled 'User Guide' and shows a document with a UID of 'SDOC\_UG'. The document content includes a section titled '1. Introduction' which describes StrictDoc as software for technical documentation and requirements management, and provides a summary of its features. On the right side, there is a table of contents for the document, listing sections from '1 Introduction' to '4 Running StrictDoc'.

StrictDoc Documentation / User Guide / Document

Edit grammar Export to PDF

/docs

- User Guide  
strictdoc\_01\_user\_guide.sdoc
- Feature Map  
strictdoc\_02\_feature\_map.sdoc
- F.A.Q.  
strictdoc\_03\_faq.sdoc
- Release Notes  
strictdoc\_04\_release\_notes.sdoc
- Troubleshooting  
strictdoc\_05\_troubleshooting.sd...
- Contributing to StrictDoc  
strictdoc\_10\_contributing.sdoc
- Developer Guide  
strictdoc\_11\_developer\_guide.sd...
- Requirements Tool Specification (L1)  
strictdoc\_20\_L1\_Open\_Requireme...
- StrictDoc Requirements Specification (L2)  
strictdoc\_21\_L2\_StrictDoc\_Requir...
- Development Plan  
strictdoc\_24\_development\_plan.s...
- Design Document

## User Guide

UID: SDOC\_UG

### 1. Introduction

StrictDoc is software for technical documentation and requirements management.

Summary of StrictDoc features:

- The documentation files are stored as human-readable text files.
- A simple domain-specific language DSL is used for writing the documents. The text format for encoding this language is called SDOC (strict-doc).
- StrictDoc reads \*.sdoc files and builds an in-memory representation of a document tree.
- From this in-memory representation, StrictDoc can generate the

- 1 Introduction
  - 1.1 Contact the developers
- 2 Examples
  - 2.1 Hello World
  - 2.2 StrictDoc Examples repository
  - 2.3 StrictDoc Templates repository
  - 2.4 Other examples
- 3 Installing StrictDoc
  - 3.1 Requirements
  - 3.2 Installing StrictDoc as a Pip package (recommended way)
  - 3.3 Installing "nightly" StrictDoc as a Pip package
  - 3.4 Installing StrictDoc into a Docker container
  - 3.5 Installing StrictDoc as a Snap package (not maintained)
- 4 Running StrictDoc
  - 4.1 Static HTML export
  - 4.2 Web server

Built with StrictDoc 0.2.0

# StrictDoc — Web interface / Edit node

The screenshot displays the StrictDoc web interface for editing a requirement node. The interface is divided into three main sections: a left sidebar, a central editor, and a right sidebar.

**Left Sidebar:** A list of documents under the path `/docs`. The documents include:

- User Guide (`strictdoc_01_user_guide.sdoc`)
- StrictDoc Feature Map (`strictdoc_02_feature_map.sdoc`)
- F.A.Q. (`strictdoc_03_faq.sdoc`)
- Release Notes (`strictdoc_04_release_notes.sdoc`)
- Troubleshooting (`strictdoc_05_troubleshooting.sd...`)
- Contributing to StrictDoc (`strictdoc_10_contributing.sdoc`)
- Developer Guide (`strictdoc_11_developer_guide.sd...`)
- Requirements Tool Specification (L1) (`strictdoc_20_L1_Open_Requireme...`)
- StrictDoc Requirements Specification (L2)** (`strictdoc_21_L2_StrictDoc_Reqir...`)
- Development Plan (`strictdoc_24_development_plan.s...`)
- Design Document

**Central Editor:** A form for editing a requirement node. It has three tabs: **Fields**, **Relations**, and **Comments**. The **Fields** tab is active, showing the following fields:

- MID:** 8330d61fd5b1438fa90f127f88903a0d
- UID:** SDOC-SRS-30
- STATUS:** Active
- TITLE:** Detect links cycles
- STATEMENT:** The Traceability Index shall detect cycles between requirements.
- RATIONALE:** Enter RATIONALE here...

At the bottom of the form are **Save** and **Cancel** buttons. A **REQUIREMENT** label is visible at the bottom right of the editor area.

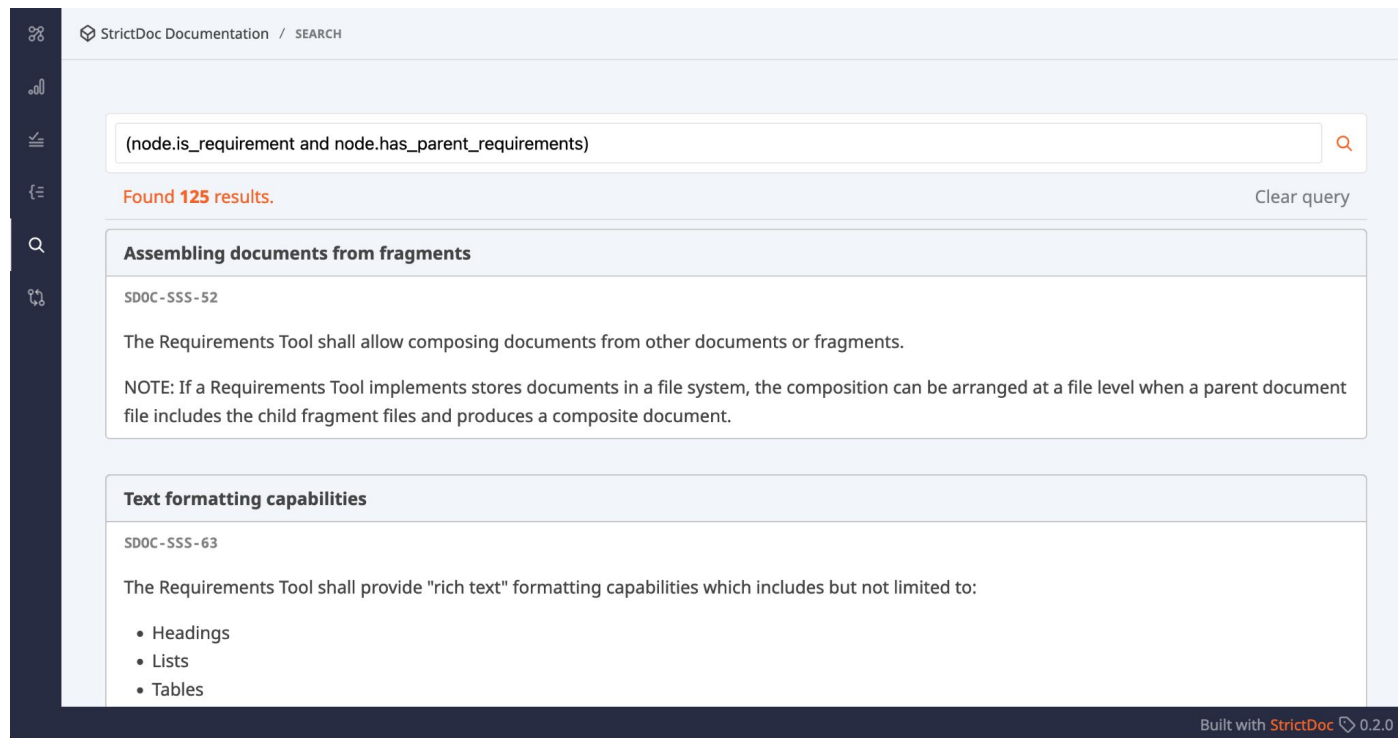
**Right Sidebar:** A list of nodes in the document hierarchy:

- 1 SDoc data model
  - 1.1 Data model
  - 1.2 Requirement model
  - 1.3 Requirement model fields
  - 1.4 Requirement model default fields
  - 1.5 Document model
  - 1.6 Document metadata
  - 1.7 Section model
  - 1.8 Free text
  - 1.9 Composeable document
  - 1.10 Requirement relations
  - 1.11 Requirement relation roles
- 2 SDoc text markup
  - 2.1 SDoc markup language
  - 2.2 Identical SDoc content by import/export roundtrip
  - 2.3 SDoc and Git storage
  - 2.4 SDoc file extension

**Top Bar:** The breadcrumb path is `StrictDoc Documentation / StrictDoc Requirements Specification (L2) / Document`. On the right, there are buttons for **Edit grammar** and **Export to PDF**.

**Bottom Bar:** The text "Built with StrictDoc 0.2.0" is displayed.

# StrictDoc — Search screen and query engine



The screenshot displays the StrictDoc search interface. At the top, a dark sidebar contains navigation icons. The main header shows 'StrictDoc Documentation / SEARCH'. A search bar contains the query '(node.is\_requirement and node.has\_parent\_requirements)' with a magnifying glass icon to its right. Below the search bar, it states 'Found 125 results.' and provides a 'Clear query' link. The results are organized into two sections: 'Assembling documents from fragments' and 'Text formatting capabilities'. The first section shows a result for 'SDOC-SSS-52' with a paragraph of text and a note. The second section shows a result for 'SDOC-SSS-63' with a paragraph of text and a bulleted list.

StrictDoc Documentation / SEARCH

(node.is\_requirement and node.has\_parent\_requirements) 🔍

Found 125 results. [Clear query](#)

### Assembling documents from fragments

SDOC-SSS-52

The Requirements Tool shall allow composing documents from other documents or fragments.

NOTE: If a Requirements Tool implements stores documents in a file system, the composition can be arranged at a file level when a parent document file includes the child fragment files and produces a composite document.

### Text formatting capabilities

SDOC-SSS-63

The Requirements Tool shall provide "rich text" formatting capabilities which includes but not limited to:

- Headings
- Lists
- Tables

Built with StrictDoc 0.2.0

# StrictDoc — Traceability metrics

- Two interconnected features: Statistics and Search.
- Inspired by the ECSS SW metrication handbook
- Writing project-specific statistics generators in Python.
- Calculate total numbers of entities
- Calculate the number of present and missing relations
- Each statistic can be queried individually.
- Inspired by the ECSS SW metrication HB:
  - Requirements coverage
  - Calculate a number of TBD/TBC.

StrictDoc Documentation / SEARCH

node.is\_source\_file\_with\_complete\_coverage() 🔍

Found 175 results. Clear query

pyproject.toml

Lines: 134

Connected requirements: 1

Coverage: 100.0%

tasks.py

# StrictDoc — Diff screen

StrictDoc Documentation / DIFF

HEAD HEAD+

Diff Changelog

**StrictDoc Requirements Specification (L2)** -

+ [R] 1.6 Document metadata

- [R] 1.7 Section model

[MID] 63821c507d584cf985f05904710b9779

[UID] SD0C-SRS-99

[STATUS] Active

[TITLE] Section model

[STATEMENT] StrictDoc's data model shall support a concept of a "Section" which nests other Sections, Requirements, Texts.

[RATIONALE] "Section" corresponds to a chapter or a section in a document and helps to organize a document by grouping text nodes, requirements and other sections.

[RELATION] SD0C-SSS-51 Documents with nested sections/chapters structure

**StrictDoc Requirements Specification (L2)** -

+ [R] 1.6 Document metadata

- [R] 1.7 [Modified] Section model

[MID] 63821c507d584cf985f05904710b9779

[UID] SD0C-SRS-99

[STATUS] Active

[TITLE] [Modified] Section model

[STATEMENT] [Modified] StrictDoc's data model shall support a concept of a "Section" which nests other Sections, Requirements, Texts.

[RATIONALE] [Modified] "Section" corresponds to a chapter or a section in a document and helps to organize a document by grouping text nodes, requirements and other sections.

