An automated software metadata harvesting pipeline

CLARIAH FAIR Tool Discovery

Maarten van Gompel, KNAW Humanities Cluster

October 7th, 2024

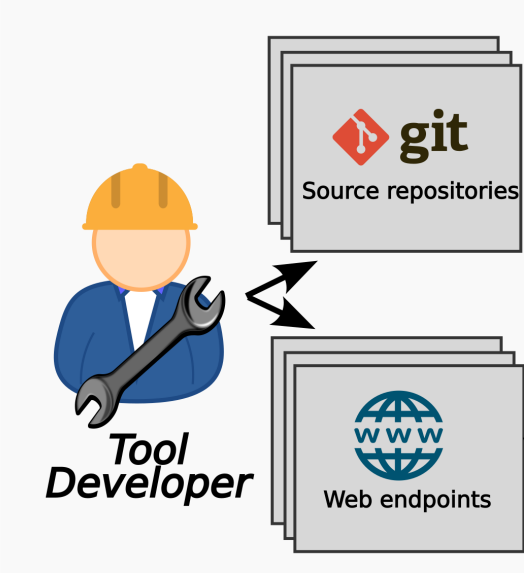
Introduction

- **Objective:** For scholars it is important to find and identify tools suitable for their research.
  - Users must be able to make an informed enough judgement
- **Problem:** Many software catalogues exist, but:
  - Information is often outdated
    - manually curated once, version information, provenance, broken links
  - Information is often incomplete
    - maintenance status, documentation
  - Information is duplicated and not aligned
    - multiple metadata schemes

Introduction (2)

- We developed a pipeline for **automatic** software metadata harvesting.
- We **periodically** harvest software metadata **from the source**
- This is not another software catalogue system; we aim to feed data to various catalogues

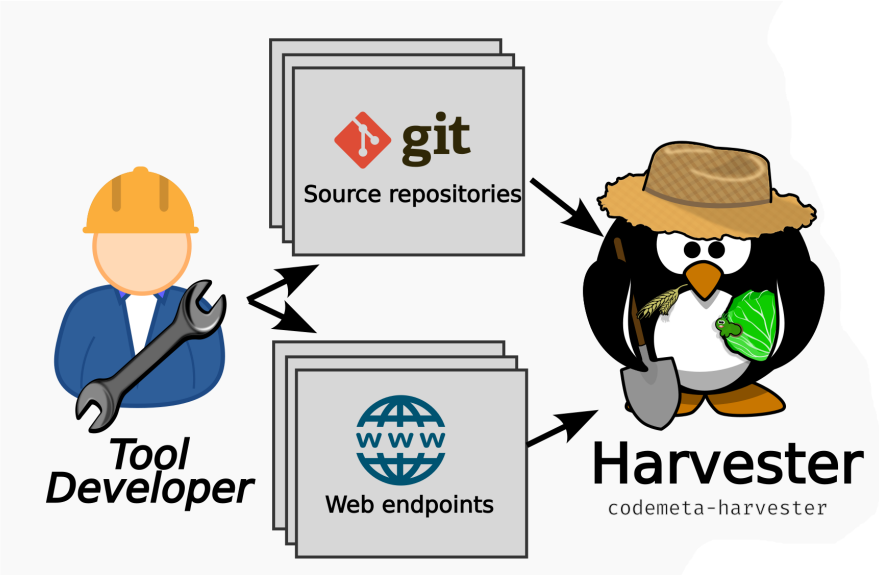
Bottom-up harvesting from the source (1)



- Software metadata is kept at the source
  1. Metadata lives alongside the source code in a version controlled repository (e.g. git) and published in a forge (e.g. GitHub).
  2. (SaaS) metadata provided directly by web endpoints

