東京 2025 LINUX PLUMBERS CONFERENCE

TOKYO, JAPAN / DECEMBER 11-13, 2025

BASIL - Traceability as Code

Who I am

Luigi Pellecchia

Principal
Software Quality Engineer
Red Hat In-vehicle OS

LinkedIn

ELISA
Enabling Linux in
Safety Applications

Member of Technical Steering Committee
ELISA (Linux Foundation)

ELISA TSC

東京 2025 LINUX PLUMBERS CONFERENCE
TOKYO, JAPAN / DEC. 11-13, 2025

Agenda

BASIL Overview

What is it? What problems can it help solve? Distinctive factors

Traceability as Code

The idea and how it is implemented in BASIL

BASIL - Overview

Tool to generate and maintain a traceability matrix for software applications in collaborative and multi user environment

- Centralized web application with user interface and REST API
- Granular user permissions management
- Keep track of all the changes
- Work items life cycle
- Promote collaboration and clarifies gaps
- Embedded Test Infrastructure and plugins for external ones (e.g. KernelCI, LAVA)
- SPDX SBOM generation
- Work items import/export
- Support for in-app AI suggestions

Tool name "BASIL" comes from ASIL B

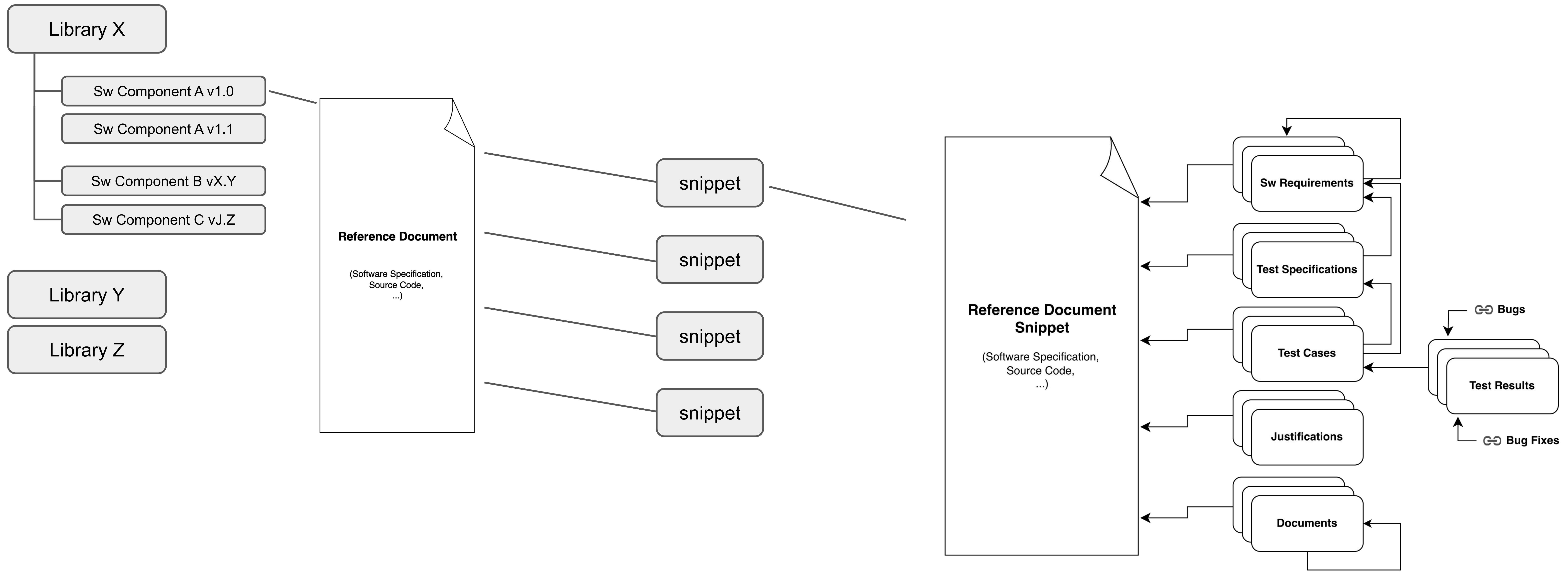
Born at Red Hat to support the In-Vehicle OS

