FAIR Tool Discovery: an automated software metadata harvesting pipeline for CLARIAH

Maarten van Gompel
KNAW Humanities Cluster
Amsterdam, the Netherlands
proycon@anaproy.nl

Menzo Windhouwer
KNAW Humanities Cluster
Amsterdam, the Netherlands
menzo.windhouwer@di.huc.knaw.nl

Abstract

We present the Tool Discovery pipeline, a core component of the CLARIAH infrastructure in the Netherlands. This pipeline harvests software metadata from the source, detects existing heterogeneous metadata formats already in use by software developers, and converts them to a single uniform representation based on schema.org and codemeta. The resulting data is then made available for further ingestion into other user-facing catalogue/portal systems.

1 Introduction

Software is indispensable in a lot of modern-day research, including in sectors such as the Humanities and Social Sciences that may have traditionally been less focused on information technology. It is also appreciated more and more as valid research output, alongside more conventional output such as academic publications, presentations, and datasets. Scholars often have a need for research software to do their research efficiently.

For scholars it is therefore important to be able to find and identify tools suitable for their research, we call this process *tool discovery*. We define *tool* here and throughout this paper to broadly refer to any kind of software, regardless of the interface it offers and the audience it targets. The scholar's requirement to find tools is reflected in the letter F for *Findable* in the ubiquitous acronym FAIR [1] that has received a lot of attention in recent years in academic circles. The term is often adopted to promote quality and sustainability in research software (Jiménez et al., 2018). In order to find tools, researchers must have access to catalogues that relay *accurate* software metadata.

There is no shortage in existing initiatives in building such catalogues; many research groups, projects or institutes have some kind of website featuring their tools. Aggregation of software metadata from multiple sources is also not new. CLARIN itself already does this in the CLARIN Virtual Language Observatory[2] (van Uytvanck et al., 2010), and DARIAH in the SSHOC Open Marketplace[3] . However, the system we describe in this paper is not an attempt to build another catalogue. We developed a generic pipeline that harvests software metadata from the software's source, leveraging various existing metadata formats, and converting those to a uniform linked open data representation. This data can then be used to feed catalogues.

2 The need for high-quality metadata

Unlike most digital data, software is uniquely characterised as a constantly moving target rather than a static deliverable entity. Releases at different points in time address bugs, security vulnerabilities, or add new features. Moreover, software lives not in isolation, but in connection to other software; its dependencies. Updates are needed to adapt to changes in its runtime environment.

For software metadata to be informative in this dynamic setting, it needs to reflect this moving target and explicitly link to a particular version of the software. This also facilitates provenance keeping and

[1] Findable, Accessible, Interoperable and Reusable

[2] https://vlo.clarin.eu

[3] https://marketplace.sshopencloud.eu/

scientific reproducibility. Furthermore, metadata should convey information about the stage of development the software is in and the level of support an end-user may expect. The user would be wise to exercise caution in adopting software that is unmaintained and unsupported. In practice we often find this information lacking and come across catalogues that were manually compiled once but rarely updated since.

The need for accurate up-to-date metadata goes hand-in-hand with the need for *complete* metadata. If vital details are missing, the end-user may not be able to make an informed judgment.

A common pitfall we have observed in practice is that metadata is often manually collected at some stage and published in a catalogue, but never or rarely updated or revised. In best case, the software has moved on and the metadata covers a mere subset, in worst case, the software or the entire catalogue is unmaintained and out of date.

3 Bottom-up harvesting from the source

What we propose is a *fully automated* pipeline where software metadata is kept at the source, i.e. alongside the software source code, and *periodically* harvested from there. This is in contrast to approaches where metadata primarily resides in an intermediate system that is manually constructed or curated, which is a common approach for many software catalogues[4]. Our approach has a number of important advantages:

1. Source code is often already accompanied by software metadata in existing schemas because many programming language ecosystems already either require or recommend this. Our aim is to avoid any duplication of metadata and *reuse* these existing sources to the maximum extent possible.

