StrictDoc project

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

StrictDoc — Development team

StrictDoc core team:
Stanislav Pankevich, s.pankevich@gmail.com
Maryna Balioura, 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

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

```
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.

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

```
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)
    """
```

```
[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 .
```

REQUIREMENT

1.5. Document model

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

REQUIREMENT

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: • UID • Document version • Document classification • Document authors.
PARENT RELATIONS:	← **SDOC-SSS-53** Document meta information (UID, version, authors, signatures, etc) ← **SDOC-SSS-75** Document versioning
FILE	</> strictdoc/backend/sdoc/models/document_config.py, *lines: 1-191*, entire file

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

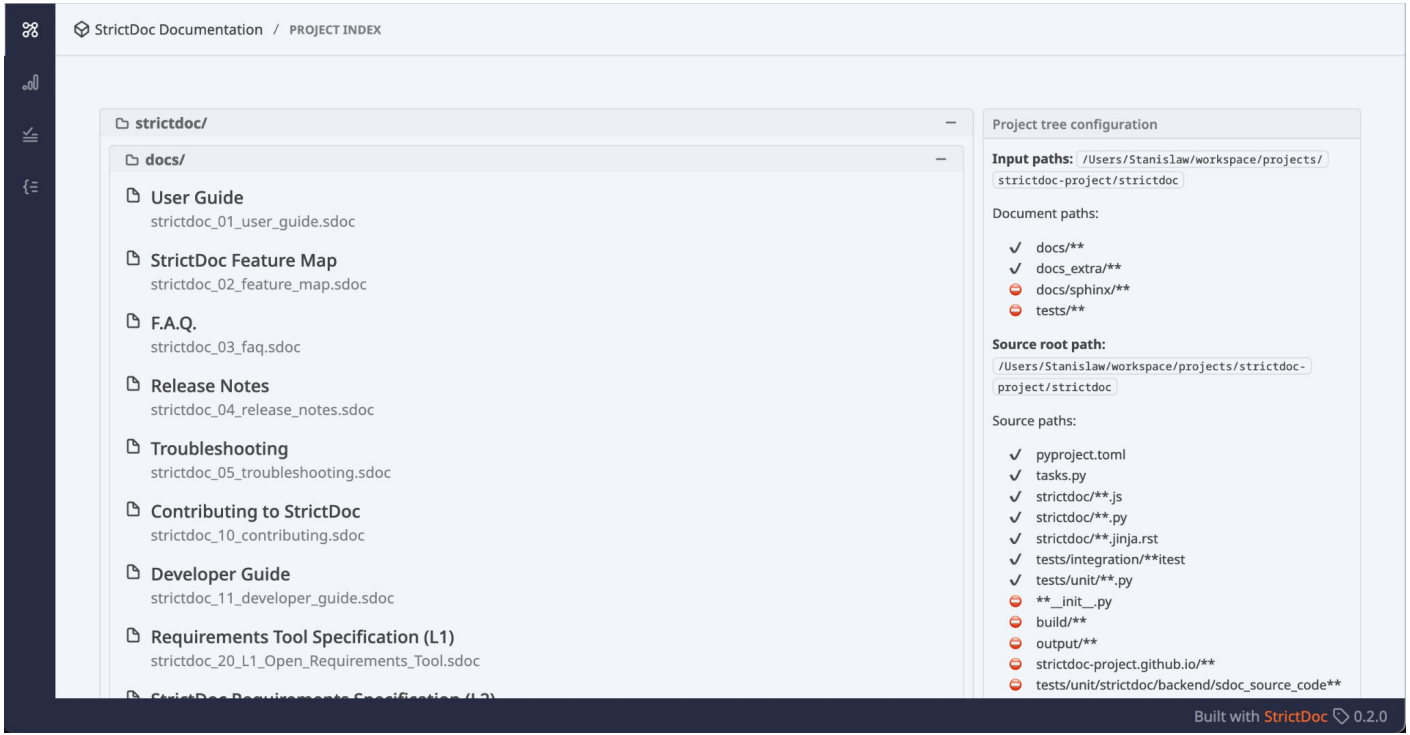
strictdoc_02_feature_map.sdoc

```
[FEATURE]
UID: SDOC-FEAT-1
STATUS: Stable
TITLE: SDoc text markup
STATEMENT: >>>
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.
<<<
DOCUMENTATION: >>>
- [LINK: SECTION-UG-SDoc-syntax]
<<<

[SECTION]
TITLE: Use case

[TEXT]
STATEMENT: >>>
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:
```

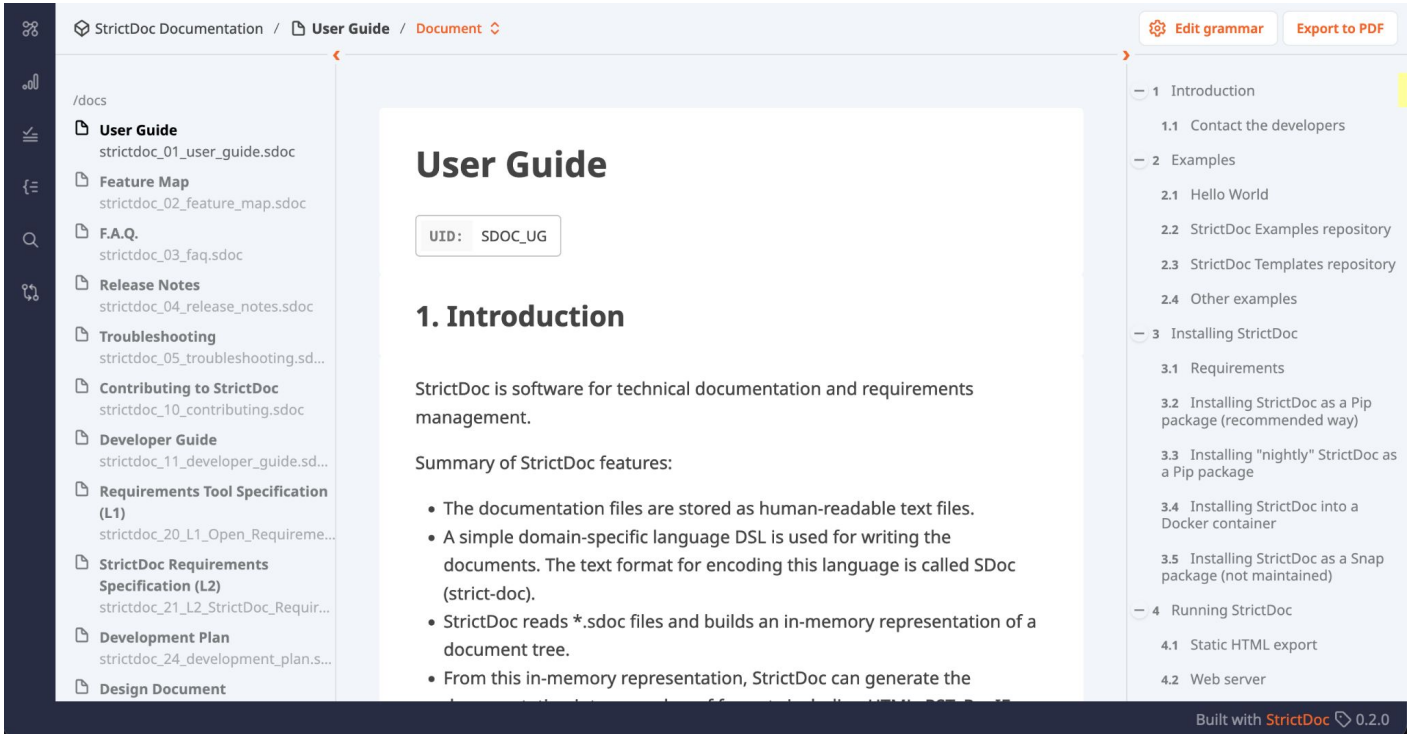
strictdoc_02_feature_map.sdoc

```
[DOCUMENT]
TITLE: Feature Map
UID: SDOC_FEATURE_MAP
REQ_PREFIX: SDOC-FEAT-
OPTIONS:
  REQUIREMENT_STYLE: Inline
  REQUIREMENT_IN_TOC: True

[GRAMMAR]
ELEMENTS:
- TAG: TEXT
  FIELDS:
  - TITLE: UID
    TYPE: String
    REQUIRED: False
  - TITLE: STATEMENT
    TYPE: String
    REQUIRED: False
- TAG: FEATURE
  FIELDS:
  - TITLE: MID
    TYPE: String
    REQUIRED: False
  - TITLE: UID
    TYPE: String
    REQUIRED: False
  - TITLE: STATUS
    TYPE: String
    REQUIRED: False
  - TITLE: TITLE
```

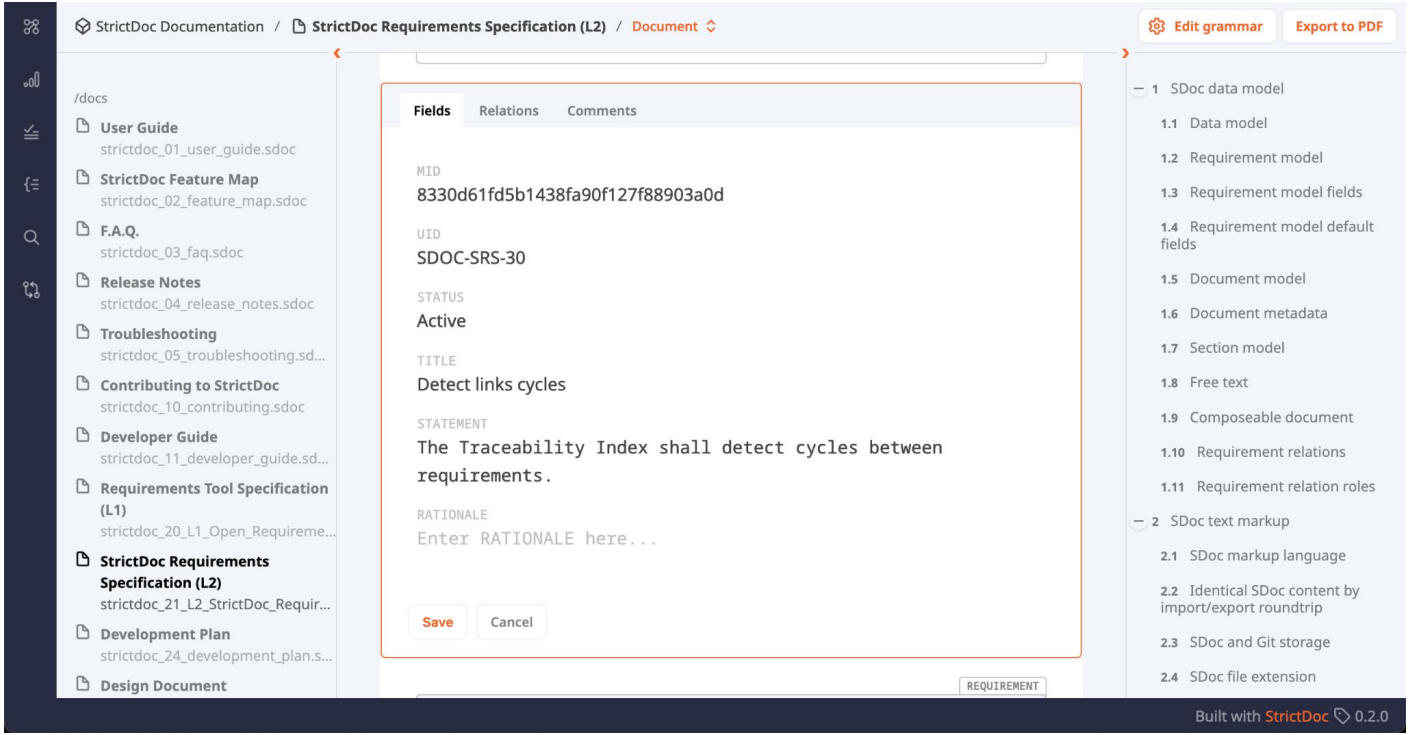
StrictDoc — Project tree



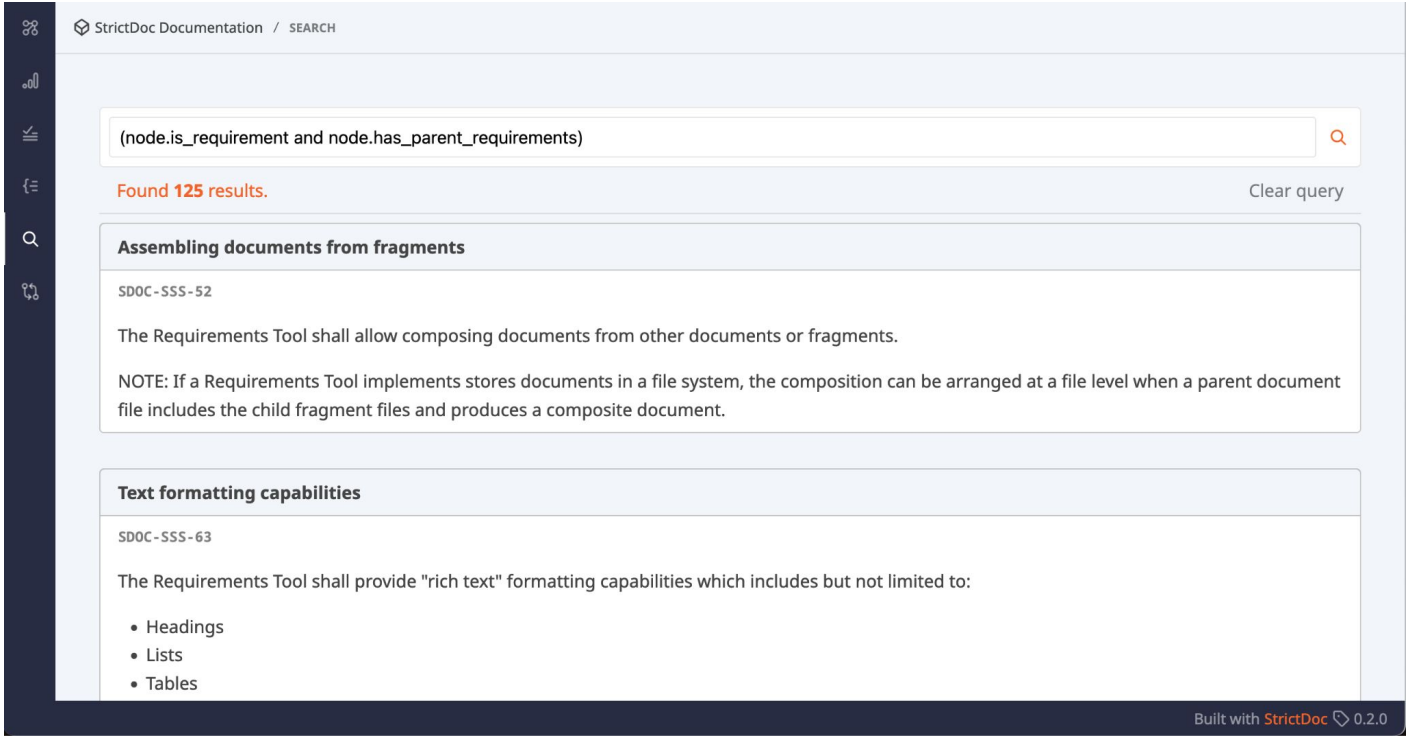
StrictDoc — Web interface



StrictDoc — Web interface / Edit node

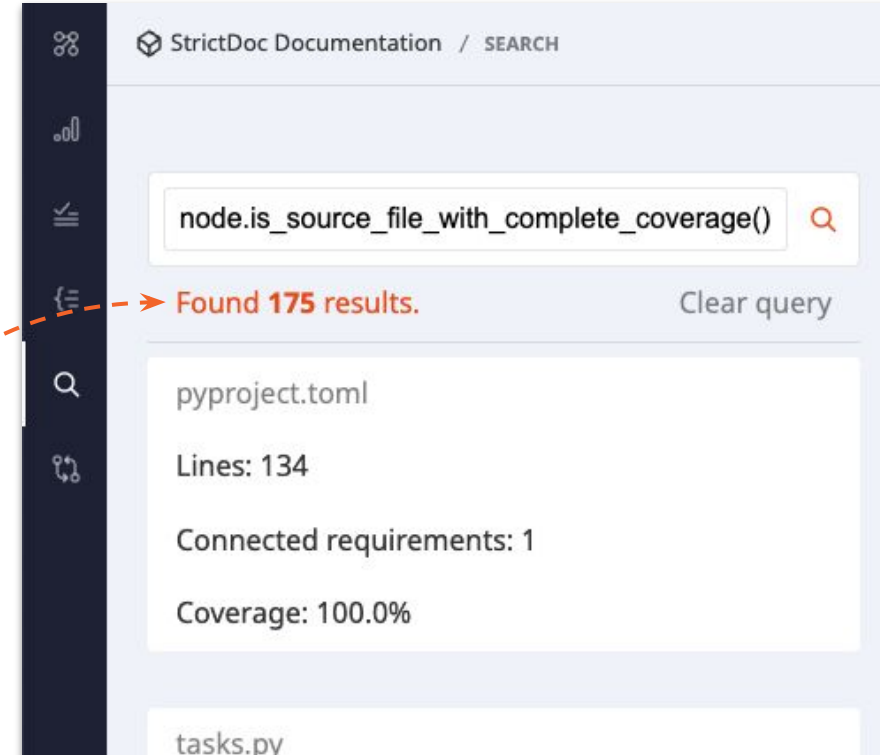


StrictDoc — Search screen and query engine

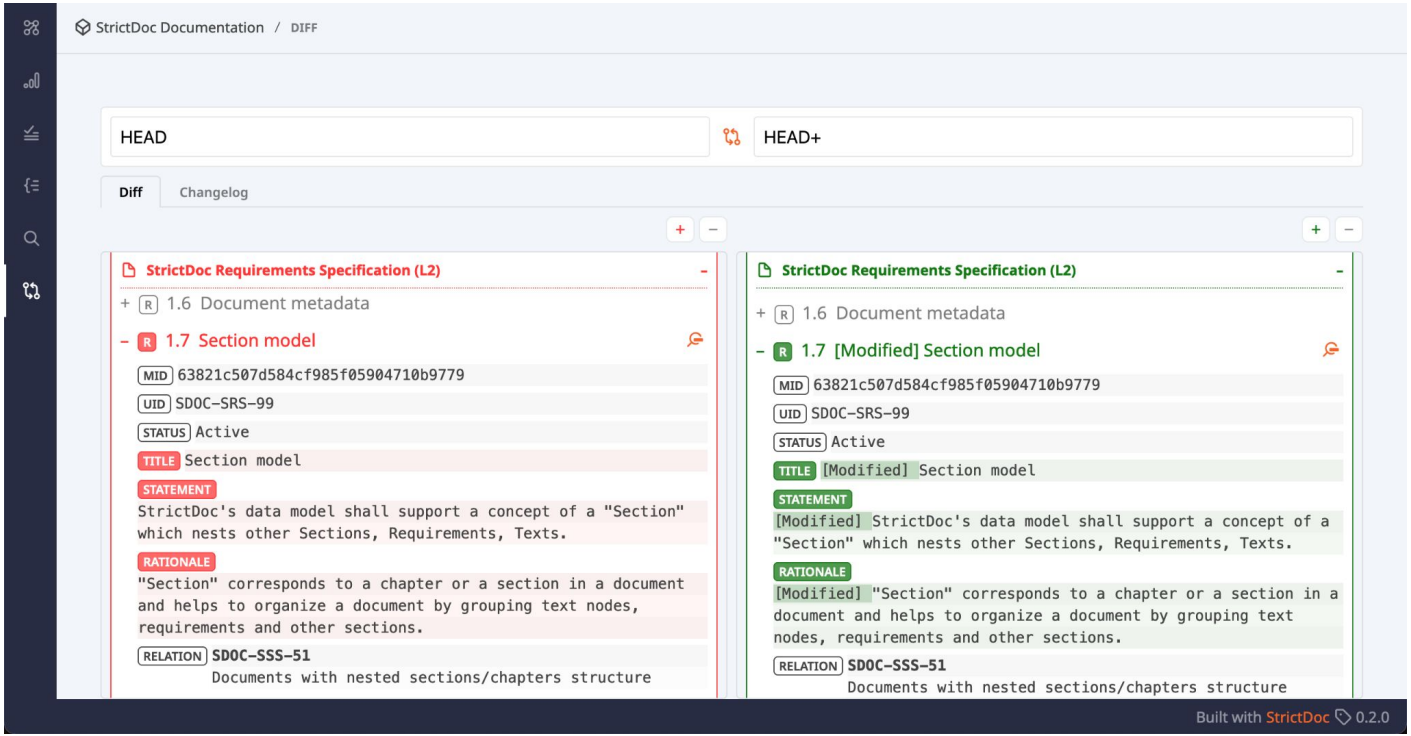


StrictDoc — Traceability metrics

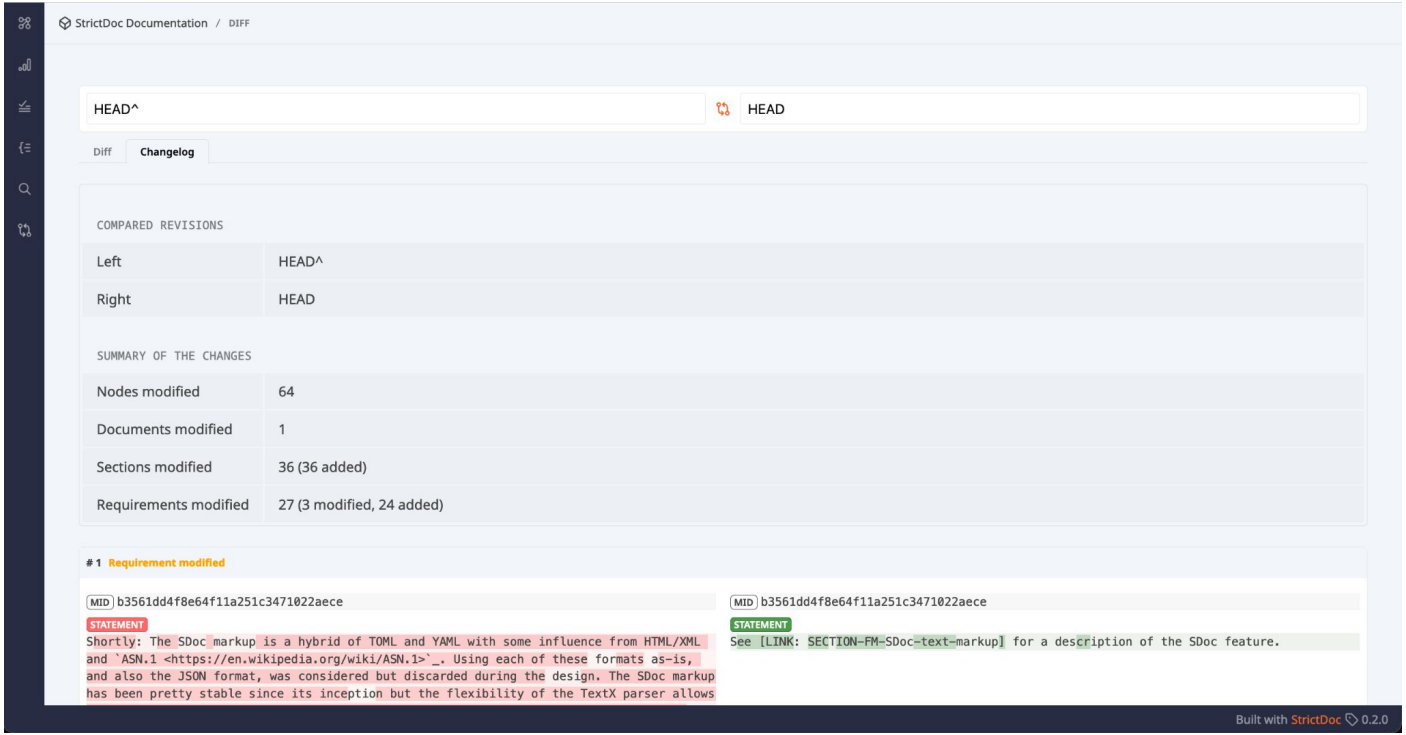
- Two interconnected features: Statistics and Search.
- Inspired by the ECSS SW metrication handbook
- Writing project-specific statistics generators in Python.
- Calculate total numbers of entities
- Calculate the number of present and missing relations
- Each statistic can be queried individually.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



StrictDoc — Diff screen



StrictDoc — Changelog screen

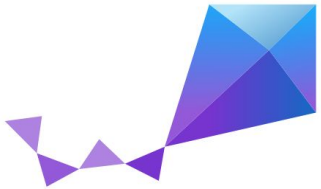


StrictDoc — Case studies

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



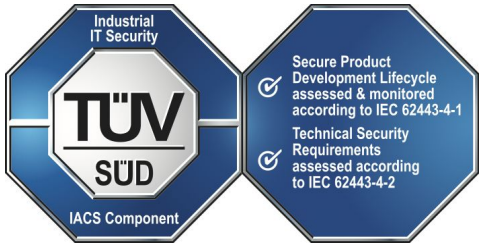
- The Zephyr Safety Working Group is working to make the Zephyr RTOS certifiable for use in safety-related projects:
  - Zephyr Safety Overview
  - Safety FAQ
- StrictDoc is used in two ways:
  - Technical requirements are captured in SDoc files in a dedicated Git repository: https://github.com/zephyrproject-rtos/reqmgmt.
  - 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.