Presented to ELISA on June 2023 during the annual workshop

Hosted on ELISA GitHub at https://github.com/elisa-tech/BASIL

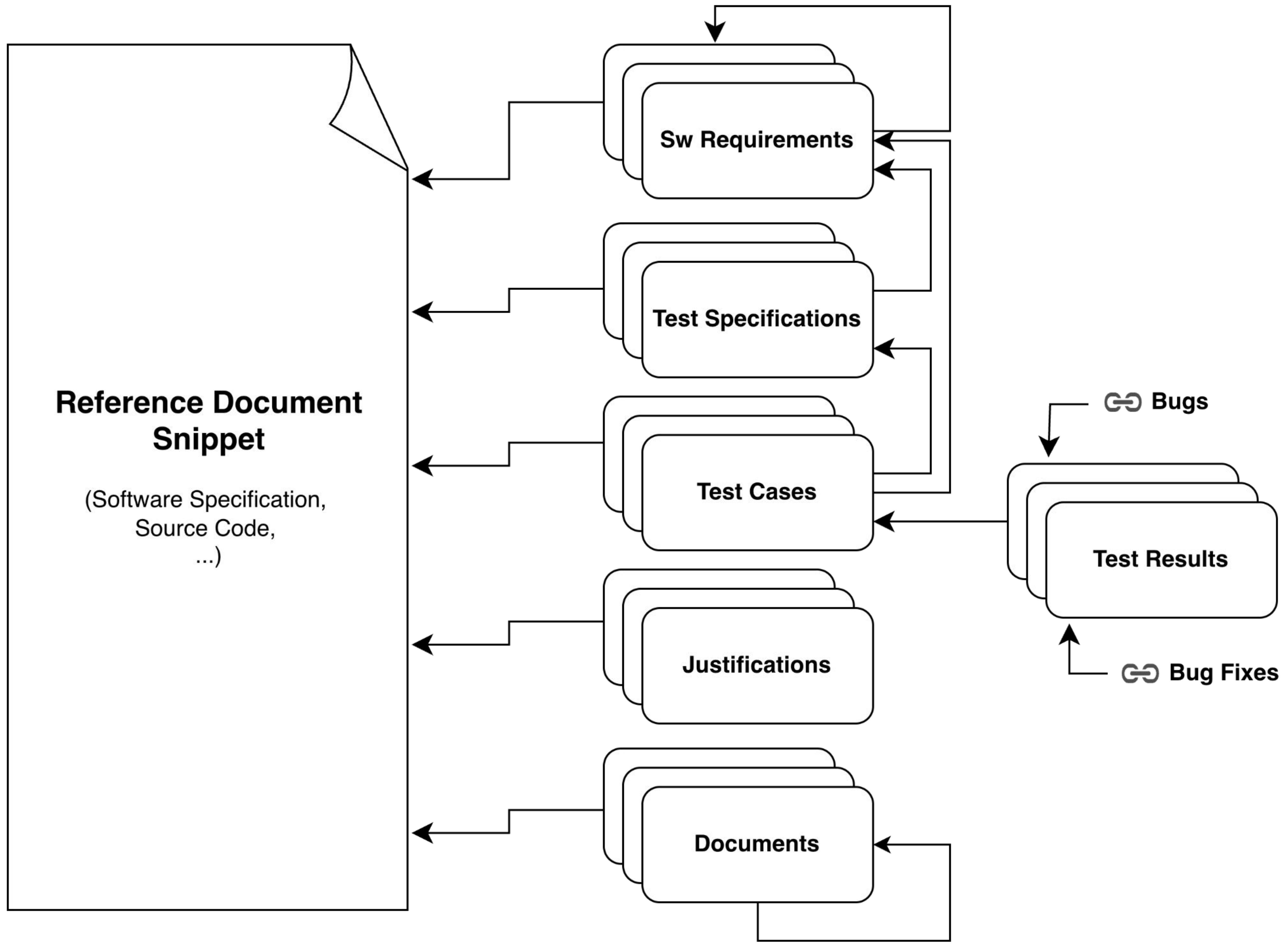
BASIL - Traceability workflow

User can generate a traceability matrix for a software component defining a Reference Document and assigning work items to any snippet of it



BASIL - Work items relationships in the traceability matrix

Overview of supported relationships between work items for each Reference Document snippet



Traceability for Linux Kernel

Goals

Generate a traceability matrix with complex hierarchy considering source code, specification, test cases, test results, bugs and more...