   Consider for example `pyproject.toml` or `setup.py` for Python projects, `package.json` for javascript/npm/nodejs projects, `pom.xml` for Java/Maven and `Cargo.toml` for Rust. Aside from these, valuable machine parsable metadata may be extracted from other conventional files such as a `LICENSE` file or a `README.md` file. The latter often contains machine-interpretable badges. Badges are small images often included on top of the README to express certain properties of the software, such as links to documentation, continuous integration services, development status, packaging status. Research software developers also often include a `CITATION.cff`[5] file which we can automatically parse for metadata. All these different sources may be present and can be recombined in our harvesting process.

2. Source code is typically held in a version control system (usually git) and published in forges such as Github, Gitlab, Bitbucket, Codeberg or Sourcehut. This solves versioning issues and ensures metadata can exactly describe the version alongside which it is stored. It also enables the harvester to properly identify the latest stable release, provided some kind of industry-standard versioning system like semantic versioning is adhered to. Software forges themselves may also provide an Application Programming Interface (API) that may serve as an extra source to find software metadata (e.g. descriptions, keywords, links to issue trackers and/or continuous integration services).

3. The developers of the tool have full control and authorship over their metadata. There are no middlemen.

4. Software forges were designed precisely for collaboration on open source software development, so mechanisms for any third party to amend or correct the metadata are already in place (e.g. via a pull/merge request or patch via e-mail). So while developers retain full authorship, this does not mean outside contribution and curation is not possible.

We do not harvest any metadata from intermediaries. By that we mean that we do not use other catalogues as sources (e.g. via the aforementioned OAI-PMH endpoints), only the software source itself.

[4] for example, https://research-software-directory.org/ offers such a platform. Metadata can often be exported via OAI-PMH.
[5] https://citation-file-format.github.io/

Using intermediaries would defeat our philosophy. We do have one extra input source for harvesting: In case the tool in question is Software as a Service (SaaS), i.e. a web-application, web-service, or website, we harvest not only its software source code, but also its web endpoint and attempt to automatically extract metadata from there. We make a clear distinction between the software source code, software instances (executables) you can run locally, and software instances offered as a service via the web. Formally, the software source code has no knowledge when, where, and by whom it may be deployed, neither locally on some user's computer nor as a service on some server. This link is therefore established at an independent and higher level. In the resulting metadata, there will be an explicit link between the source code and *applications* of that source code[6]. The sources for harvesting source repositories and web endpoints (both effectively just URLs) are the only input that needs to be manually provided to our harvesting system, we call this the *source registry*. This is the higher level we referred to earlier. We keep the source registry in a simple git repository containing very minimalistic configuration files (one yaml file per tool). This is also the only point in our pipeline where there is the possibility for a human curator to decide whether or not to include a tool.

Usage of such a manually curated source registry means that, for this project, automatic discovery of tools is not in scope. That is, we do not actively crawl the web in search for tools that might or might not fit a certain domain. Some interpret the term 'tool discovery' to also include such functionality, but we do not. Such a step, however, can be envisioned as a separate step prior to execution of our pipeline.

4 A unified vocabulary for software metadata

The challenge we are facing is primarily one of mapping multiple heterogeneous sources of software metadata to a unified vocabulary. Fortunately, this is an area that has been explored previously in the CodeMeta project[7]. They developed a generic vocabulary for describing software source code, extending schema.org vocabulary and contributing their efforts back to them. Moreover, the CodeMeta project defines mappings, which they call *crosswalks*, between their vocabulary and many existing metadata schemes, such as those used in particular programming languages ecosystems or by particular package managers.

Schema.org and CodeMeta are both linked open data (LOD) vocabularies[8], and codemeta is canonically serialised to a JSON-LD[9] file which makes it easily parsable for both machine and human alike. This `codemeta.json` file can be kept under version control alongside a tool's source code. A rather minimal example of a such a file is shown below:

```
{
    "@context": [
        "https://w3id.org/codemeta/3.0",
        "http://schema.org",
    ],
    "@id": "https://example.org/mysoftware",
    "@type": "SoftwareSourceCode",
    "identifier": "mysoftware",
    "name": "My Software",
    "author": {
        "@type": "Person",
        "givenName": "John",
        "familyName": "Doe"
    },
    "description": "My software does nice stuff",
    "codeRepository": "https://github.com/someuser/mysoftware",
    "license": "https://spdx.org/licenses/GPL-3.0-only",
    "developmentStatus": "https://www.repostatus.org/#active",
    "thumbnailUrl": "https://example.org/thumbnail.jpg"
}
```

[6] Applications are instances of the source-code in executable form after build and deployment and may also refer to availability as a service over a network

[7] https://codemeta.github.io

[8] i.e. building upon RDF and being retrievable over HTTP

[9] https://www.w3.org/TR/json-ld/

The developer has a choice to either run our harvester and converter themselves and commit the resulting codemeta file, or to not add anything and let the harvester dynamically reconstruct the metadata every harvest cycle.