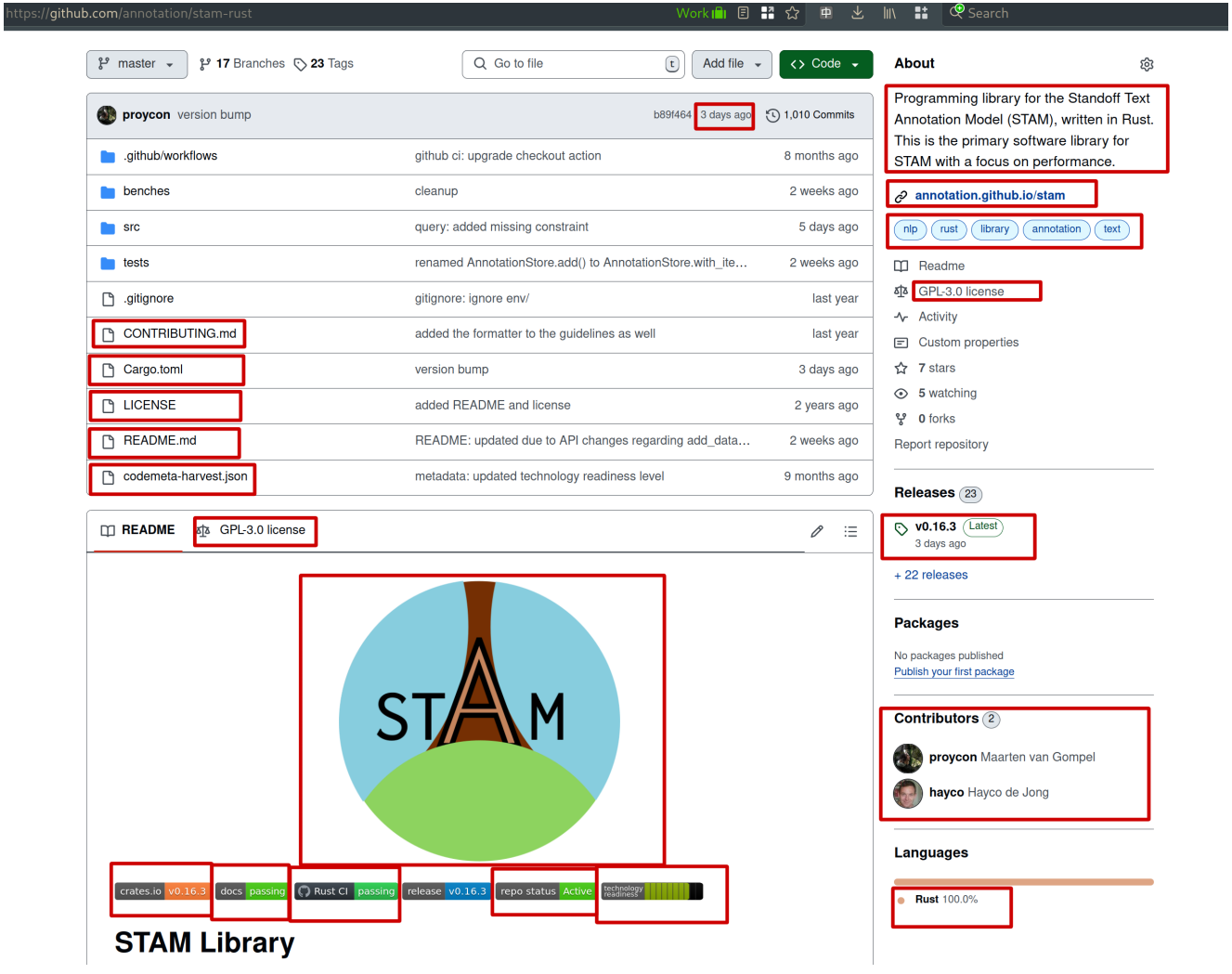
Advantages: Full authorship with the developers, versioned, no duplication, no intermediaries

Bottom-up harvesting from the source (2)