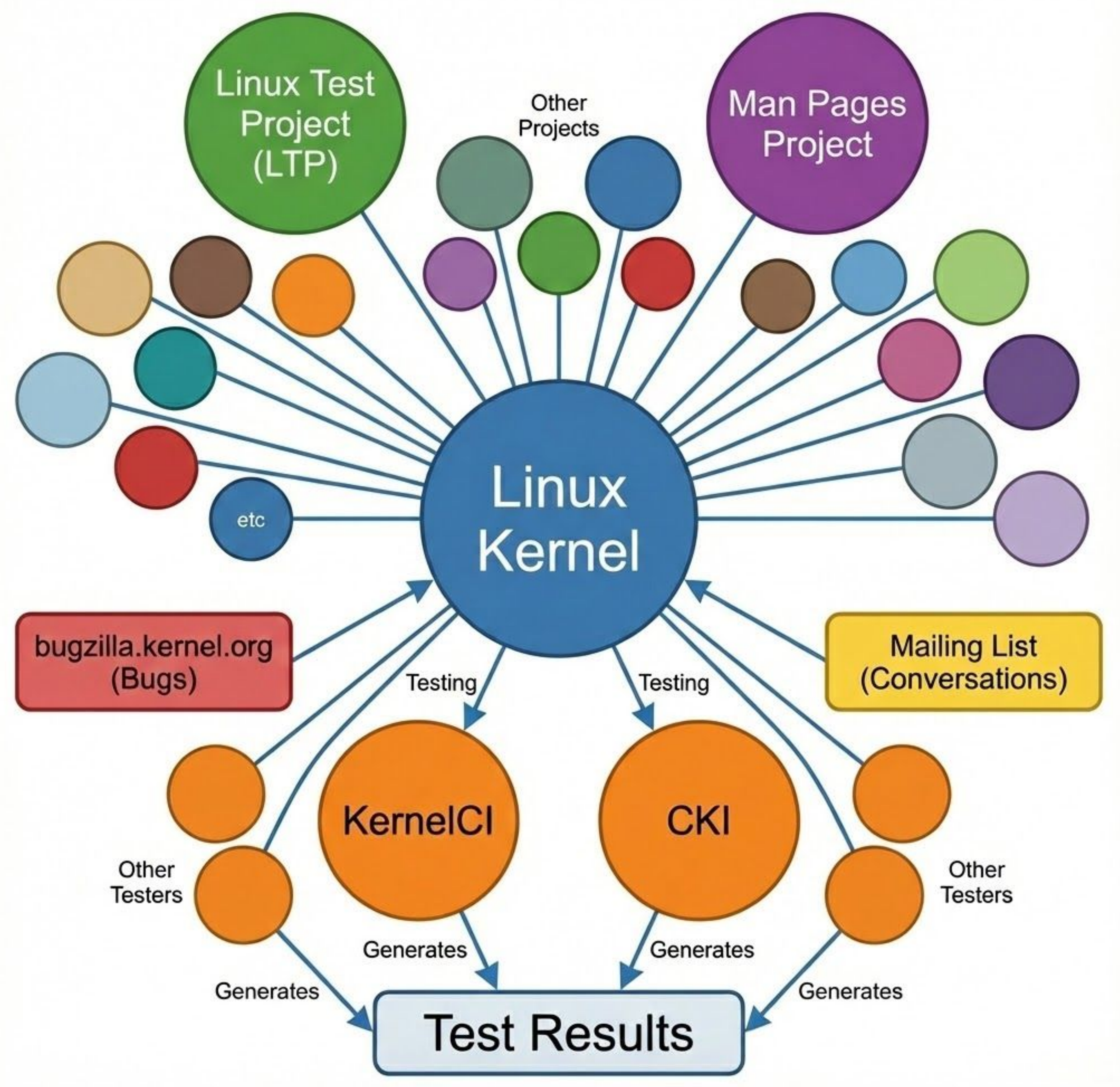
The same traceability matrix must be generated each time for a target version