The convention to add a `codemeta.json` file alongside the source code was established by the CodeMeta project. In addition to this, we define another method that is specific for our metadata harvester: Developers can add a `codemeta-harvest.json` file instead of `codemeta.json`. Whereas `codemeta.json` by definition contains the complete metadata, the `codemeta-harvest.json` file contains an arbitrary subset and is used to supplement any automatically harvested metadata. This allows developers to rely on the harvester for most fields, without having to run it themselves, but still allows them to provide additional manual metadata. All these different options ensure that developers themselves can choose precisely how much control to exert over the metadata and harvester. It allows us to accommodate both projects that aren't even aware they're being harvested, as well as projects that want to fine-tune every metadata field to their liking, effectively rendering most of our periodic harvester out of work at run-time.

4.1 Additional Vocabularies

We link to various other linked open data vocabularies, listed below. Most of these are formulated as SKOS[10] vocabularies.

- **repostatus.org**[11] – *Development Status* – The repostatus.org project allows developers to express usability and development/support status of a project. A LOD (SKOS) version of this vocabulary was developed in the scope of this project and contributed back to the repostatus project.
- **SPDX**[12] – *Open-source software licenses* – The Software Package Data Exchange project is a Linux Foundation project that defines, amongst other things, open source software licenses. It is widely used, e.g. by package managers.
- **TaDiRaH**[13] – *Research activities* – The TaDiRAH vocabulary "classifies and categorizes the activities that comprise digital humanities" (Borek et al., 2016) with the aim to help scholars group and identify projects that share certain commonalities. We adopt this vocabulary to describe the research activities a software tool can be used for. Example of some top-level categories in this vocabulary are 'Analyzing', 'Capturing', 'Creating', 'Enriching'. An example of a deeper-level category that is particularly common in language resources such as those seen in CLARIN is for example 'Enriching → Annotating → Named Entity Recognition'.
- **NWO Research Domains**[14] – NWO is the Dutch Research Council. They define several research fields that are used for official grant applications. As CLARIAH is a Dutch project, we use this vocabulary to express the research domain a tool is used for. A LOD (SKOS) version of this vocabulary was developed in the scope of this project.

The first two vocabularies are generic enough to be applicable to almost all software projects, we strongly recommend their usage. The latter two may be more constrained to research software as developed in CLARIAH and CLARIN. In your projects, you can adopt whatever you find suits your needs best, the power to mix and match is at the heart of linked open data after all.

Moreover, we formulated some of our own extensions on top of codemeta and schema.org:

- **Software Types**[15] – Software comes in many shapes and forms, targeting a variety of audiences with different skills and needs. We want software metadata to be able to accurately

[10] https://www.w3.org/TR/skos-reference/
[11] https://repostatus.org
[12] https://spdx.dev
[13] https://vocabs.dariah.eu/tadirah/
[14] https://www.nwo.nl/en/nwo-research-fields
[15] https://github.com/SoftwareUnderstanding/software_types

express what type(s) of interface their software provides. The schema.org vocabulary distinguishes `softwareApplication`, `WebApplication`, `MobileApplication` and even `VideoGame`. This covers some interfaces from a user-perspective, but is not as extensive nor as fine-grained as we would like yet. Interface types from a more developer-oriented perspective are not formulated. We therefore define additional classes such as `DesktopApplication` (software offering a desktop GUI), `CommandLineApplication`, `SoftwareLibrary` and others in this add-on vocabulary.