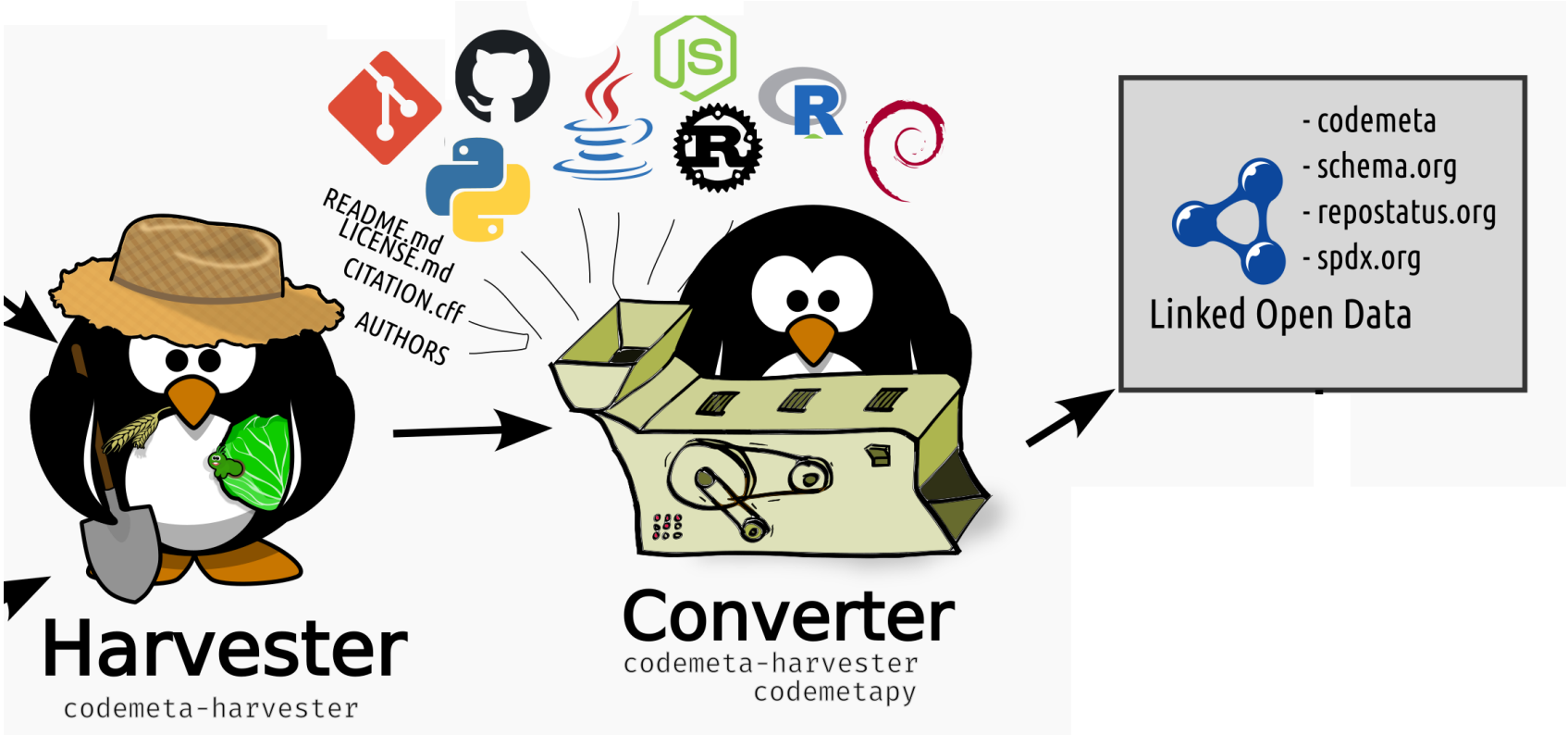
- A minimal manually-curated *source registry* contains all repositories/endpoints to harvest
- Our harvester periodically checks all of these.
- For each, it finds and collects various existing and heterogeneous software metadata schemes
  - Reuse what developers already use, avoid any duplication of metadata.
  - (e.g `pyproject.toml`, `project.json`, `pom.xml`, badges in `README.md`, `LICENSE`, `AUTHORS`)

Bottom-up harvesting from the source (3)



Many instances of metadata can be found!