The traceability matrix can be generated as part of a CI/CD pipeline

Challenges

Leverage existing data that is:

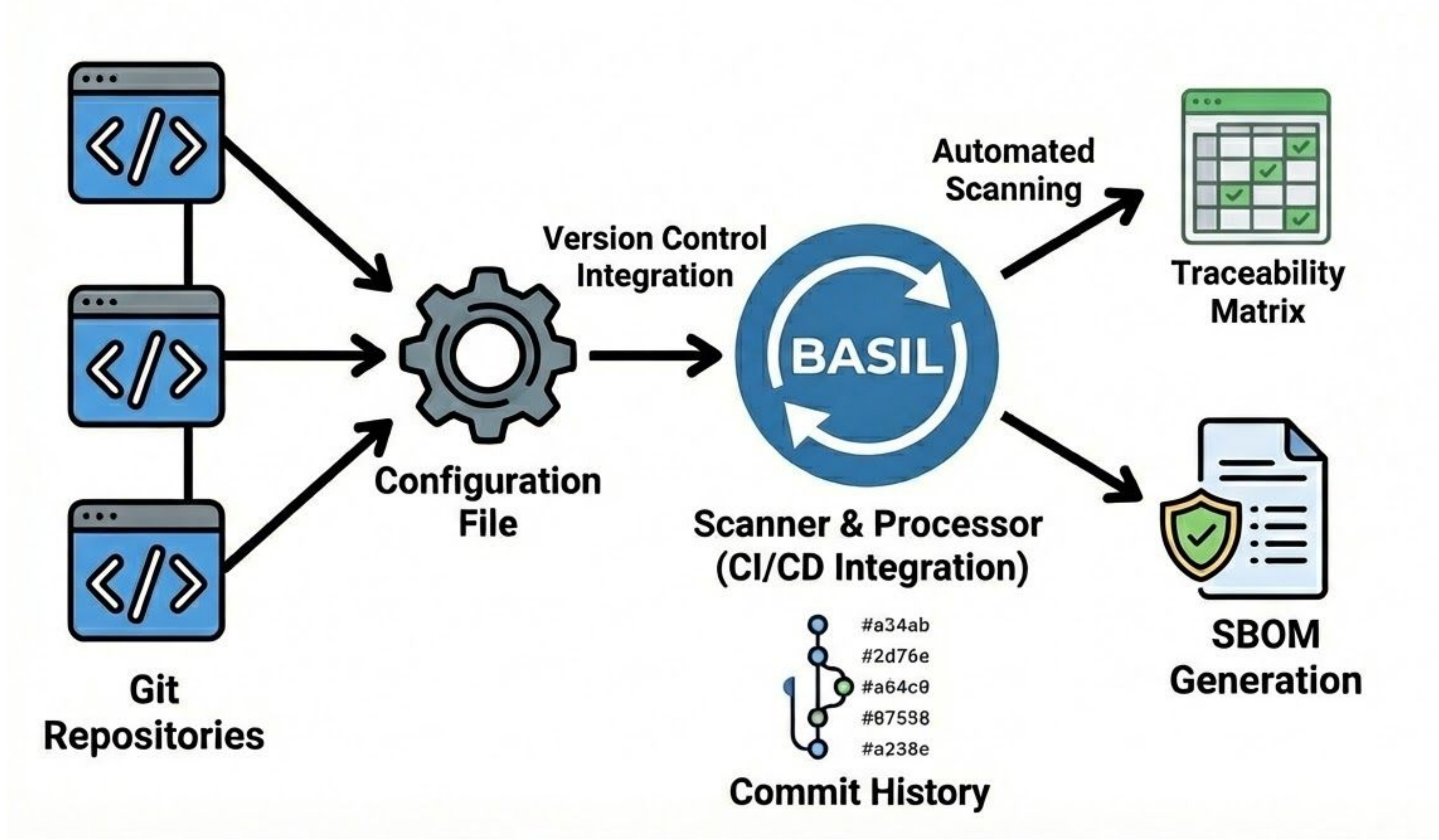
- scattered throughout multiple projects
- defined in a no standard way