IGLOS
SECURE BEACON
by LINUTRONIX

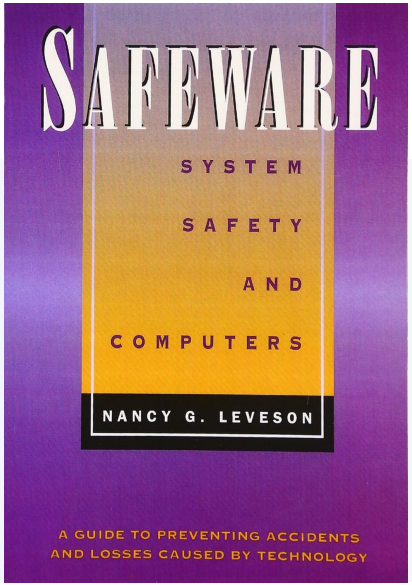


StrictDoc —

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)`

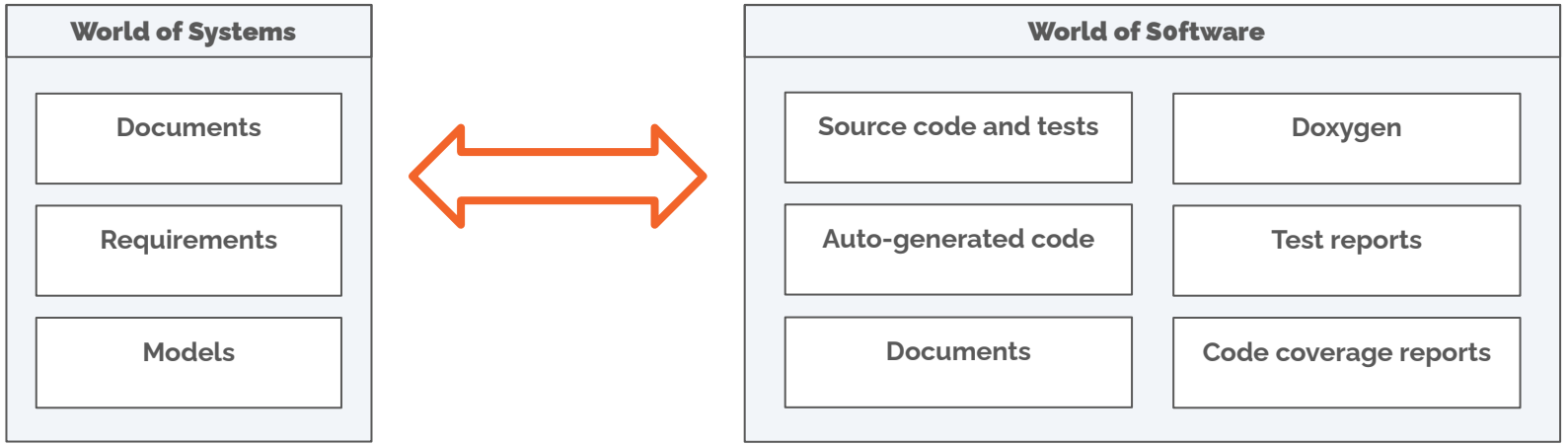
Relations/roles:
e.g., parent, child, parent/refines, etc.

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

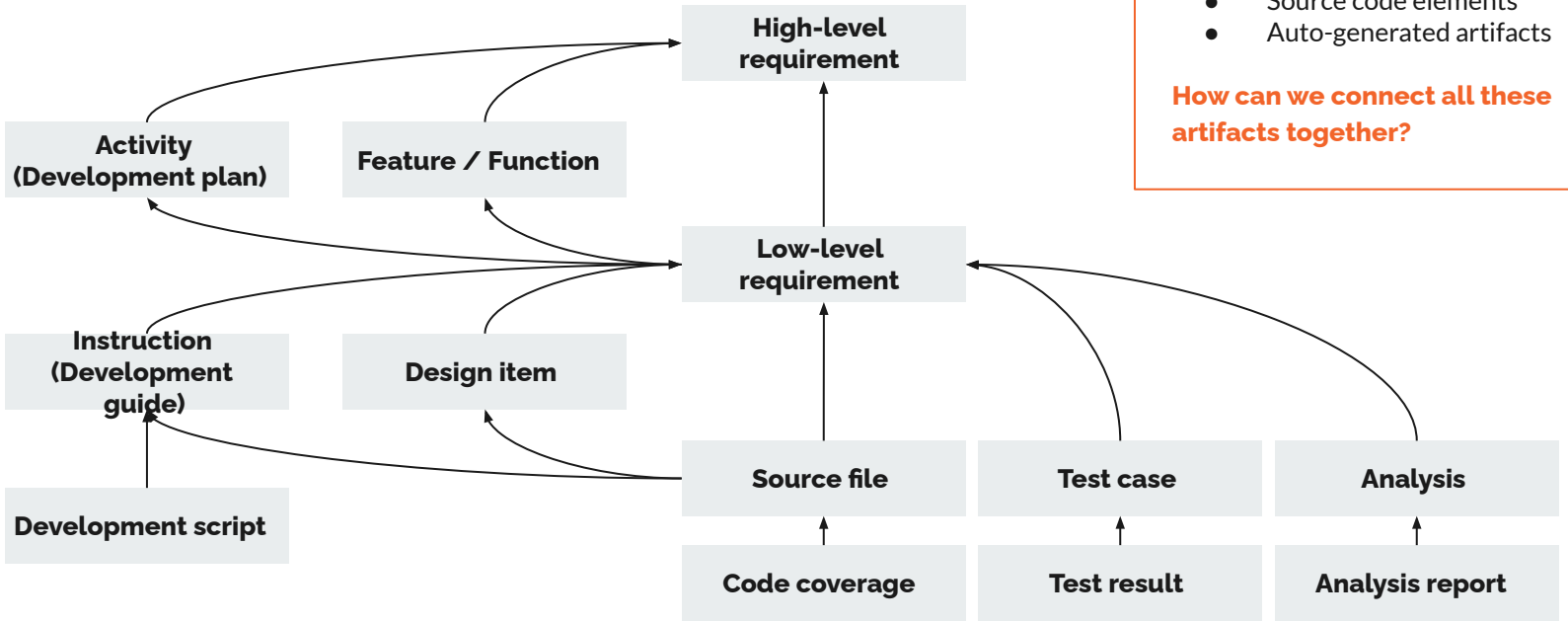
- For each spec level, a functional analysis results in function partitioning.
- Discovered functions become separate documents or spec chapters.
- Requirements that are functionally related are easier to implement, maintain, verify.
- Functional, performance, interface requirements allocated to functions.
- 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.