[RELATION] SD0C-SSS-51 Documents with nested sections/chapters structure

Built with StrictDoc 0.2.0

# StrictDoc — Changelog screen

🔍

📄

📁

🔍

🔍

🔍

🔍

StrictDoc Documentation / DIFF

HEAD^

HEAD

Diff

Changelog

COMPARED REVISIONS

Left	HEAD^
Right	HEAD

SUMMARY OF THE CHANGES

Nodes modified	64
Documents modified	1
Sections modified	36 (36 added)
Requirements modified	27 (3 modified, 24 added)

# 1 Requirement modified

MID

b3561dd4f8e64f11a251c3471022aece

STATEMENT

Shortly: The SDoc markup is a hybrid of TOML and YAML with some influence from HTML/XML and ASN.1 <https://en.wikipedia.org/wiki/ASN.1>\_. Using each of these formats as-is, and also the JSON format, was considered but discarded during the design. The SDoc markup has been pretty stable since its inception but the flexibility of the TextX parser allows

MID

b3561dd4f8e64f11a251c3471022aece

STATEMENT

See [LINK: SECTION-FM-SDoc-text-markup] for a description of the SDoc feature.

Built with StrictDoc 0.2.0

---

# StrictDoc — Case studies



# Who is using StrictDoc?

- StrictDoc is used by multiple companies, mainly those needing to certify software to industry standards.
- The represented industries:
  - Medical
  - Aerospace
  - Industrial
  - Open source software projects
- Examples:
  - Zephyr RTOS
  - Linux-based OS for industrial use
  - [Security Requirements for Vehicle Security Gateways 2024-01-2806](#)



## Use case: The Zephyr project



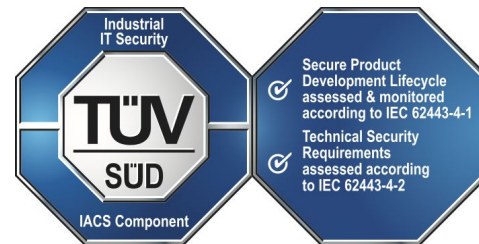
- The Zephyr Safety Working Group is working to make the Zephyr RTOS certifiable for use in safety-related projects:
  - [Zephyr Safety Overview](#)
  - [Safety FAQ](#)
- StrictDoc is used in two ways:
  - Technical requirements are captured in SDoc files in a dedicated Git repository: <https://github.com/zephyrproject-rtos/reqmgmt>.
  - StrictDoc is also used for the Zephyr safety-related IEC 61508 compliance documentation.
- The StrictDoc HTML export has been approved by the safety auditors as an acceptable tool for requirements review.

See also the FOSDEM 2024 talk: [Application of the SPDX Safety Profile in the Safety Scope of the Zephyr Project](#).

## Use case:

# Operating system development at Linutronix

- Technical documentation for **IGLOS**, a secure Linux-based OS for industrial use, started in 2024 from scratch using StrictDoc (no commercial RE tools).
- Structure based on arc42, extended with requirements, compliance matrix, threat model, and user guide.
- Edited via Web UI or text editors, reviewed in GitLab MRs. HTML export deployed to an internal web server. A diff-UI supports requirement reviews.
- Requirements trace to Robot/pytest tests and GitLab reviews.  
Code accessed via git submodules. Minor scripting for custom import/export.  
External PDFs interfaced by converting ToC to \*.sdoc.
- Certification according to IEC 62443-4-2 accomplished, EU CRA upcoming.  
Gap analysis by sending HTML/PDF exports to certification body.  
Audit focused on StrictDoc "Compliance Matrix" document including conformity statements.



---

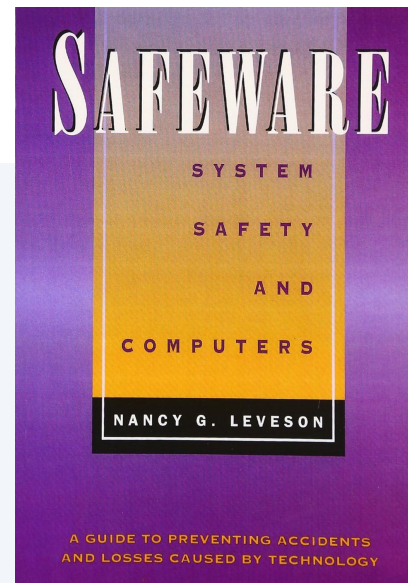
**StrictDoc —**

**The motivation behind the project**

## Motivation for this work: Bridging requirements and software

“The vast majority of accidents in which software was involved can be traced to requirements flaws and, more specifically, to incompleteness in the specified and implemented software behavior — that is, incomplete or wrong assumptions about the operation of the controlled system or required operation of the computer and unhandled controlled-system states and environmental conditions. Although coding errors often get the most attention, they have more of an effect on reliability and other qualities than on safety.”

— *Nancy Leveson, Safeware: System Safety and Computers (1995)*



- How can developers work more effectively with requirements during software development?
- How to connect code to requirements?
- Which tools are needed to support this connection?



## More questions than answers

- Why is open source software (OSS) typically developed without formal requirements?
  - Why?
  - Lack of need? Lack of skill? Lack of time? Lack of tools? Lack of process?
- Can open source projects benefit from implementing formal requirements?
- "The best requirements are code itself. I don't need requirements."
  - How to answer this?
- How to make requirements useful for open source software?
- How to argue about requirements? What makes a good requirements engineer?
- Requirements are simply captured decisions or there is more to it?
- Are we missing a tool / approach to handle requirements in a practical way?



# Issues in Requirements Engineering

- Commercial requirements tools can be (very) expensive
  - A previously affordable commercial solution is becoming increasingly expensive over time.
- Exchanging requirements
  - How to build a working group with several organizations collaborating?
  - 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

## Requirements **as a tool**

- Requirements capture project decisions.
- Requirements are the foundation of traceability.
  - Each requirement is uniquely identified.
  - Requirements can be linked to each other for formal tracking.
- Requirements can be used to manage work.
  - Requirements drive project organization and work breakdown.
  - Requirements define contracts and acceptance criteria.
- Requirements can be used for measurements.
- Requirements are powerful abstractions.

**SYS-222** The spacecraft shall be designed and verified to achieve a minimum reliability of 99.7% over the mission duration, as defined from launch through end-of-life, including all nominal and contingency operations.

**MIS-111 ← SYS-222**

**60%** of requirements are already implemented by a contractor.

**87%** of requirements are verified by test.

Requirements define the 'public interface' of the work performed; the source code is the implementation.