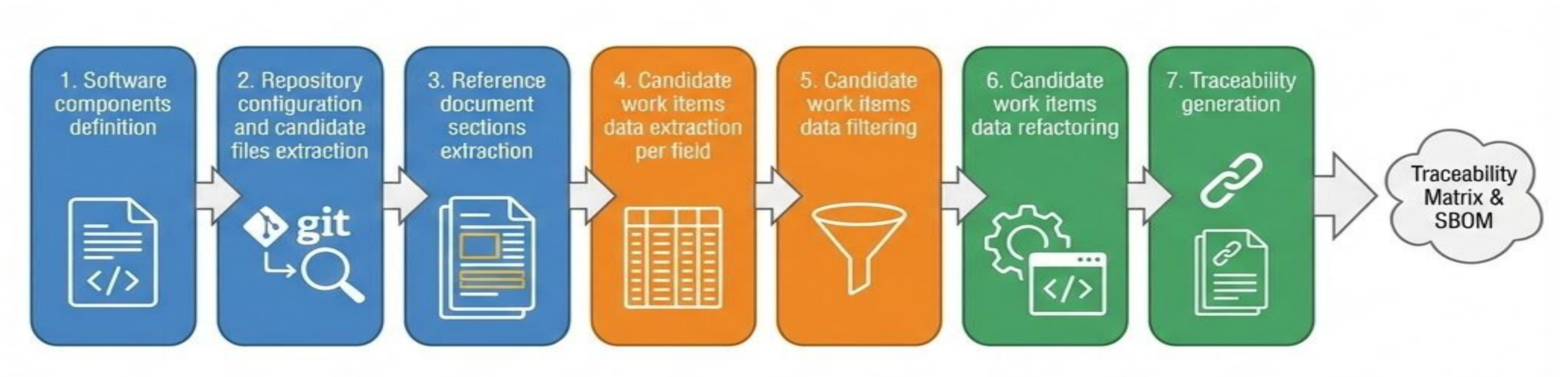
BASIL - Traceability as Code

BASIL is moving traceability definitions in a configuration file

- Work items handled in git repositories
- Scan multiple external git repositories to extract data
- Shareable configuration file generates same results
- Automatically takes into account new work items for evolving branches
- Recreate the traceability of a target git commit
- Not tied to a single work item format
- Can be used in CI as the feature is provided by a command line tool
- Easy to extend with custom rules
- Can be used in CI automating the generation of an SBOM



BASIL - Traceability as Code workflow


1. Software components definition
2. Repository configuration and candidate files extraction
git
3. Reference document sections extraction
4. Candidate work items data extraction per field
5. Candidate work items data filtering
6. Candidate work items data refactoring
7. Traceability generation
Traceability Matrix & SBOM

BASIL - Traceability as Code - Repository files filtering

Define rules to identify the files containing the work items data

- Is targeting a git commit/branch
- Reusable via YAML anchor
- Filter over
  - Files
  - Folders
  - File content

```
repository: &repository_config
  url: "https://github.com/elisa-tech/BASIL.git"
  branch: "main"
  filename_pattern: "*.c"
  folder_pattern: "*examples*"
  hidden: False
  file_contains: ["read_mem"]
  file_not_contains: []
```

BASIL - Traceability as Code - Work item field data extraction

Define how each field should be populated identifying **start** and **end** rules that will be applied over the candidate files.

The match condition can generate list of sections.