SYSTEMS ENGINEERING FUNDAMENTALS, p.46

Traceability graph and its nodes



Mixed content from multiple tools:

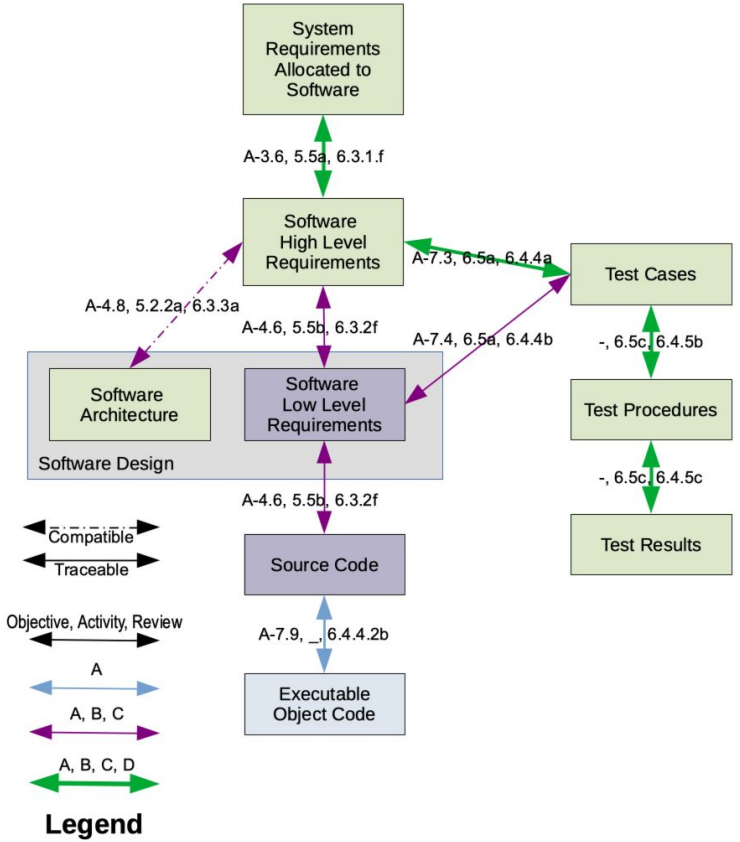
- Documents
- Source code elements
- Auto-generated artifacts

How can we connect all these artifacts together?

Document organization

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

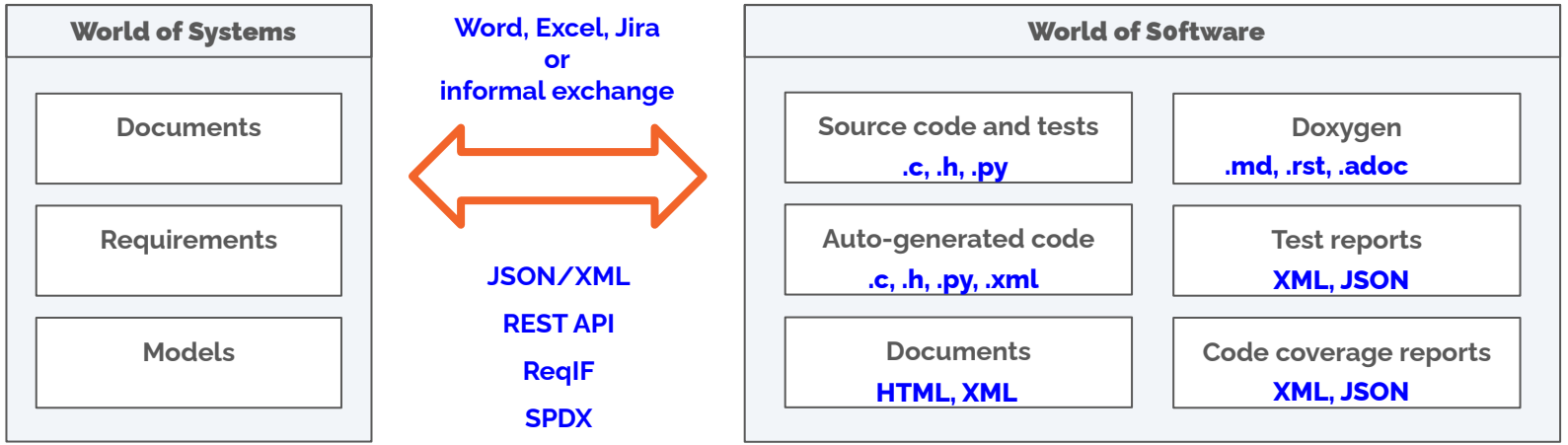
- 1. Business-level
  - Vision / Motivation
  - Use cases
  - Target audience
- 2. High-level requirements
  - "A product" requirements
  - Implementation agnostic
  - Several products can be developed to this spec
  - Different suppliers
- 3. Low-level requirements
  - "The product" requirements
  -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



[Wikipedia – DO-178C Traceability]