- **Software Input/Output Data**[16] – This minimal vocabulary defines just two new properties that allows for software metadata to express what kind of data it consumes (e.g. takes as input) and what kind of data it produces (e.g. output). It does not define actual data types because schema.org already has classes covering most common data types (e.g. `AudioObject`, `ImageObject`, `VideoObject`, `TextDigitalDocument`, etc...) and properties like `encodingFormat` to tie these to MIME-types or `inLanguage` to tie it to natural languages.

  Do note that describing a full API is explicitly out of scope for our project. A full API description would describe exactly which function or web-endpoints take and return what data. Although this too can be considered a type of metadata, such functionality goes beyond what we consider the primary software metadata which end-users need to make a informed decision regarding the suitability of a tool for their ends. Other existing projects such as the OpenAPI Initiative[17] delve into this realm for Web APIs. For software libraries there are various existing API documentation generators[18] that derive documentation directly from the source code in a formalised way. We do not intend to duplicate those efforts.

5 Architecture

The full architecture of our pipeline is illustrated schematically in Figure 1. Although we demonstrate this in the context of the CLARIAH project, the underlying technology is generic and can also be used for other projects.

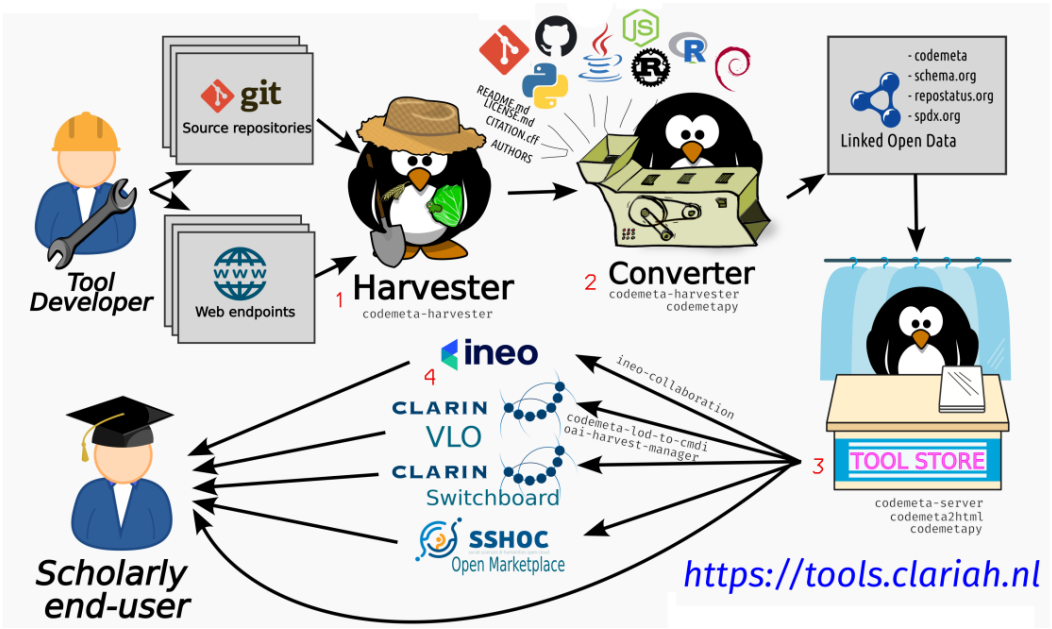

Figure 1: The architecture of the CLARIAH Tool Discovery pipeline. Key steps are numbered in red and referenced in the text.

[16] https://github.com/SoftwareUnderstanding/software-iodata
[17] https://www.openapis.org
[18] e.g. doxygen, sphinx, rustdoc, etc...

Using the input from the source registry, our *harvester*[19] (1) (van Gompel et al., 2024) fetches all the git repositories and queries any service endpoints. It does so at regular intervals (e.g. once a day). This ensures the metadata is always up to date. When the sources are retrieved, it looks for different kinds of metadata it can identify there and calls the converter (2) powered by codemetapy[20] (van Gompel, 2024) to turn and combine these into a single codemeta representation. This produces one codemeta JSON-LD file per input tool.

All of these together are loaded in our *tool store* (3), powered by codemeta-server[21] (van Gompel, 2023) and codemeta2html[22]. This is implemented as an RDF triple store and serves both as a backend to be queried programmatically using SPARQL, as well as a simple web frontend to be visited by human end-users as a catalogue. The frontend for CLARIAH is accessible as a service at https://tools.clariah.nl and shown in Figures 2 and 3. At the time of writing, there are 114 registered source repositories and 34 web endpoints.