Supports relational search with **closest** match in a target **direction**

Allows **split** of a section based on a **delimiter** to generate multiple matches

```
description:
  start:
    line_contains: " read_mem("
  closest:
    line_contains: "Function's expectations:"
      direction: "up"
  end:
    line_contains: "* Context"
  split:
    by: "\n*\n"
```

BASIL - Traceability as Code - Work items candidates extraction

Define rules to extract work items data from identified candidate files:

Rules for each work item fields

- Magic variables
- Relational search
- Support constant values
- List generation from each result

```
software_requirements:
  rules:
    - name: "sr1"
      repository: *repository_config
      skip_top_items: 1
      title:
        value: |
          "Function expectation __software_requirement_index__"
      description:
        start:
          line_contains: " read_mem("
        closest:
          line_contains: "Function's expectations:"
            direction: "up"
        end:
          line_contains: "* Context"
        split:
          by: "\n*\n"
```

BASIL - Traceability as Code - Work items candidates filtering

Define rules to filter over the identified candidate work items:

- contains
- not contains
- regex

```
software_requirements:
  rules:
    - name: "sr1"
      repository: *repository_config
      skip_top_items: 1
      title:
        value: |
          "Function expectation __software_requirement_index__"
        filter:
          contains: ["keep", "1"]
          Case_sensitive: false
```

BASIL - Traceability as Code - Work items candidates refactoring

Define rules to refactor and format work items data:

- replace
- regex substitution
- uppercase
- lowercase
- left trim characters
- right trim characters
- global trim
- prefix
- suffix

```yaml
software_requirements:
  rules:
    - name: "sr1"
      repository: *repository_config
      skip_top_items: 1
      title:
        value: |
          "Function expectation __software_requirement_index__"
        rstrip: “!?)]}.,;:”
        transform:
          - how: replace
            what: what_to_find
            with: replace_with
          - how: prefix
            value: “Requirement: ”
```

BASIL - Traceability as Code - Example

```
/**
* read_mem - read from physical memory (/dev/mem).
* @file: struct file associated with /dev/mem.
* @buf: user-space buffer to copy data to.
* @count: number of bytes to read.
* @ppos: pointer to the current file position, representing the physical
*       address to read from.
*
* This function checks if the requested physical memory range is valid
* and accessible by the user, then it copies data to the input
* user-space buffer up to the requested number of bytes.
*
* Function's expectations:
*
* 1. This function shall check if the value pointed by ppos exceeds the
*    maximum addressable physical address;
*
* 2. This function shall check if the physical address range to be read
*    is valid (i.e. it falls within a memory block and if it can be mapped
*    to the kernel address space);
*
.....
*/
 static ssize_t read_mem(struct file *file, char __user *buf,
                         size_t count, loff_t *ppos)
```