# Requirement = (Statement + meta information)

Common fields:

TITLE, STATEMENT, RATIONALE, COMMENT

Unique identifiers:

MID / UID (machine- vs human-readable)

Additional meta information:

STATUS, TAGS, LEVEL, PREFIX

Industry-specific meta information:

e.g., ASIL(A,B,C,D), VERIFICATION (R,A,I,T)

Relations/roles:

e.g., parent, child, parent/refines, etc.

## Example of a real-world requirement (adopted from ECSS-E-ST-40C):

**UID:** 5.4.2.3a

**TITLE:** Construction of a software logical model

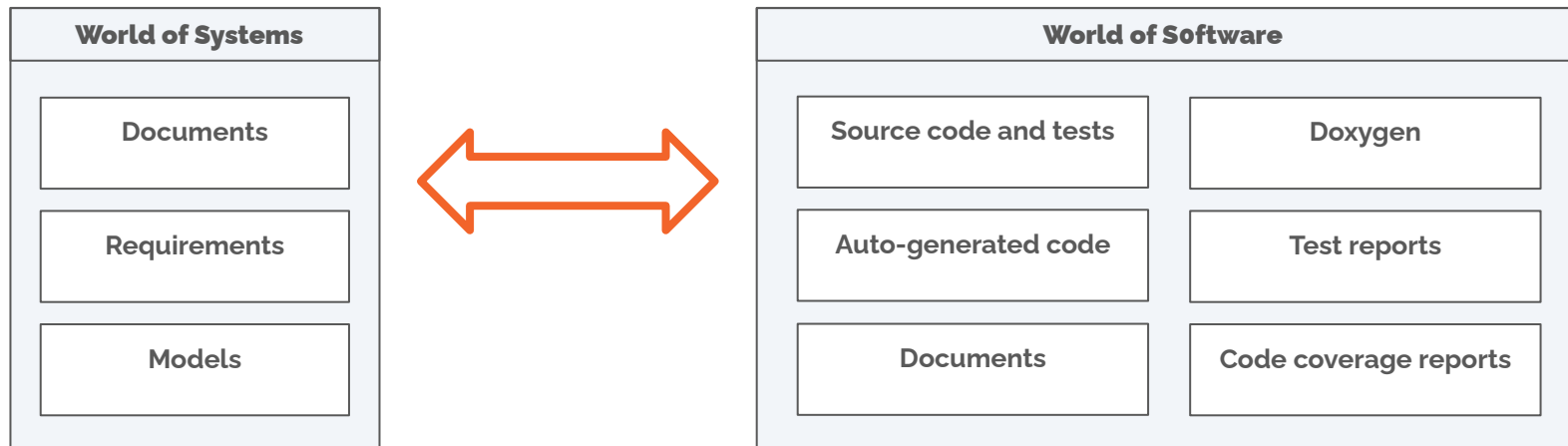
**STATEMENT:** The supplier shall construct a logical model of the functional requirements of the software product.

**VERIFICATION:** Review, Inspection.



# What is Traceability?

In software development, traceability refers to the ability to track and link artifacts across the software development lifecycle. This typically means creating explicit connections between requirements, design, code, tests, and documentation.



# Why does **Traceability** matter?

- **Connect intent with implementation**
  - What was requested is what was implemented.
- **Coverage information**
  - Reduce the chance of missing requirements or building incorrect functionality.
  - Ensure every requirement is tested and verified (forward traceability).
  - Ensure every feature or test is tied to an actual need (backward traceability).
- **Traceable development process and project memory**
  - Prove that contractual or legal obligations are fulfilled.
  - Onboard new engineers and preserve rationale over time.
  - Prevent duplication, scope creep, and orphaned work.
- **Impact analysis**
  - If a requirement changes, what else has to be changed?





