Towards FAIR Research Software

Daniel Garijo, Ontology Engineering Group, Universidad Politécnica de Madrid, Spain

Assessing Best Research Software Practices through Metadata

daniel.garijo@upm.es
@dgarijov

Research Software is one of the pillars of Open Science

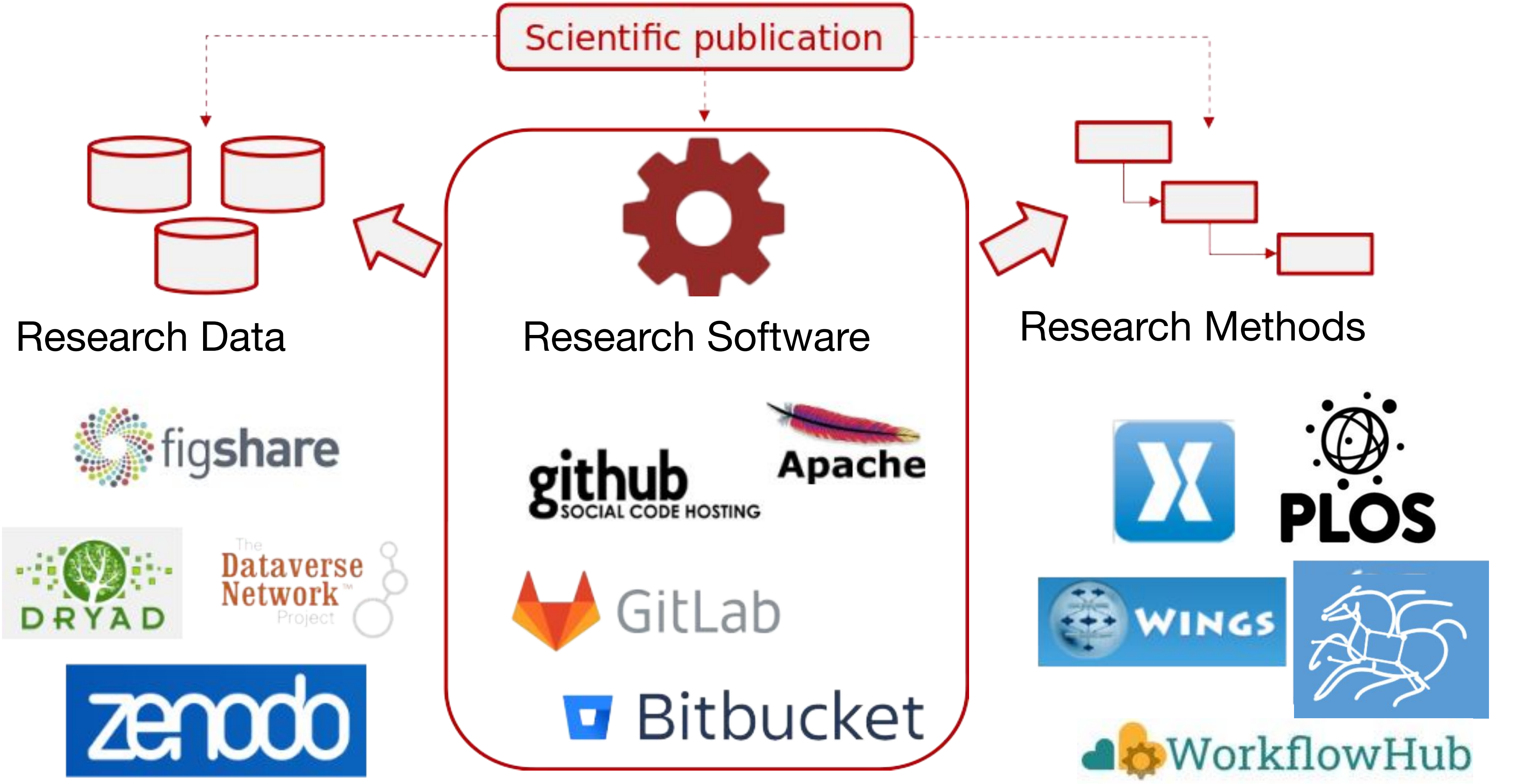
Research Software is one of the pillars of Open Science

Research Data

Research Software

Research Methods

Research Software is one of the pillars of Open Science



The rise of the FAIR Data principles

Findable Accessible Interoperable Reusable

Data (initially) [1]

Research Software

Methods

Semantic artefacts

Other guidelines:

- Guidelines for Transparency and Openness Promotion (TOP) [2]
- Reproducibility Enhancement Principles (REP) [3]
- ...