Figure 2: Screenshot of the CLARIAH Tool Store showing the index page

5.1 Propagation to Software Catalogues

Our web front-end is not the final destination; our aim is to propagate the metadata we have collected to other existing portal/catalogue systems (4), such as the CLARIN VLO, the CLARIN Switchboard, the SSH Open Marketplace, and CLARIAH's Ineo[23]. The latter has already been implemented, the VLO export will be done via a conversion from codemeta to CMDI, and the Marketplace conversion has started in collaboration with DARIAH.

Propagation of software metadata can be visualised as a simple input/output process where the input side connects to our tool store, either via our SPARQL endpoint or by simply obtaining the entire (or

[19] https://github.com/proycon/codemeta-harvester, marked with a red 1 in Figure 1
[20] https://github.com/proycon/codemetapy
[21] https://github.com/proycon/codemeta-server
[22] https://github.com/proycon/codemeta2html
[23] https://vlo.clarin.eu/, https://switchboard.clarin.eu/, https://marketplace.sshopencloud.eu/, https://ineo.tools

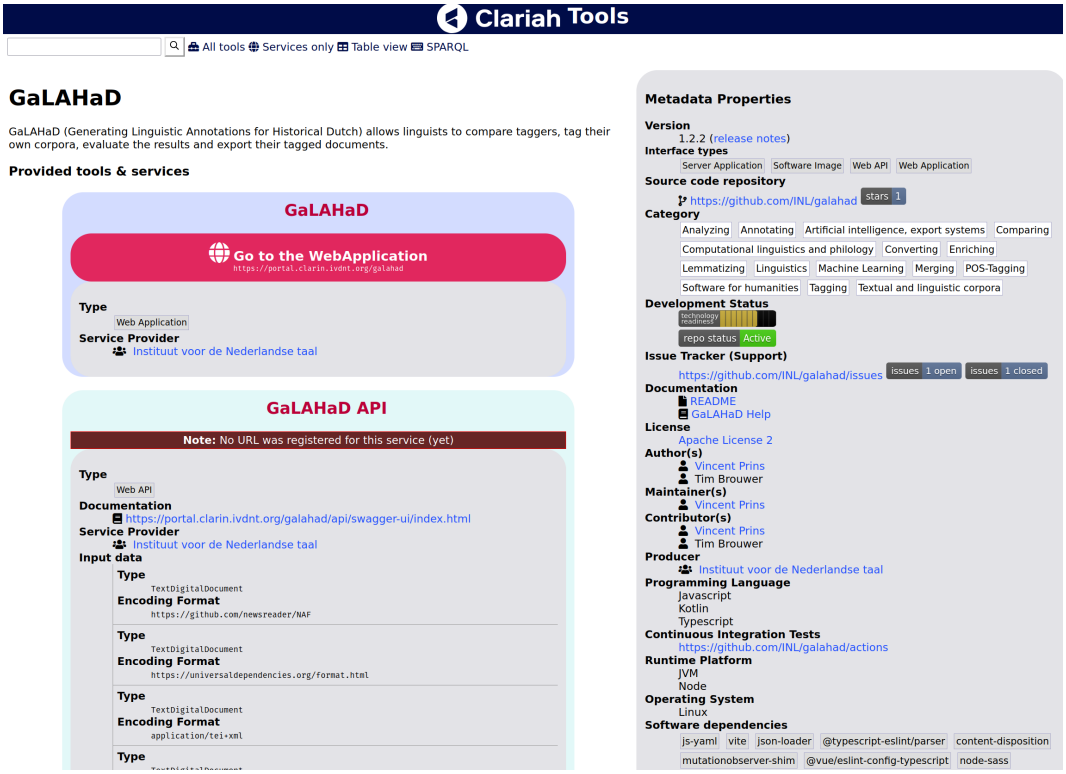

Figure 3: Screenshot of the CLARIAH Tool Store showing the metadata page for a specific tool

a specific part of the) graph in JSON-LD. This may be a periodic query or even a real-time query. The output side connects to either a catalogue-specific API or directly to some kind of database underlying the catalogue system. The process itself consists of conversion from our codemeta representation to whatever representation is suited for the catalogue system.