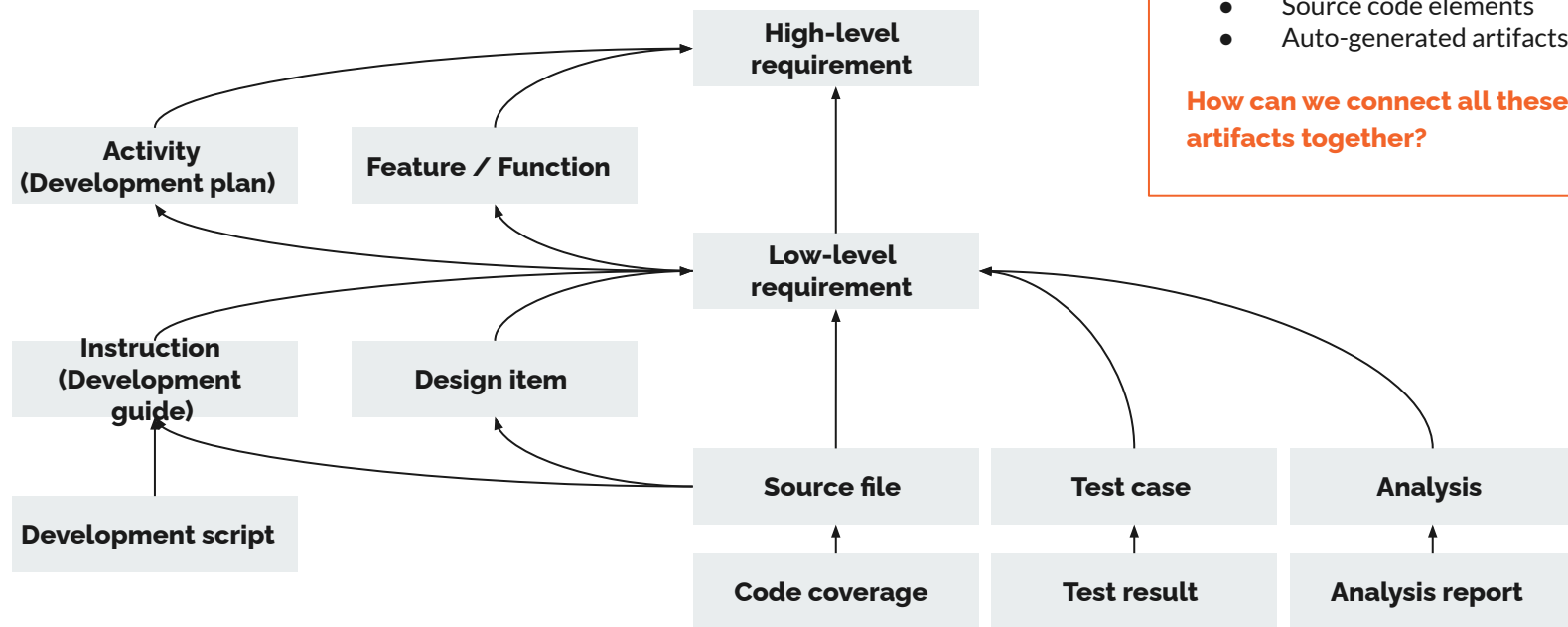
# Functional breakdown structure

- For each spec level, a functional analysis results in function partitioning.
- Discovered functions become separate documents or spec chapters.
- Requirements that are functionally related are easier to implement, maintain, verify.
- Functional, performance, interface requirements allocated to functions.
- General cross-cutting requirements (security, safety, quality, etc) may apply to all requirements within a functional group.
- A suboptimal functional breakdown structure is harder to maintain. Choose the functions wisely.

## Functional Partitioning

Functional partitioning is the process of grouping functions that logically fit with the components likely to be used, and to minimize functional interfaces. Partitioning is performed as part of functional decomposition. It identifies logical groupings of functions that facilitate the use of modular components and open-system designs. Functional partitioning is also useful in understanding how existing equipment or components (including commercial) will function with or within the system.

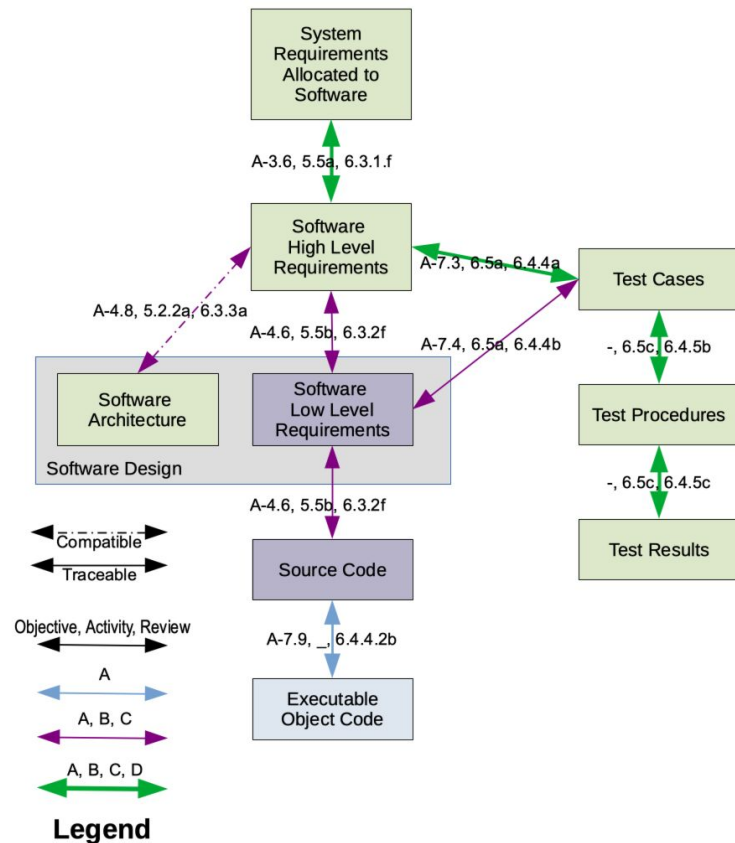
# Traceability graph and its nodes



# Document organization

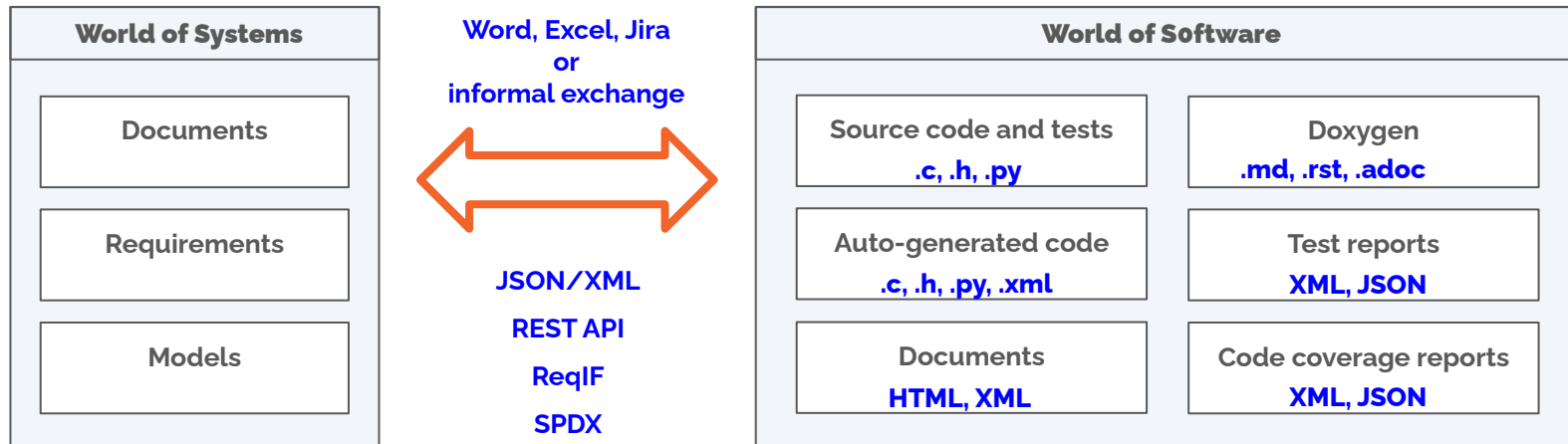
At least three spec layers (DO-178C, ECSS):

- 1. Business-level
  - Vision / Motivation
  - Use cases
  - Target audience
- 2. High-level requirements
  - "A product" requirements
  - Implementation agnostic
  - Several products can be developed to this spec
  - Different suppliers
- 3. Low-level requirements
  - "The product" requirements
  - Specific supplier
  - Implementation-specific
  - Touches the ground (source code)
- For every layer, avoid linking to the same spec.