RESEARCH DATA ALLIANCE

FORCE11 The Future of Research Communications and e-Scholarship

[1] Wilkinson, M., Dumontier, M., Aalbersberg, I. *et al.* The FAIR Guiding Principles for scientific data management and stewardship. *Sci Data* **3,** 160018 (2016). https://doi.org/10.1038/sdata.2016.18
[2] https://www.cos.io/initiatives/top-guidelines
[3] Stodden, V et al Enhancing reproducibility for computational methods https://www.science.org/lookup/doi/10.1126/science.aah6168

F
1. Create a description of your software
2. Register your software in a software registry
3. Use a Unique and Persistent Identifier for your software

A
4. Make sure that people can download your software

I
5. Explain the functionality of your software
6. Use standard (community agreed) formats for inputs and outputs

R
7. Document your software
8. Give your software a licence
9. State how to cite your software
10. Follow best practices for software development

https://github.com/LibraryCarpentry/Top-10-FAIR

F: Software, and its associated metadata, is easy for both humans and machines to find.

F1. Software is assigned a globally unique and persistent identifier.
- F1.1. Components of the software representing levels of granularity are assigned distinct identifiers.
- F1.2. Different versions of the software are assigned distinct identifiers.

F2. Software is described with rich metadata.
F3. Metadata clearly and explicitly include the identifier of the software they describe.
F4. Metadata are FAIR, searchable and indexable.

A: Software, and its metadata, is retrievable via standardized protocols.

A1. Software is retrievable by its identifier using a standardized communications protocol.
- A1.1. The protocol is open, free, and universally implementable.
- A1.2. The protocol allows for an authentication and authorization procedure, where necessary.

A2. Metadata are accessible, even when the software is no longer available.

I: Software interoperates with other software by exchanging data and/or metadata, and/or through interaction via application programming interfaces (APIs), described through standards.

I1. Software reads, writes and exchanges data in a way that meets domain-relevant community standards.
I2. Software includes qualified references to other objects.

R: Software is both usable (can be executed) and reusable (can be understood, modified, built upon, or incorporated into other software).

R1. Software is described with a plurality of accurate and relevant attributes.
- R1.1. Software is given a clear and accessible license.
- R1.2. Software is associated with detailed provenance.

R2. Software includes qualified references to other software.
R3. Software meets domain-relevant community standards.

FAIR4RS (RDA) [1]

[1] Chue Hong, Neil P., Katz, Daniel S., Barker, Michelle, Lamprecht, Anna-Lena, Martinez, Carlos, Psomopoulos, Fotis E., Harrow, Jen, Castro, Leyla Jael, Gruenpeter, Morane, Martinez, Paula Andrea, Honeyman, Tom, Struck, Alexander, Lee, Allen, Loewe, Axel, van Werkhoven, Ben, Jones, Catherine, Garijo, Daniel, Plomp, Esther, Genova, Francoise, … RDA FAIR4RS WG. (2022). FAIR Principles for Research Software (FAIR4RS Principles) (1.0). https://doi.org/10.15497/RDA00068

FAIR is highly related to metadata

F	1	Create a description of your software
	2	Register your software in a software registry
	3	Use a Unique and Persistent Identifier for your software
A	4	Make sure that people can download your software
I	5	Explain the functionality of your software
	6	Use standard (community agreed) formats for inputs and outputs
R	7	Document your software
	8	Give your software a licence
	9	State how to cite your software
	10	Follow best practices for software development

https://github.com/LibraryCarpentry/Top-10-FAIR

F: Software, and its associated metadata, is easy for both humans and machines to find.

F1. Software is assigned a globally unique and persistent identifier.
- F1.1. Components of the software representing levels of granularity are assigned distinct identifiers.
- F1.2. Different versions of the software are assigned distinct identifiers.