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.

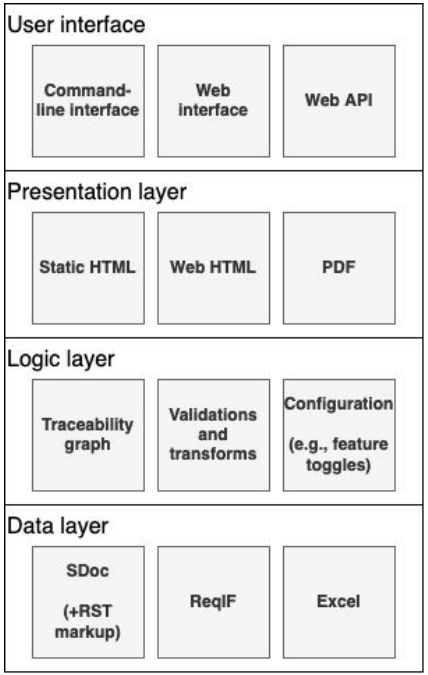
StrictDoc —

Implementation details

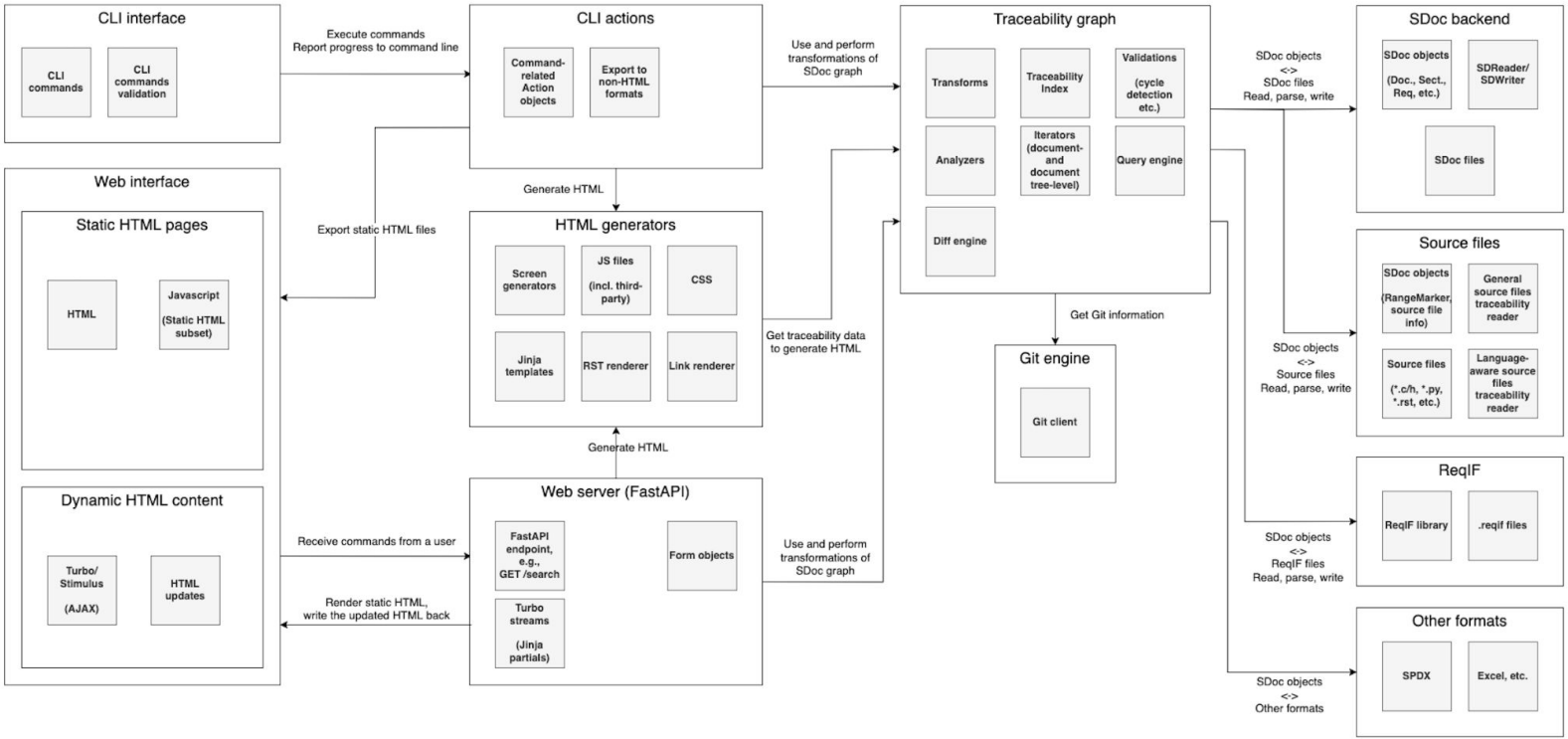
StrictDoc — Technical details

- 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

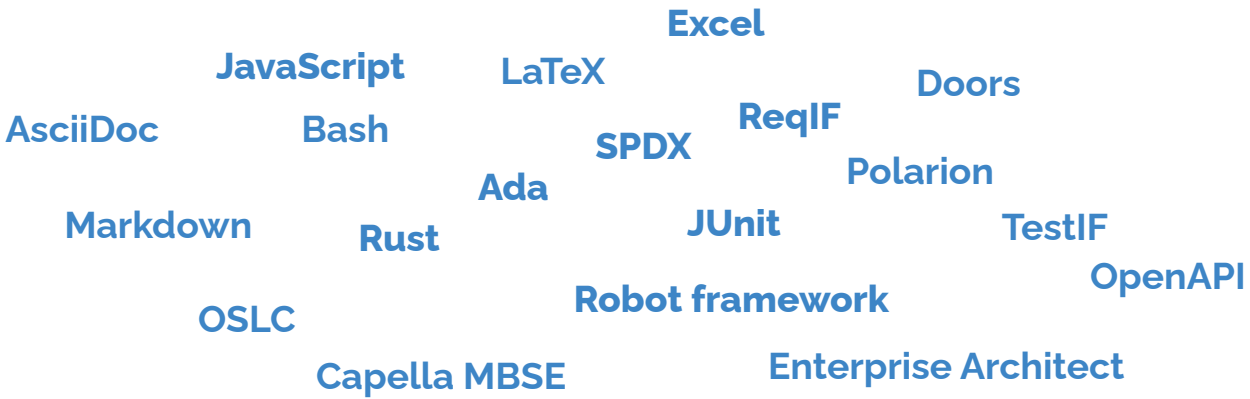
- ReqIF library is a satellite project of StrictDoc.
- Bottom-up implementation derived from ReqIF files in the wild.
- Integration tests:
  - Examples generated with Polarion, Doors, Enterprise Architect, ReqIF Studio
  - Test samples from Java's ReqIF at ci.eclipse.org
- There is no one way to export/import 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 —

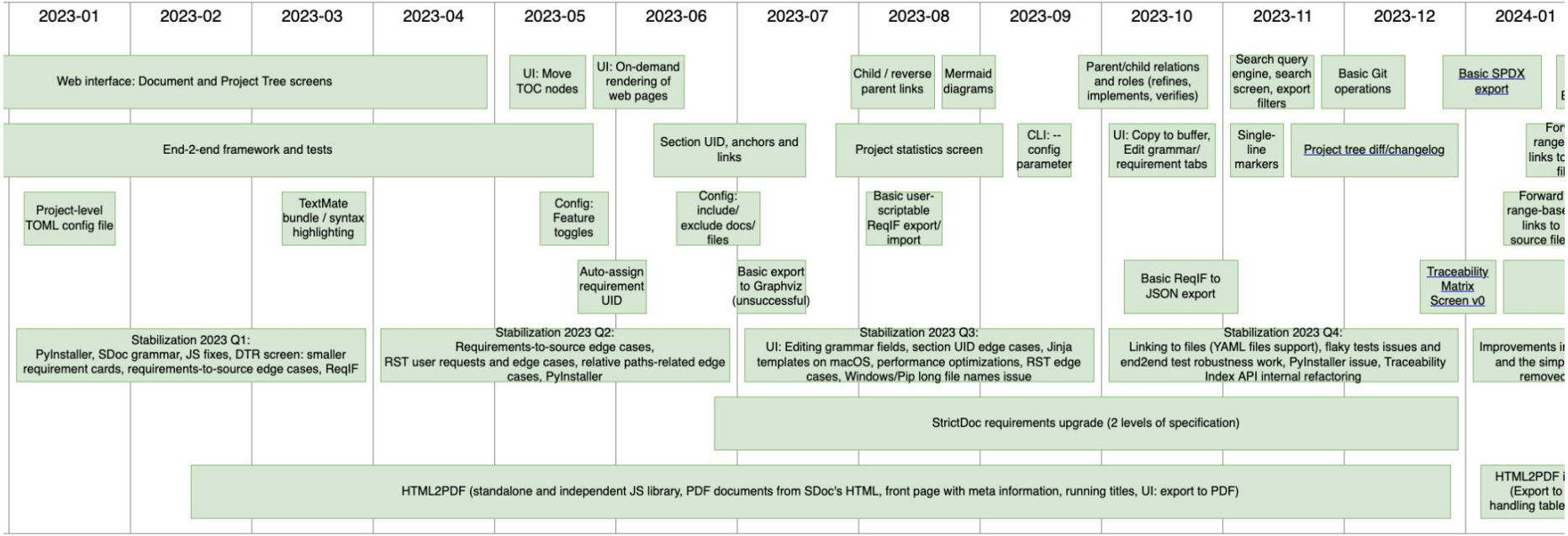
Development plan, roadmap, backlog

Feedback from the user community