Metadata conversion to codemeta



- Existing heterogeneous software metadata schemes are converted into a uniform linked-open-data representation
  - CodeMeta and schema.org as our base vocabulary
  - Additional vocabularies on top (e.g. spdx, repostatus.org)
- The result of the conversion is a single `codemeta.json` file (JSON-LD)
  - May be committed back into the source repo

Publishing in catalogues

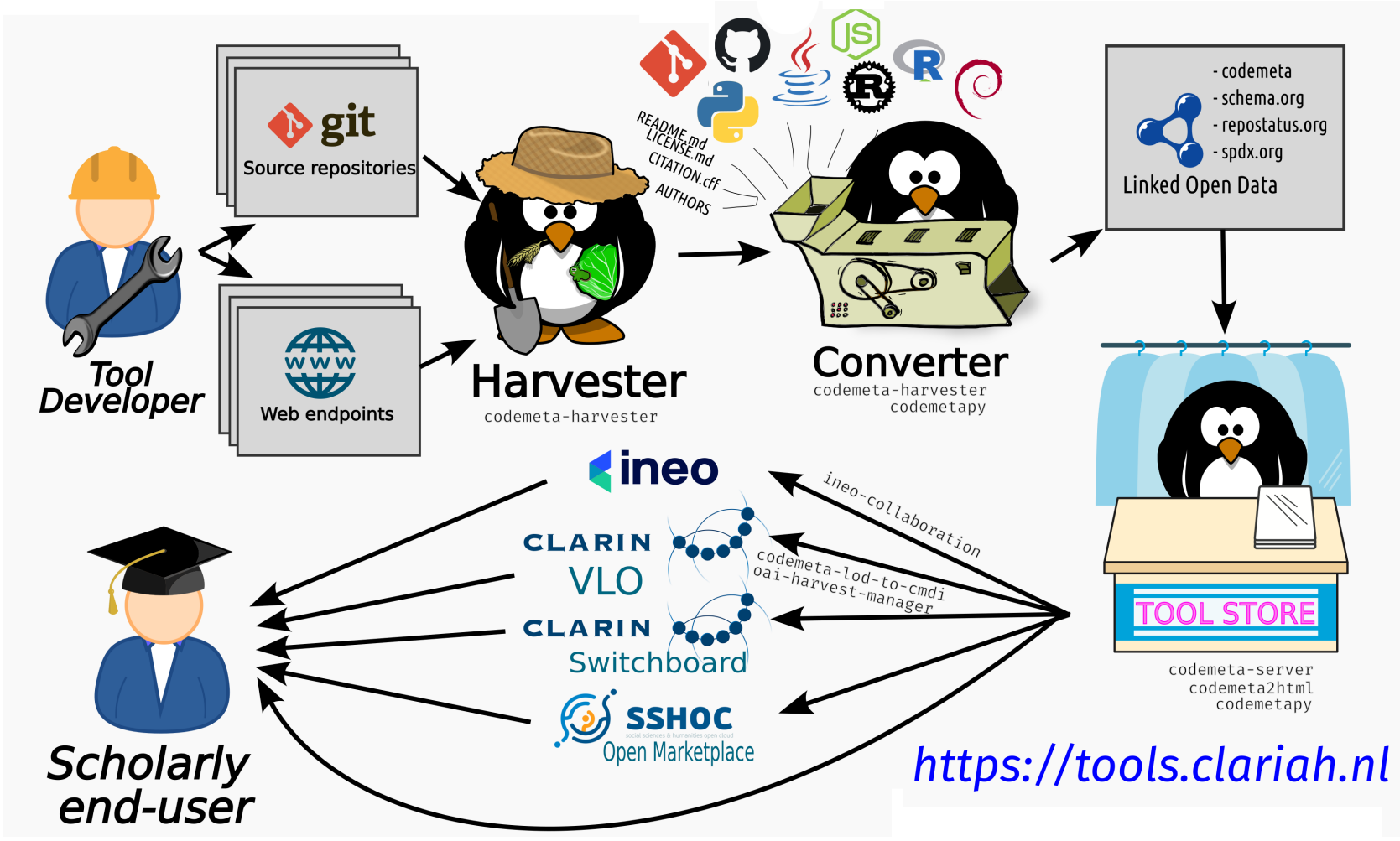


Figure 1: The full architecture

Use-case: CLARIAH (114 source repositories, 34 web endpoints)