The connection to the SSHOC Open Marketplace is still ongoing work[24]. For this conversion, we load the JSON-LD graph into an in-memory triple store, iterate over specific triples, and then perform API calls to the SSHOC Open Marketplace API.

In the case of CLARIN's VLO the codemeta schema has been translated into a CMDI profile (Windhouwer & Goosen, 2022). The VLO's harvester has been extended to, next to the traditional OAI protocol, allow other "protocols" to be plugged in[25]. In this case the plugin takes the JSON-LD dump from the tool store and converts the records to equivalent CMDI records compliant with the profile. The changes we in CLARIAH made to the VLO's harvester are currently being tested by CLARIN. Once a new stable version of this harvester has been released the harvesting cycle of CLARIN will be extended to harvest the metadata.

Finally, CLARIAH's Ineo also has a harvesting cycle, which transforms the JSON-LD records into the JSON expected by Ineo's update API. These transformations are minimal as the tool metadata has been designed with this target catalogue in mind.

6 Validation & Curation

Having an automated metadata harvesting pipeline may raise some concerns regarding quality assurance. Data is automatically converted from heterogeneous sources and immediately propagated to our tool store, this is not without error. In absence of human curation, which is explicitly out of our intended scope, we tackle this issue through an automatic validation mechanism. This mechanism provides feedback for the developers or curators.

[24] An initial prototype can be found at https://github.com/proycon/codemeta2mp

[25] https://github.com/clarin-eric/oai-harvest-manager, https://github.com/CLARIAH/oai-harvest-manager

The harvested codemeta metadata is held against a validation schema (SHACL) that tests whether certain fields are present (completeness), and whether the values are sensible (accuracy; it is capable of detecting various discrepancies). The validation process outputs a human-readable validation report which references a set of carefully formulated *software metadata requirements* [26]. These requirements state exactly what kind of metadata we expect for software in the CLARIAH project, using normative keywords such as MUST, SHOULD and MAY in accordance with RFC2119 (Bradner, 1997). These requirements provide instructions to developers about how they can provide this metadata in their `codemeta.json` or `codemeta-harvest.json` if metadata can not be automatically extracted from existing sources. The validation schema and requirements document are specific to the CLARIAH project, but may serve as an example for others to adapt and adopt. An example of a validation report referencing the metadata requirements is shown in Figure 4.

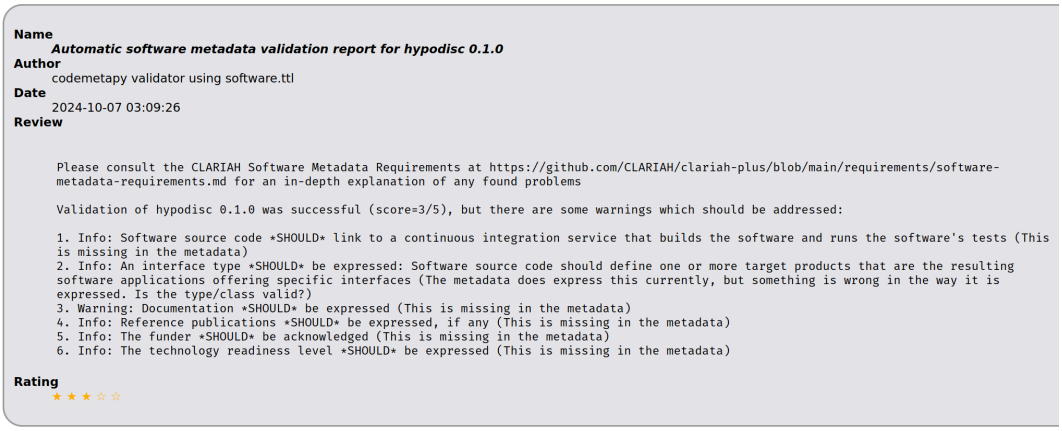

Figure 4: Screenshot of a validation report for a particular tool, viewed from the tool store

Using this report, developers can clearly identify what specific requirements they have not met. The over-all level of compliance is expressed on a simple scale of 0 (no compliance) to 5 (perfect compliance), and visualised as a coloured star rating in our interface. This evaluation score and the validation report itself becomes part of the delivered metadata and is something which both end users as well as other systems can filter on. It may even serve as a kind of 'gamification' element to spur on developers to provide higher quality metadata.