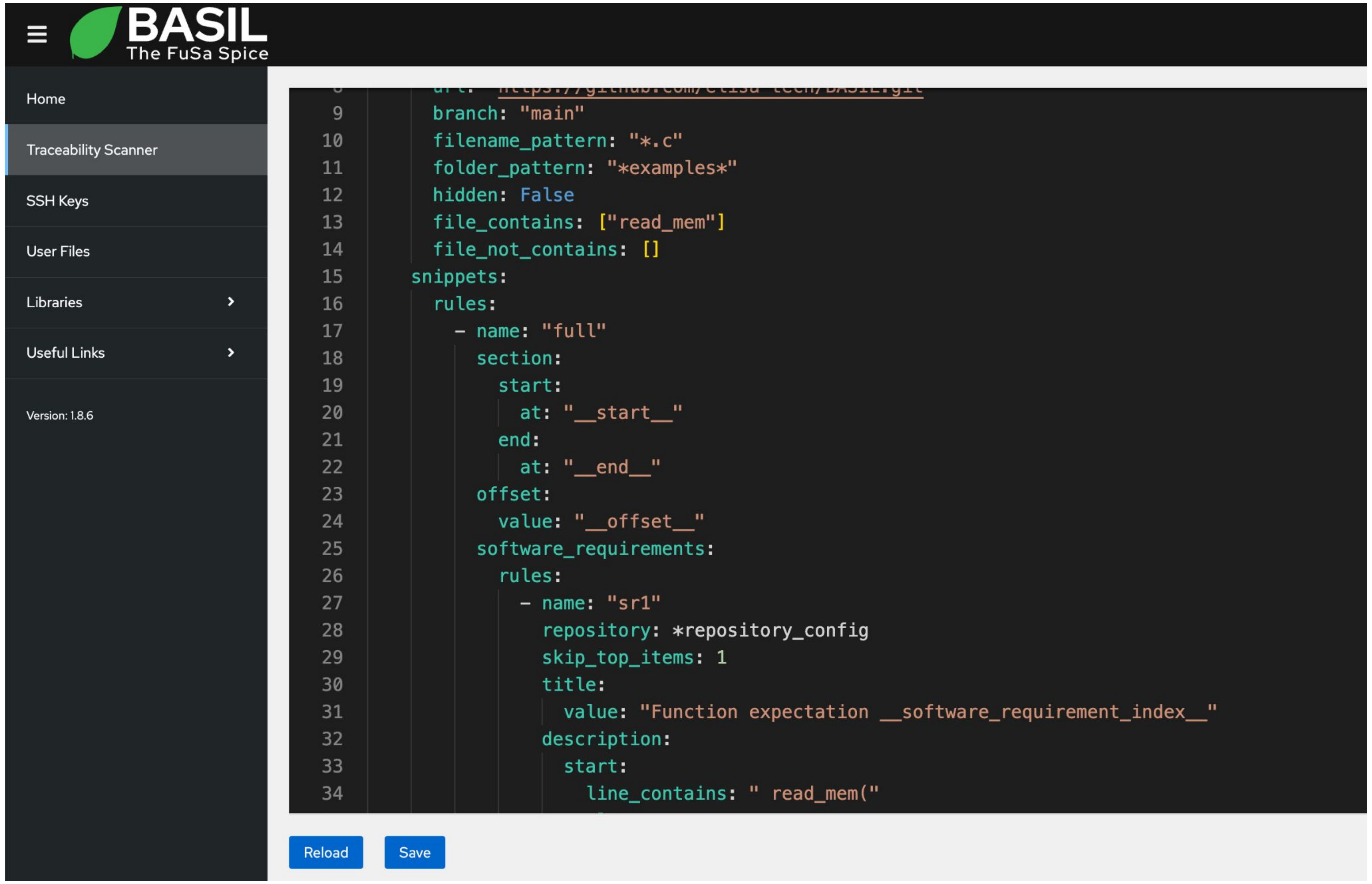
```
software_requirements:
  rules:
  - name: "linux kernel requirements"
    repository: *linux_repo_config
     skip_top_items: 1
      title:
        value: "Function expectation
__software_requirement_index__"
      description:
        start:
          line_contains: " __api__("
          closest:
            line_contains: "Function's expectations:"
            direction: "up"
        end:
          line_contains: "* Context"
        split:
          by: "\n*\n"
        transform:
          - how: "regex_sub"
            what: " +"
            with: " "
          - how: "suffix"
            value: "."
        rstrip: ",.;:!? "
```

BASIL - Traceability as Code - Hierarchy

Work items hierarchy is defined in the configuration file

```
software_requirements:
  rules:
  - name: …
    software_requirements:
      rules:
      - name: …
        test_specification:
          rules:
          - name: …
            test_case:
              rules:
                - name: …
```

BASIL – Traceability as Code – UI



```yaml
  url: "https://github.com/elisa-tech/BASIL.git"
  branch: "main"
  filename_pattern: "*.c"
  folder_pattern: "*examples*"
  hidden: False
  file_contains: ["read_mem"]
  file_not_contains: []
snippets:
  rules:
    - name: "full"
      section:
        start:
          at: "__start__"
        end:
          at: "__end__"
      offset:
        value: "__offset__"
      software_requirements:
        rules:
          - name: "sr1"
            repository: *repository_config
            skip_top_items: 1
            title:
              value: "Function expectation __software_requirement_index__"
            description:
              start:
                line_contains: " read_mem("
```

BASIL - ELISA online instance - Try it out

Create an account using a valid email address

BASIL Admins will approve your request and will

promote your account from GUEST to USER

You will be able to create your Software Components and Work Items

You will be able to:

- allow other people to contribute assigning write permissions
- hide your software components defining read restrictions
- Run test cases with tmt on Fedora VMs

http://elisa-builder-00.iol.unh.edu:9056

東京 2025

LINUX PLUMBERS CONFERENCE

TOKYO, JAPAN / DECEMBER 11-13, 2025