Publishing in catalogues (1)

Figure 2: Our own catalogue frontend

Publishing in catalogues (2)

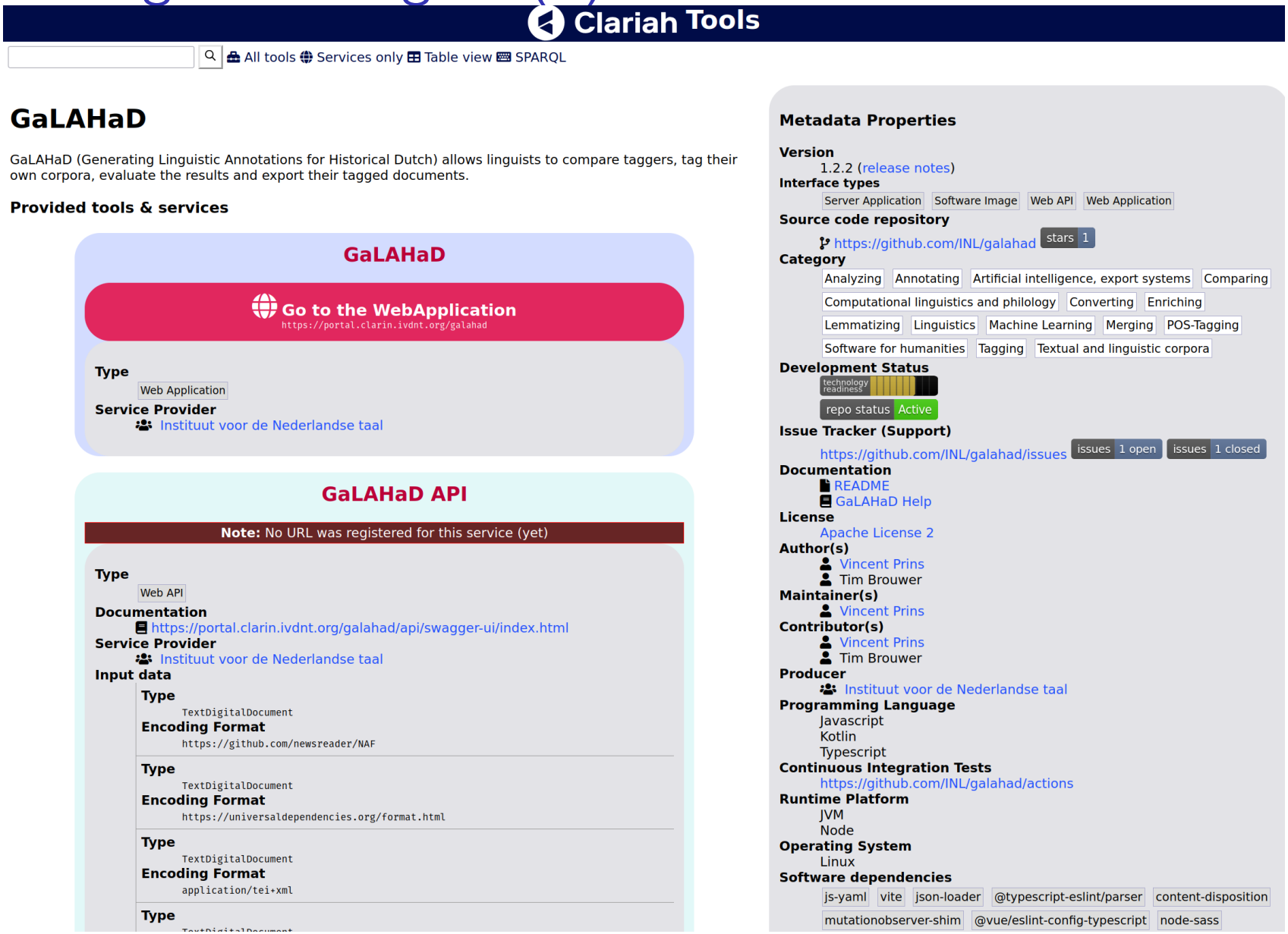


Figure 3: Our own catalogue frontend

Publishing in catalogues (3)