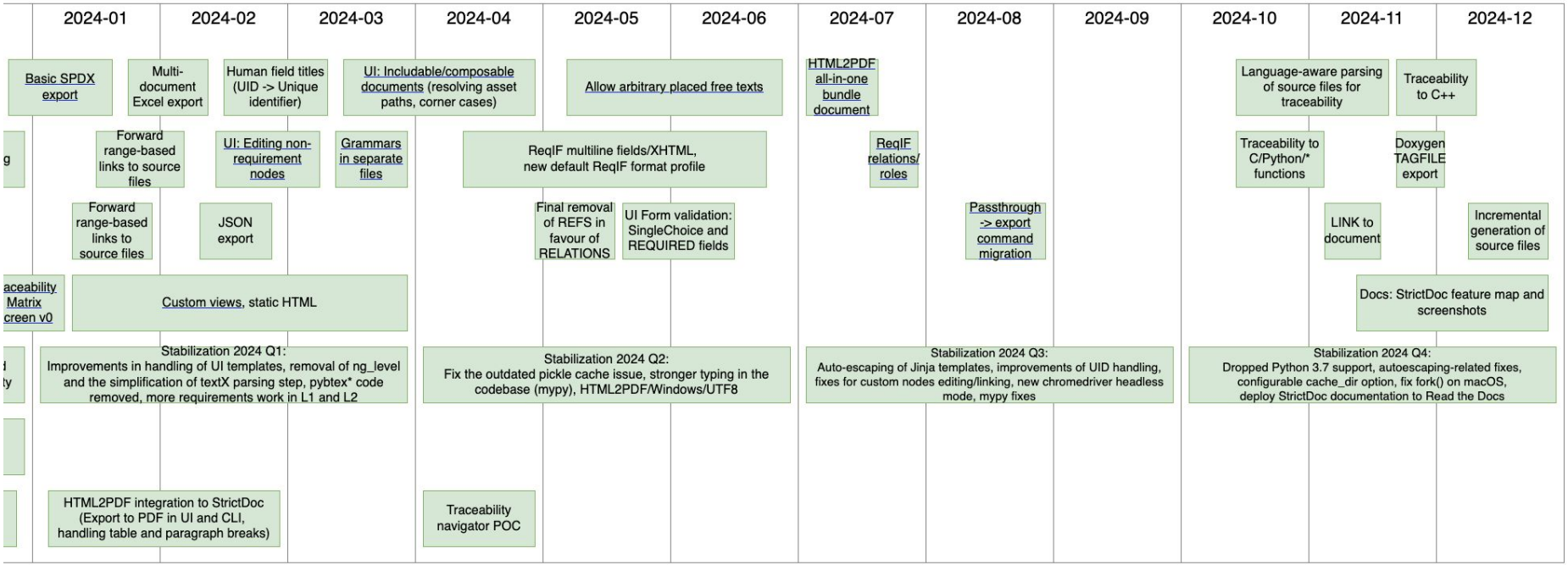
- Many users want the same feature set, many users want something else.
- As of Q3 2025, there are 65 open feature requests of various types.
- Features that are in high demand are implemented with higher priority.
- Tech keywords from the requested features and interfaces to other tools, new and recently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



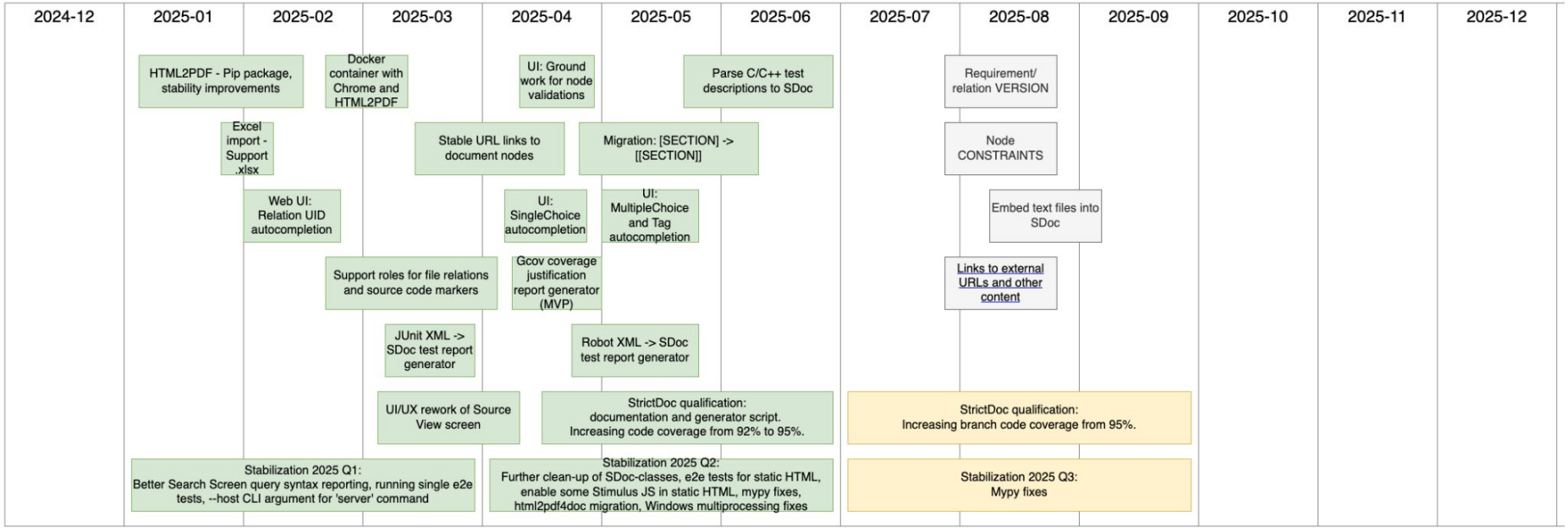
StrictDoc — Roadmap 2023



StrictDoc — Roadmap 2024



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:

RDM

Doorstop

BASIL

OpenFastTrace

Sphinx-Needs

trlc and LOBSTER

Duvet

NASA's FRET

Open-Needs

sphinx-traceability-extension

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

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.