F2. Software is described with rich metadata.
F3. Metadata clearly and explicitly include the identifier of the software they describe.
F4. Metadata are FAIR, searchable and indexable.

A: Software, and its metadata, is retrievable via standardized protocols.

A1. Software is retrievable by its identifier using a standardized communications protocol.
- A1.1. The protocol is open, free, and universally implementable.
- A1.2. The protocol allows for an authentication and authorization procedure, where necessary.

A2. Metadata are accessible, even when the software is no longer available.

I: Software interoperates with other software by exchanging data and/or metadata, and/or through interaction via application programming interfaces (APIs), described through standards.

I1. Software reads, writes and exchanges data in a way that meets domain-relevant community standards.
I2. Software includes qualified references to other objects.

R: Software is both usable (can be executed) and reusable (can be understood, modified, built upon, or incorporated into other software).

R1. Software is described with a plurality of accurate and relevant attributes.
- R1.1. Software is given a clear and accessible license.
- R1.2. Software is associated with detailed provenance.

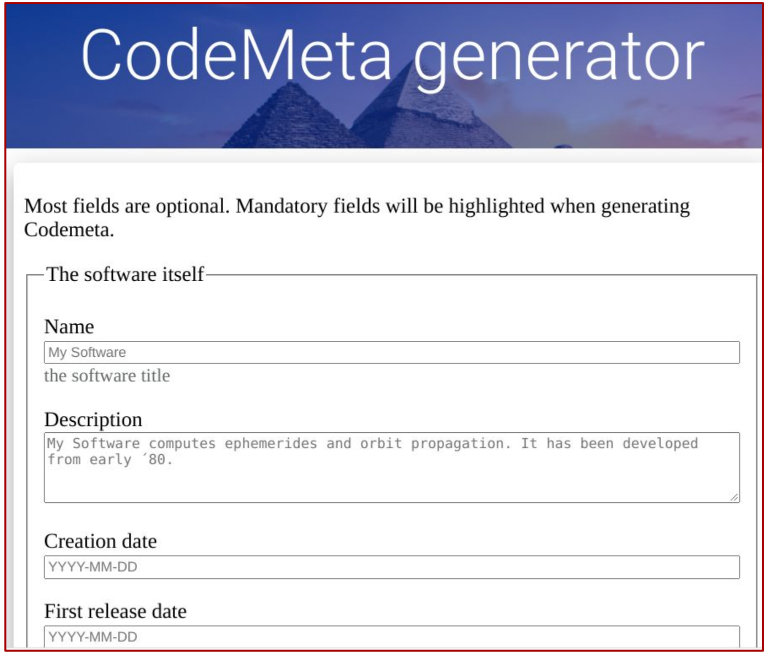
R2. Software includes qualified references to other software.
R3. Software meets domain-relevant community standards.

FAIR4RS (RDA) [1]

[1] Chue Hong, Neil P., Katz, Daniel S., Barker, Michelle, Lamprecht, Anna-Lena, Martinez, Carlos, Psomopoulos, Fotis E., Harrow, Jen, Castro, Leyla Jael, Gruenpeter, Morane, Martinez, Paula Andrea, Honeyman, Tom, Struck, Alexander, Lee, Allen, Loewe, Axel, van Werkhoven, Ben, Jones, Catherine, Garijo, Daniel, Plomp, Esther, Genova, Francoise, … RDA FAIR4RS WG. (2022). FAIR Principles for Research Software (FAIR4RS Principles) (1.0). https://doi.org/10.15497/RDA00068

*“The goal of CodeMeta is to create a **concept vocabulary** that can be used to standardize the exchange of software metadata across repositories and organizations”* - https://github.com/codemeta/codemeta

Website: https://codemeta.github.io/



JSON-LD representation

Needs to be **filled by hand**

Research Software metadata is not ~~abundant~~ machine readable

Can you please describe your software component with metadata?

I already did! Did you read the project readme?