## Traceability — Three challenges

- 1) There is no tool that connects all artifacts together in a single representation.
- 2) There is limited open-source tooling for the World of Systems.
- 3) Exchanging data and metadata between tools and formats. Which human-readable format to use?





# A pragmatic response to three challenges

- 1) **There is no tool that connects all artifacts together in a single representation.**
  - It is not realistic that someone would solve this with just one open-source project.
  - Instead, identify the biggest gaps and solve them.
  - Integrate with other existing tools to ensure complete traceability coverage.
- 2) **There is limited open-source tooling for the World of Systems.**
  - Implement a lightweight user interface for editing documents and requirements..
  - At the same time, embrace the existing systems tools and bridge them with the Software World.
- 3) **Exchanging data and metadata between tools and formats.**
  - What is the best format for exchanging traceability information?
  - In any case, focus on supporting common data exchange formats like XML and JSON.



## Let's focus and start from somewhere

- Find a suitable text format to hold requirements.
- Review Doxygen, Sphinx, and other software documentation tools to determine if they support traceability.
- Create a document model that supports traceability graph modeling and refine it to interface with other formats.
- Parse source code and integrate its elements into the traceability graph.
- Focus on publishing HTML and PDF as the most common output formats.
- Develop a GUI to assist in creating and editing documentation and requirements.
- Implement compatibility with other formats.



---

**StrictDoc —**

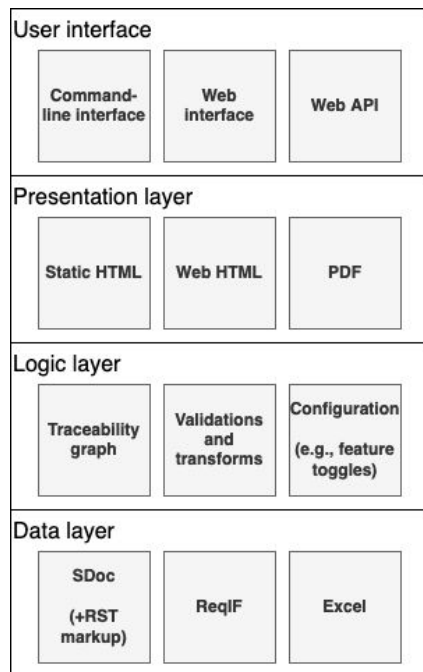
**Implementation details**



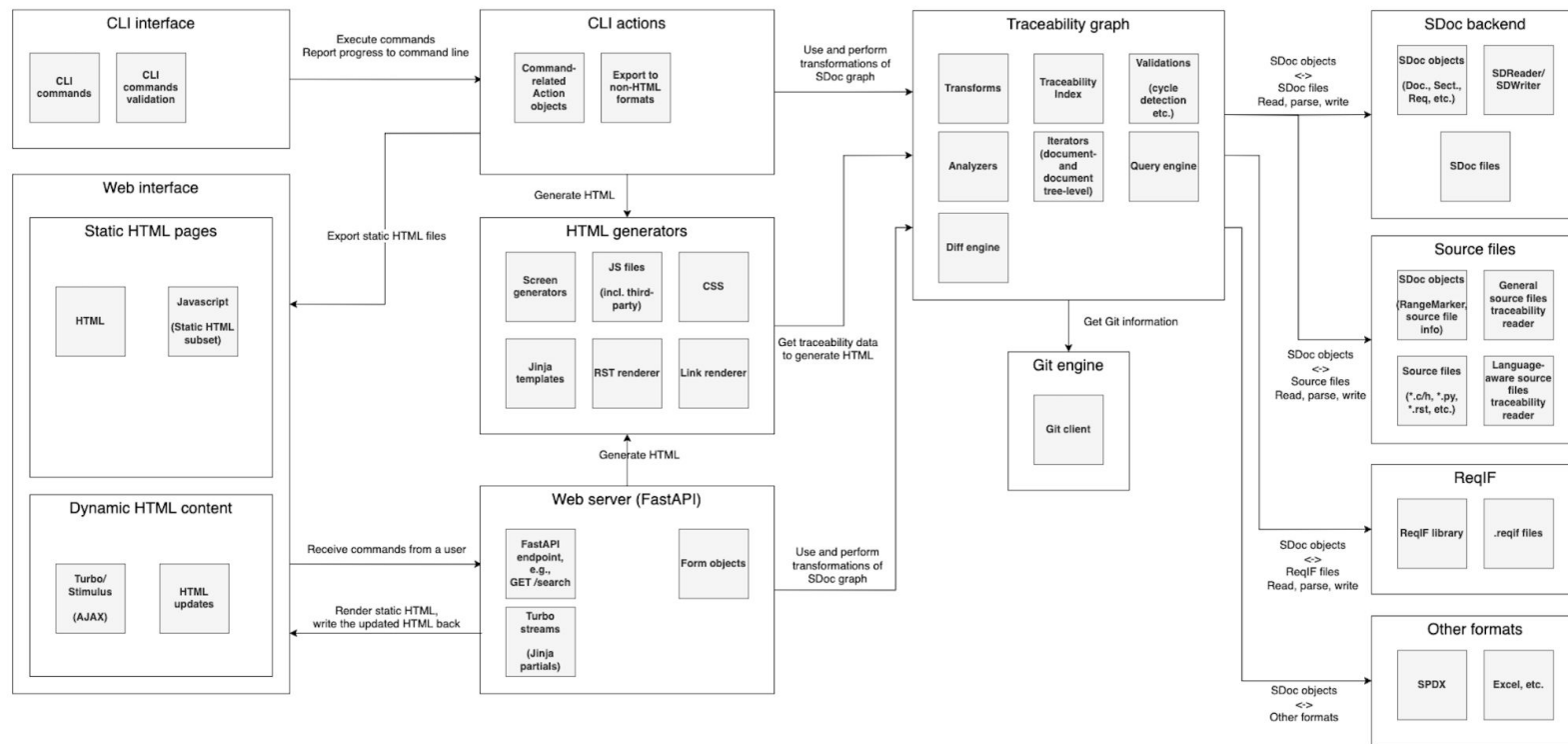
## StrictDoc — Technical details

- Requirements are stored in text files in the SDoc format.
- 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:
  - textX and docutils for parsing SDoc and RST
  - FastAPI and minimal JavaScript for the editable web interface.