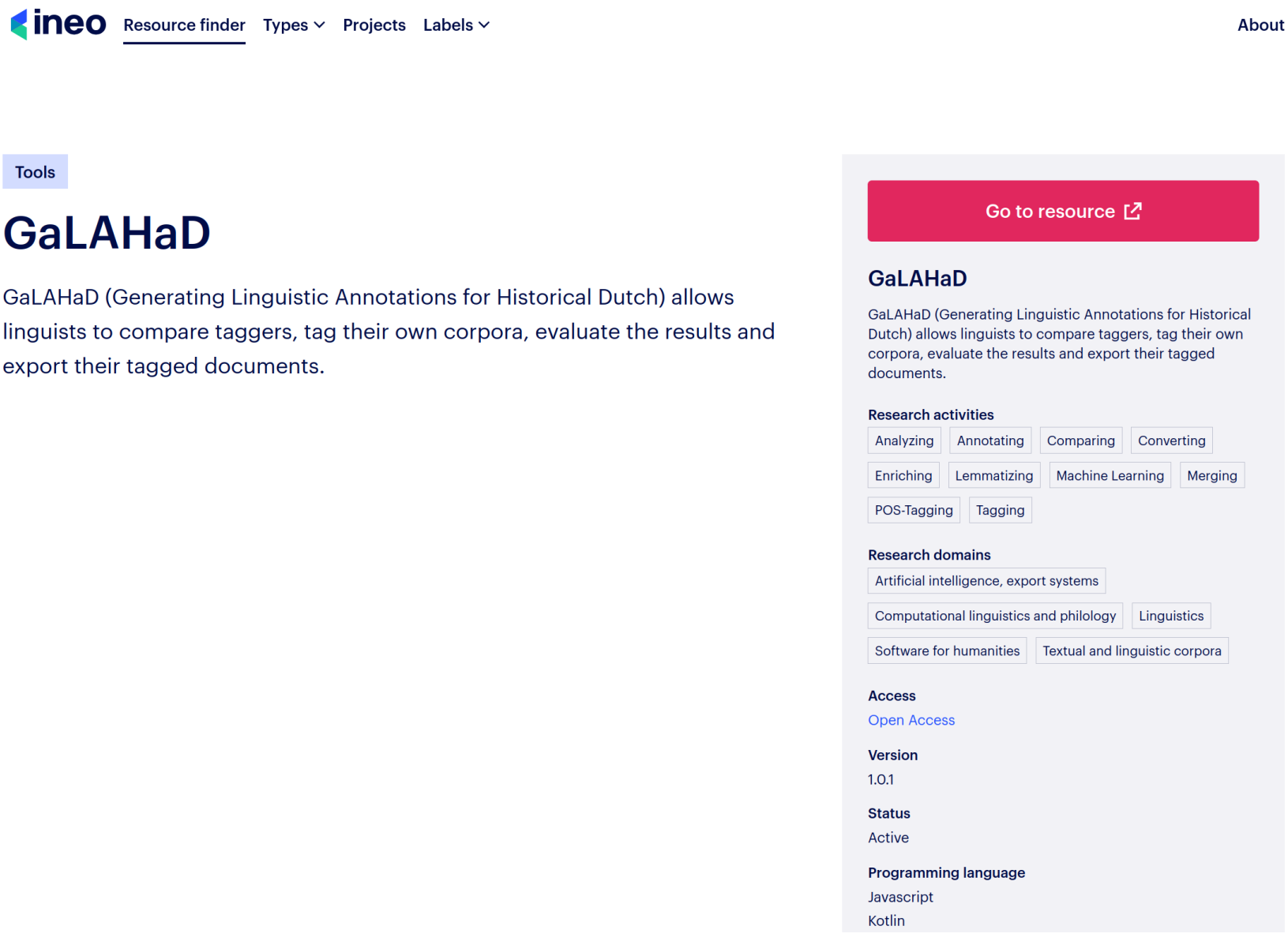


Figure 4: CLARIAH's INEO frontend

Challenges

- Consolidating distinct vocabularies
- Extra domain-specific vocabulary
  - e.g. TaDiRaH for research activities
  - NWO Research Domains
  - Technology Readiness Level
- Validation & Curation