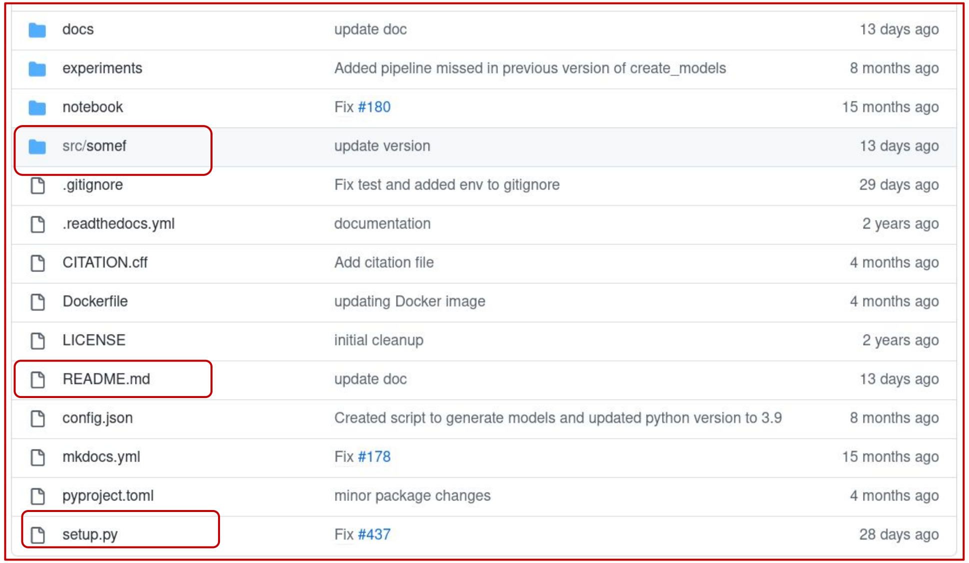
Did you see the online documentation?

Perhaps the you saw the paper?

Many domain-specific registries are curated by hand by experts

- Documentation
  - Text classification
  - Named entity recognition and relation extraction

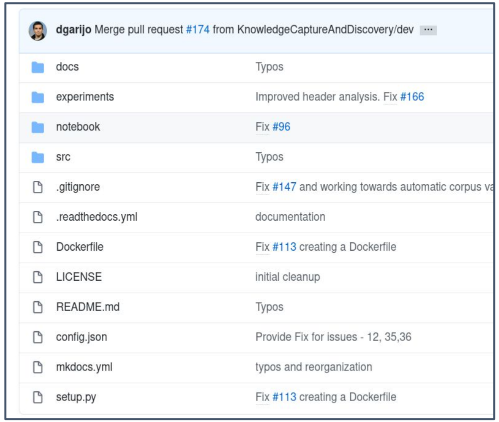
- Code
  - Static code analysis



Text classification: Software Metadata Extraction Framework

https://github.com/KnowledgeCaptureAndDiscovery/somef/

Repository → SOMEF Extraction → Results (Metadata)


dgarijo Merge pull request #174 from KnowledgeCaptureAndDiscovery/dev

docs	Typos
experiments	Improved header analysis. Fix #166
notebook	Fix #96
src	Typos
.gitignore	Fix #147 and working towards automatic corpus va
.readthedocs.yml	documentation
Dockerfile	Fix #113 creating a Dockerfile
LICENSE	initial cleanup
README.md	Typos
config.json	Provide Fix for issues - 12, 35,36
mkdocs.yml	typos and reorganization
setup.py	Fix #113 creating a Dockerfile

- **Readme Analysis**
  - Supervised classification
  - Regular expressions
  - Header analysis
- **File exploration**
  - Notebooks
  - Dockerfiles
  - Documentation
- **GitHub API**