# StrictDoc high-level overview (simplified)



# StrictDoc high-level architecture





# ReqIF

- [ReqIF](#) library is a satellite project of StrictDoc.
- Bottom-up implementation derived from ReqIF files in the wild.
- Integration tests:
  - Examples generated with Polarion, Doors, Enterprise Architect, ReqIF Studio
  - Test samples from Java's ReqIF at [ci.eclipse.org](http://ci.eclipse.org)
- There is no one way to export/import ReqIF.
- Every tool suggests its own document structure and type system.
- StrictDoc has its default ReqIF scheme based on a best guess.
- The lack of a common scheme is also the case for Excel export/import
  - See [ReqXLS: Is it possible to agree on a common Excel format for exchanging requirements? #798](#).
- When a general algorithm is hard to achieve, a custom Python converter always works.

---

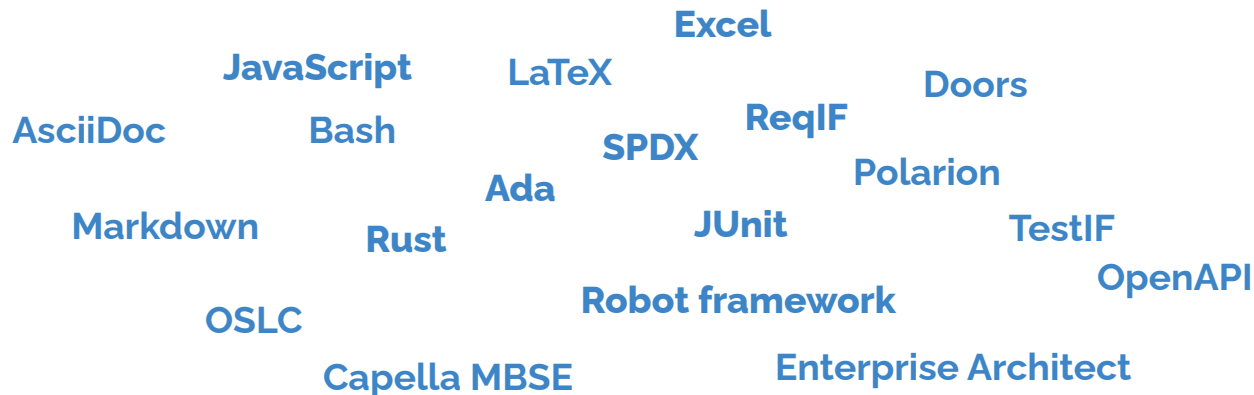
**StrictDoc —**

**Development plan, roadmap, backlog**



## Feedback from the user community

- Many users want the same feature set, many users want something else.
- As of Q3 2025, there are 65 open feature requests of various types.
- Features that are in high demand are implemented with higher priority.
- Tech keywords from the requested features and interfaces to other tools, new and recently implemented:



Excel  
JavaScript  
LaTeX  
Doors  
ReqIF  
SPDX  
Polarion  
TestIF  
OpenAPI  
Enterprise Architect  
Robot framework  
JUnit  
Ada  
Markdown  
OSLC  
Capella MBSE  
Rust  
Bash  
AsciiDoc