We find that human compliance remains the biggest hurdle and it is hard to get developers to provide metadata beyond what we can extract automatically from their existing sources. The metadata compliance rankings for CLARIAH are shown in Figure 5.

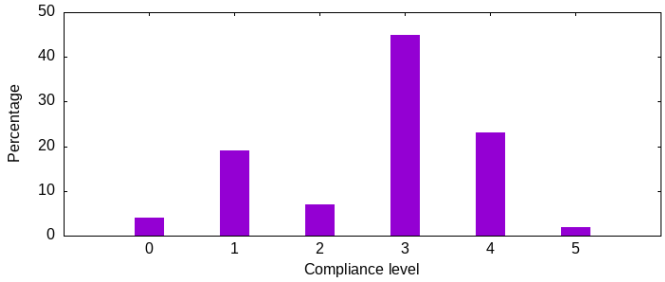

Figure 5: Histogram of metadata compliance ranking in CLARIAH (114 tools), the ranking is to the number of stars given (0 to 5, where 5 is perfect compliance)

For propagation to systems further downstream, we set a threshold rating of 3 or higher. Downstream systems may of course posit whatever criteria they want for inclusion, and may add human validation

[26] https://github.com/CLARIAH/clariah-plus/blob/main/requirements/software-metadata-requirements.md.

and curation. As metadata is stored at the source, however, we strongly recommend any curation efforts to be directly contributed back upstream to the source, through the established mechanisms in place by whatever forge (e.g. GitHub) they are using to store their source code.

7 Discussion & Related Work

We limit automatic metadata extraction to those fields and sources that we can extract fairly reliably and unambiguously. In certain cases, it is already a sufficient challenge to map certain existing vocabularies onto codemeta and schema.org, as concepts are not always used in the same manner and do not always map one-to-one.

We do extract certain information from README files, but that is mostly limited to badges which follow a very standard pattern that is easy to extract with simple regular expressions. Extracting more data from READMEs is something that was done in Kelley and Garijo, 2021 and predecessor Mao et al., 2019; they analyse the actual README text and extract metadata from it. They use various methods to do so, including building supervised classifiers to identify common section headers and mapping those to a metadata category such as 'description', 'installation', 'license', etc.... Their classifiers, however, only produced adequate results for four common categories, so they diverted to alternative methods such as exploration/detection of other files (like `LICENSE`), using regular expressions to capture badges, and calling APIs like GitHub's. All of those techniques we have implemented as well in our harvesting pipeline. In line with their findings, we did not expect much from supervised classification (measured against the effort that goes into labelling data) so did not pursue that.

With the advent of Large Language Models in recent years, we can also envision these playing a role in metadata extraction. We would, however, caution restraint here as their innate nature to hallucinate and lack of transparency is at odds with the objective to extract accurate metadata. Our extraction pipeline focusses on using relatively simple techniques to quickly get high precision results and on re-using already existing metadata schemes. We do think it is good practise to have developers manually provide metadata, we just want to ensure they only need to do it once alongside their own source-code, using schemas they use anyway, and not duplicate the effort for every software catalogue or package manager.

We also want to draw a quick line of comparison with the Research Software Directory (Cahen et al., 2024; Spaaks, 2018). This is an open-source content management system for software catalogues, so a different beast than our metadata extraction pipeline. They do, however, offer some integrations with third party services such as GitHub, Zenodo, ORCID, etc... to automatically extract or autocomplete certain metadata. It illustrates there are hybrid approaches possible where a content management system is available for human metadata curation, but with key parts automated to reduce both the human workload as well as the common pitfalls we addressed in section 2.