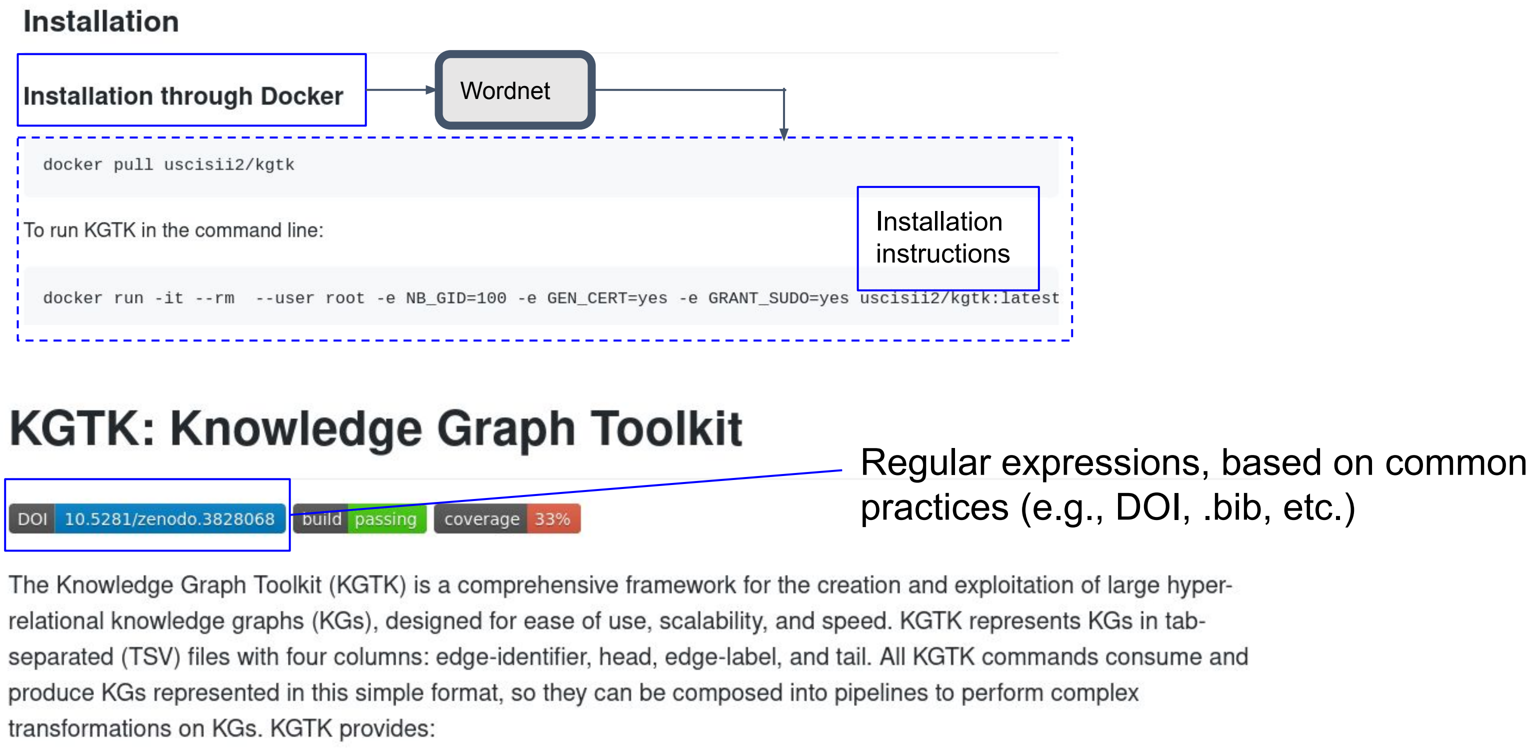
CodeMeta

TURTLE

JSON

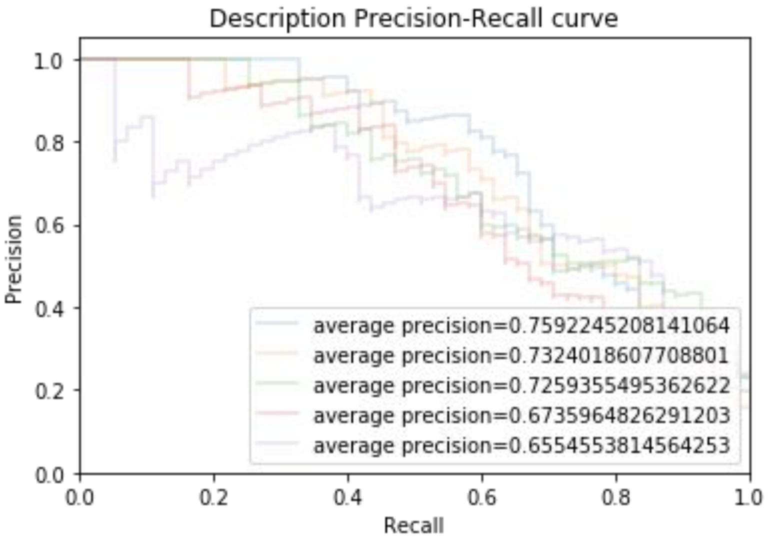
Kelley, A., & Garijo, D. (2021). A framework for creating knowledge graphs of scientific software metadata. *Quantitative Science Studies*, 1-37.