# StrictDoc — Roadmap 2023

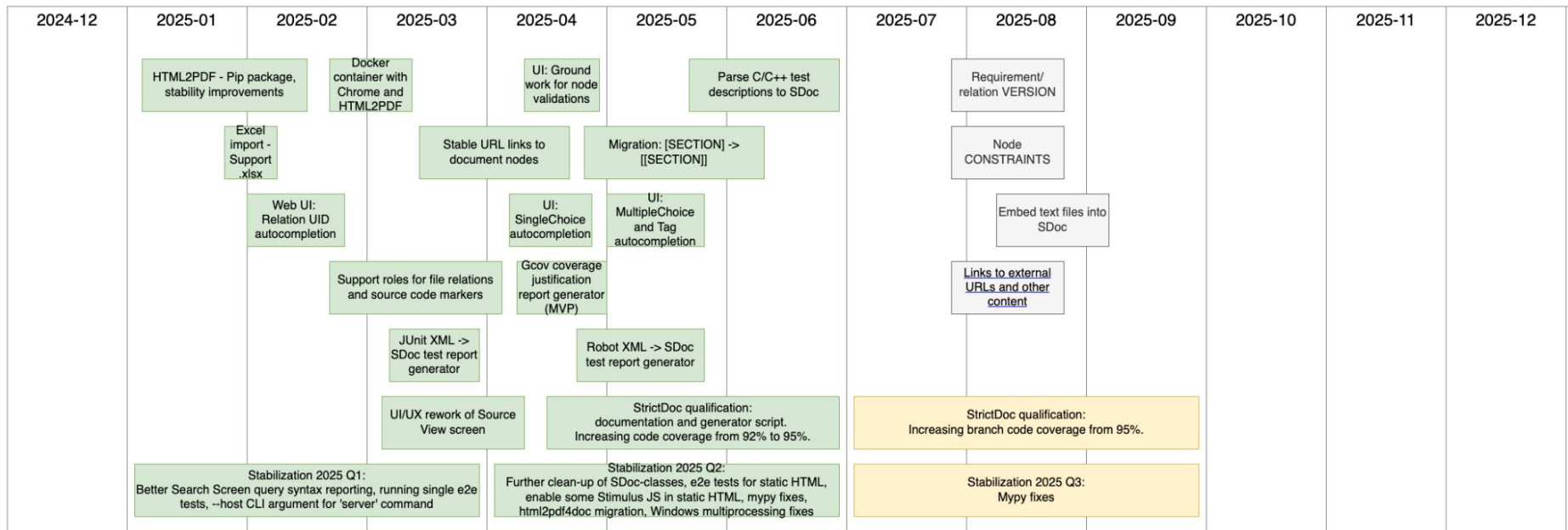
2023-01	2023-02	2023-03	2023-04	2023-05	2023-06	2023-07	2023-08	2023-09	2023-10	2023-11	2023-12	2024-01	
Web interface: Document and Project Tree screens				UI: Move TOC nodes	UI: On-demand rendering of web pages		Child / reverse parent links	Mermaid diagrams	Parent/child relations and roles (refines, implements, verifies)	Search query engine, search screen, export filters	Basic Git operations	<a href="#">Basic SPDX export</a>	
End-2-end framework and tests						Section UID, anchors and links	Project statistics screen	CLI: --config parameter	UI: Copy to buffer, Edit grammar/requirement tabs	Single-line markers	<a href="#">Project tree diff/changelog</a>	Forward range links to file	
Project-level TOML config file		TextMate bundle / syntax highlighting		Config: Feature toggles	Config: include/exclude docs/files		Basic user-scriptable ReqIF export/import					Forward range-based links to source file	
				Auto-assign requirement UID		Basic export to Graphviz (unsuccessful)			Basic ReqIF to JSON export		<a href="#">Traceability Matrix Screen v0</a>		
Stabilization 2023 Q1: PyInstaller, SDoc grammar, JS fixes, DTR screen: smaller requirement cards, requirements-to-source edge cases, ReqIF			Stabilization 2023 Q2: Requirements-to-source edge cases, RST user requests and edge cases, relative paths-related edge cases, PyInstaller			Stabilization 2023 Q3: UI: Editing grammar fields, section UID edge cases, Jinja templates on macOS, performance optimizations, RST edge cases, Windows/Pip long file names issue			Stabilization 2023 Q4: Linking to files (YAML files support), flaky tests issues and end2end test robustness work, PyInstaller issue, Traceability Index API internal refactoring			Improvements in and the simp removed	
						StrictDoc requirements upgrade (2 levels of specification)							
HTML2PDF (standalone and independent JS library, PDF documents from SDoc's HTML, front page with meta information, running titles, UI: export to PDF)												HTML2PDF (Export to handling table	



# StrictDoc — Roadmap 2024

	2024-01	2024-02	2024-03	2024-04	2024-05	2024-06	2024-07	2024-08	2024-09	2024-10	2024-11	2024-12
	Basic SPDX export	Multi-document Excel export	Human field titles (UID -> Unique identifier)	UI: Includable/composable documents (resolving asset paths, corner cases)	Allow arbitrary placed free texts		HTML2PDF all-in-one bundle document			Language-aware parsing of source files for traceability	Traceability to C++	
g		Forward range-based links to source files	UI: Editing non-requirement nodes	Grammars in separate files	ReqIF multiline fields/XHTML, new default ReqIF format profile		ReqIF relations/roles			Traceability to C/Python/* functions	Doxygen TAGFILE export	
		Forward range-based links to source files	JSON export		Final removal of REFS in favour of RELATIONS	UI Form validation: SingleChoice and REQUIRED fields		Passthrough -> export command migration		LINK to document		Incremental generation of source files
aceability Matrix green v0		Custom views, static HTML									Docs: StrictDoc feature map and screenshots	
ity		Stabilization 2024 Q1: Improvements in handling of UI templates, removal of ng_level and the simplification of textX parsing step, pybtex* code removed, more requirements work in L1 and L2		Stabilization 2024 Q2: Fix the outdated pickle cache issue, stronger typing in the codebase (mypy), HTML2PDF/Windows/UTF8			Stabilization 2024 Q3: Auto-escaping of Jinja templates, improvements of UID handling, fixes for custom nodes editing/linking, new chromedriver headless mode, mypy fixes			Stabilization 2024 Q4: Dropped Python 3.7 support, autoescaping-related fixes, configurable cache_dir option, fix fork() on macOS, deploy StrictDoc documentation to Read the Docs		
	HTML2PDF integration to StrictDoc (Export to PDF in UI and CLI, handling table and paragraph breaks)			Traceability navigator POC								

# StrictDoc — Roadmap 2025





## Existing limitations → Further work

- Single-user Git workflow. No user accounts, no concurrent use, no committing to Git from the web UI.
- StrictDoc's focus is on documents/requirements. Modeling and formal methods have been out of scope.
  - Integration with other tools should be possible.
- Analyzing and acting on requirements with AI is a promising direction.
- StrictDoc has not been tested enough on very large multi-repo projects.
  - To be implemented: recursively joining the traceability graphs of sub-projects.
- Traceability mechanics is a research topic in its own right.
  - The tooling is in place but how to connect requirements to software in the best way?
    - The easiest approach: connect requirements to whole source files.
  - How to join the parent project traceability graph with the OSS/OTS project graphs?



## Other open source requirement tools

- StrictDoc is just one tool of many.
- StrictDoc is one of the most mature.
- Some examples:

See also the RTEMS qualification package — a strong case of low-level software specification with 100% branch code coverage:

[RTEMS's approach to low-level SW specification](#)

Doorstop

RDM

BASIL

OpenFastTrace

Sphinx-Needs

trlc and LOBSTER

Duvet

NASA's FRET

Open-Needs

sphinx-traceability-extension

See more tools in this GitHub gist:

[Open source requirements managements tools](#)



## Open questions?

- Some questions are answered in the [StrictDoc FAQ](#).
- Contact us directly via:
  - Email
  - Discord
  - Mailing list
  - Join the weekly StrictDoc Office Hours call.
- Contact information is provided at the beginning of the slide deck.