8 Conclusion & Future Work

We have shown a way to store metadata at the source and reuse existing metadata sources, recombining and converting these into a single unified LOD representation using largely established vocabularies. We developed tooling for codemeta that is generically reusable and available as free open source software[27]. We hope that our pipeline results in metadata that is accurate and complete enough for scholars to assess whether certain software is worth exploring for their research. We think this is a viable solution against metadata or entire catalogues going stale, in worst case unbeknownst to the researcher who might still rely on them. Quality assurance can be addressed, in part, via automated validations against carefully formulated validation rules. Furthermore, we also showed that the metadata we collect can be propagated to other downstream software catalogue systems.

Future work will focus on keeping in sync with vocabulary developments in CodeMeta and schema.org, as well as on working on the automatic propagation of harvested metadata into catalogue systems such as the SSHOC Open Marketplace.

[27] GNU General Public Licence v3

Acknowledgements

The FAIR Tool Discovery track has been developed as part of the CLARIAH-PLUS project (NWO grant 184.034.023), as part of the Shared Development Roadmap.

Connectivity to the SSHOC Open Marketplace is being continued in the SSHOC-NL project as part of Task 1.1.

References

Borek, L., Dombrowski, Q., Perkins, J., & Schöch, C. (2016). Tadirah: A case study in pragmatic classification. *Digit. Humanit. Q.*, *10*(1). http://dblp.uni-trier.de/db/journals/dhq/dhq10.html#BorekDPS16

Bradner, S. (1997). *IETF RFC 2119: Key words for use in RFCs to Indicate Requirement Levels*. http://www.ietf.org/rfc/rfc2119.txt

Cahen, E. J., Mijatovic, D., Garcia Gonzalez, J., Maassen, J., Jong, M., Meeßen, C., Rüster, M., Hanisch, M., & Ziegner, N. (2024). *Research Software Directory (as a service)*. Zenodo. https://doi.org/10.5281/ZENODO.14243099

Jiménez, R. C., Kuzak, M., Alhamdoosh, M., Barker, M., Batut, B., Borg, M., Capella-Gutiérrez, S., Hong, N. P. C., Cook, M., Corpas, M., Flannery, M., García, L. J., Gelpi, J. L., Gladman, S. L., Goble, C. A., Ferreiro, M. G., González-Beltrán, A. N., Griffin, P., Grüning, B. A., … Crouch, S. (2018). Four simple recommendations to encourage best practices in research software. https://api.semanticscholar.org/CorpusID:214915242

Kelley, A., & Garijo, D. (2021). A Framework for Creating Knowledge Graphs of Scientific Software Metadata. *Quantitative Science Studies*, 1–37. https://doi.org/10.1162/qss_a_00167

Mao, A., Garijo, D., & Fakhraei, S. (2019). SoMEF: A Framework for Capturing Scientific Software Metadata from its Documentation. *2019 IEEE International Conference on Big Data (Big Data)*, 3032–3037. https://doi.org/10.1109/BigData47090.2019.9006447

Spaaks, J. (2018). *The Research Software Directory and how it promotes software citation* [Accessed: 2025-01]. https://blog.esciencecenter.nl/the-research-software-directory-and-how-it-promotes-software-citation-4bd2137a6b8

van Gompel, M. (2023). *codemeta-server* (Version 0.4.1). Zenodo. https://doi.org/10.5281/zenodo.10204020

van Gompel, M. (2024). *codemetapy* (Version 2.5.3). Zenodo. https://doi.org/10.5281/zenodo.11656553

van Gompel, M., de Boer, D., & Broeder, J. (2024). *codemeta-harvester* (Version 0.4.0). Zenodo. https://doi.org/10.5281/zenodo.11472618

van Uytvanck, D., Zinn, C., Broeder, D., Wittenburg, P., & Gardellini, M. (2010). Virtual language observatory: The portal to the language resources and technology universe. In N. Calzolari, K. Choukri, B. Maegaard, J. Mariani, J. Odijk, S. Piperidis, M. Rosner, & D. Tapias (Eds.), *Lrec*. European Language Resources Association. http://dblp.uni-trier.de/db/conf/lrec/lrec2010.html#UytvanckZBWG10

Windhouwer, M., & Goosen, T. (2022). Component metadata infrastructure. In D. Fišer & A. Witt (Eds.), *Clarin: The infrastructure for language resources* (pp. 191–222). De Gruyter. https://doi.org/doi:10.1515/9783110767377