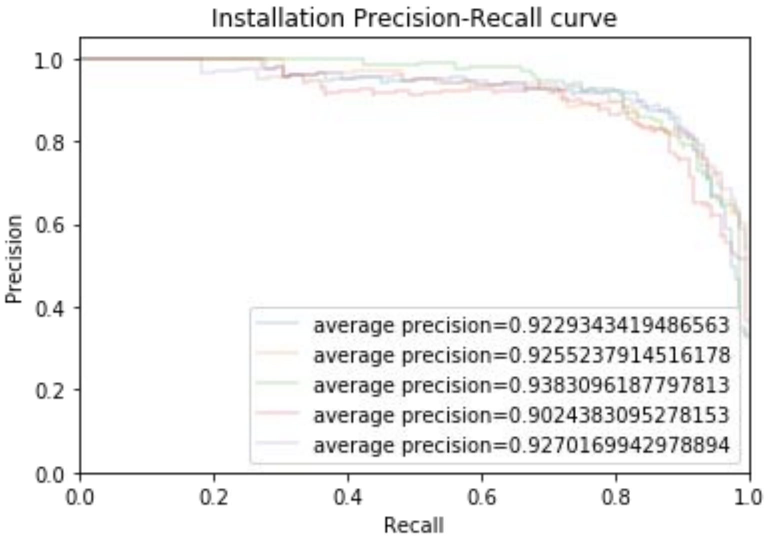
- Extraction based on frequent header analysis
  - Fuzzy matching based on synsets



- Paragraph-based text classification
- Four main categories:
  - Installation, citation, description, invocation.
- Binary classification problem

Truth Value	Category	Apprx. Ratio	Count
True	Description	0.5	275
False	Installation	0.125	68
	Invocation	0.125	68
	Citation	0.125	68
	Treebank	0.125	68
	Total	1.0	547





SOMEF: Recognizing Metadata Categories

- Name (GA)
- Full title (RE)
- Description (SC, HA)
- Citation (SC, RE, HA)
- Installation instructions (SC, HA)
- Invocation (SC)
- Usage examples (HA)
- Documentation (HA, FE)
- Requirements (HA)
- Contributors (HA)
- FAQ (HA)
- Support (HA)
- License (GA, HA, FE)
- Stars (GA)

Method used (provenance):

- Supervised Classification (SC)
- Header Analysis and Synset comparison (HA)
- File Exploration (FE)
- Regular Expressions (RE)
- GitHub API (GA)

- Contact (HA)
- Download URL (HA, GA)
- DOI (RE)
- DockerFile (FE)
- Notebooks (FE)
- Executable notebooks (Binder, Collab) (RE)
- Owner: (GA)
- Keywords (GA)
- Source code (GA)
- Releases (GA)
- Changelog (GA)
- Issue tracker (GA)
- Programming languages (GA)
- Acknowledgements (HA)
- Logos (RE)
- Images (RE)
- Shell scripts (FE)
- Code of conduct (FE)
- Repository status (RE)
- Arxiv links (RE)
- Support channels (RE)
- Software category (SC) (Work in progress)
- …

Static code analysis in Python

- Extraction of available classes and functions
  - Documentation
- Requirements (reusing existing libraries)
- Call list
- File hierarchy
- Control flow (reusing existing libraries)
- Software invocation
  - Service? Package? Library? & invocation command
- Metadata export in JSON