Validation & Curation (1)

- How to allow curation/collaboration?
  - Forges like GitHub have solid contributor mechanisms in-place
- How to assure metadata quality?
  - Harvested metadata is evaluated against a schema (SHACL) that tests metadata fields (completeness, accuracy).
  - The schema is made on the basis of carefully formulated *software metadata requirements*
  - The result of the evaluation is an automated compliance report and a compliance score on a 0-5 scale.
  - The report and score itself become part of the metadata.
  - Set a threshold for propagation to catalogues

Validation & Curation (2): Metadata Requirements

CLARIAH Software Metadata Requirements and Instructions

Introduction

This document specifies requirements and instructions for software metadata for CLARIAH software and services. It is aimed at the developers of software and attempts to provide clear documentation.

The software metadata from all of the sources within CLARIAH is harvested automatically, and periodically, by a harvesting tool. Subsequently, the harvested metadata (codemeta) is made available in the tool store, which offers an API that is in turn used by portals to present the tools.

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in RFC 2119.

Source Code Metadata Requirements

For all of the mentioned properties it in this section it holds that you *MUST NOT* use other vocabularies to exclusively express these properties.

1. Software metadata *MUST* be kept as close to the source code as possible

The main principle underlying the way we keep our software metadata is that **metadata is kept at the source and by the maintainers of the software**. This means that the tool developers themselves are responsible for providing accurate metadata and that the best place to keep this metadata is *with the source code* of the software, in a version-controlled repository.

This is in line with current best-practices as most programming language ecosystems already have their own ways of describing software metadata alongside the source code. Further details on these best-practises and the format of the metadata is expressed in other points below.

This also entails that if a third party wants to correct/revise the metadata, this is done via the tool developers, typically a third part can simply do a pull/merge request on the source repository.

2. All tools *MUST* be registered in the Tool Store Registry

Tools MUST be *registered* with the harvester via the Tool Source Registry. This simple registry tells the harvester where to find, for each tool, both the source code repository *and* any service endpoint(s) where the software is hosted as a service (optional). In order to register your tool, you issue a simple pull request on our git repository as explained in these contribution guidelines.

Validation & Curation (3): Validation report

Name
Automatic software metadata validation report for hypodisc 0.1.0
Author
codemetapy validator using software.ttl
Date
2024-10-07 03:09:26
Review

Please consult the CLARIAH Software Metadata Requirements at https://github.com/CLARIAH/clariah-plus/blob/main/requirements/software-metadata-requirements.md for an in-depth explanation of any found problems

Validation of hypodisc 0.1.0 was successful (score=3/5), but there are some warnings which should be addressed:

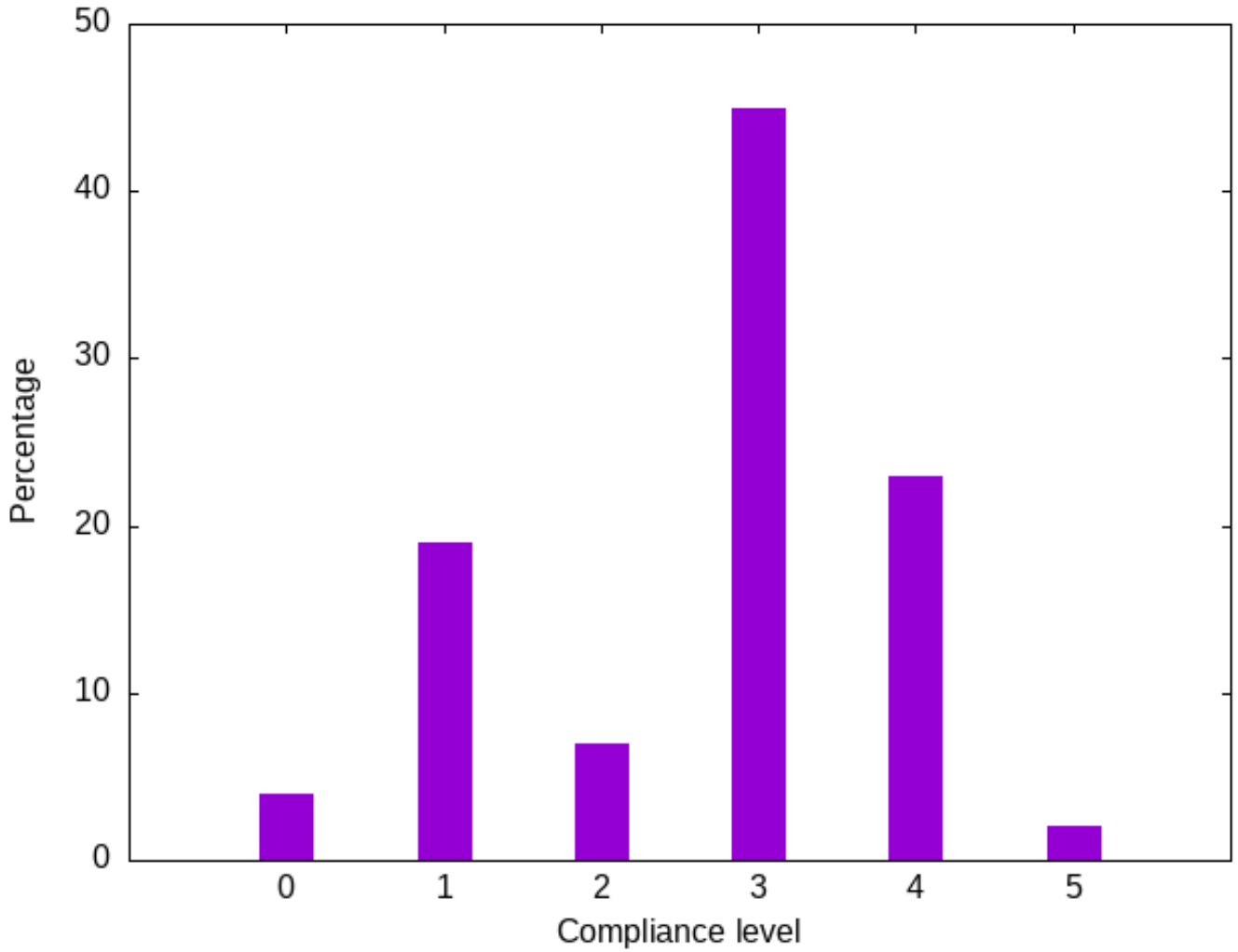
1. Info: Software source code *SHOULD* link to a continuous integration service that builds the software and runs the software's tests (This is missing in the metadata)
2. Info: An interface type *SHOULD* be expressed: Software source code should define one or more target products that are the resulting software applications offering specific interfaces (The metadata does express this currently, but something is wrong in the way it is expressed. Is the type/class valid?)
3. Warning: Documentation *SHOULD* be expressed (This is missing in the metadata)
4. Info: Reference publications *SHOULD* be expressed, if any (This is missing in the metadata)
5. Info: The funder *SHOULD* be acknowledged (This is missing in the metadata)
6. Info: The technology readiness level *SHOULD* be expressed (This is missing in the metadata)

Rating
★ ★ ★ ☆ ☆

Figure 6: An automatically generated validation report

Validation & Curation (4): Findings

- Human compliance is the biggest hurdle
- Hard to get developers to provide extra metadata
- Metadata compliance ranking in CLARIAH (114 tools):



Questions?