Inspect4py

Rosa Filgueira

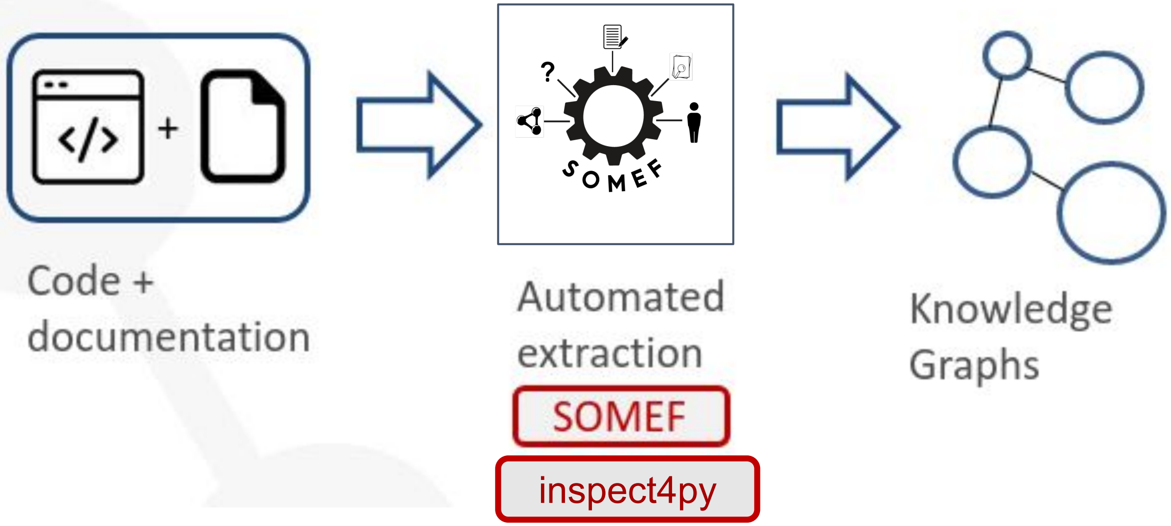
University of St Andrews

Benefits

- Understanding, reuse, ML featurization, similarity, best practices

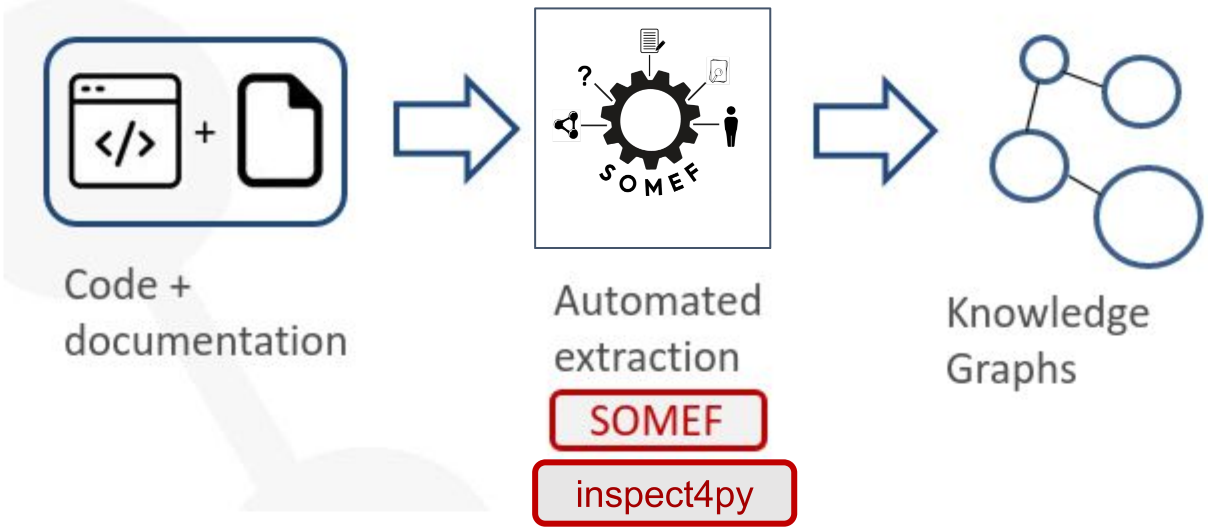
https://github.com/SoftwareUnderstanding/inspect4py

Filgueira, R. and Garijo, D. (2022). Inspect4py: A Knowledge Extraction Framework for Python Code Repositories. To appear in Mining Software Repositories, 2022 (demo)



What can we do now?

- Assist
  - Ease descriptions
  - Ease reuse
  - Augment impact
- Assess (measure practices)



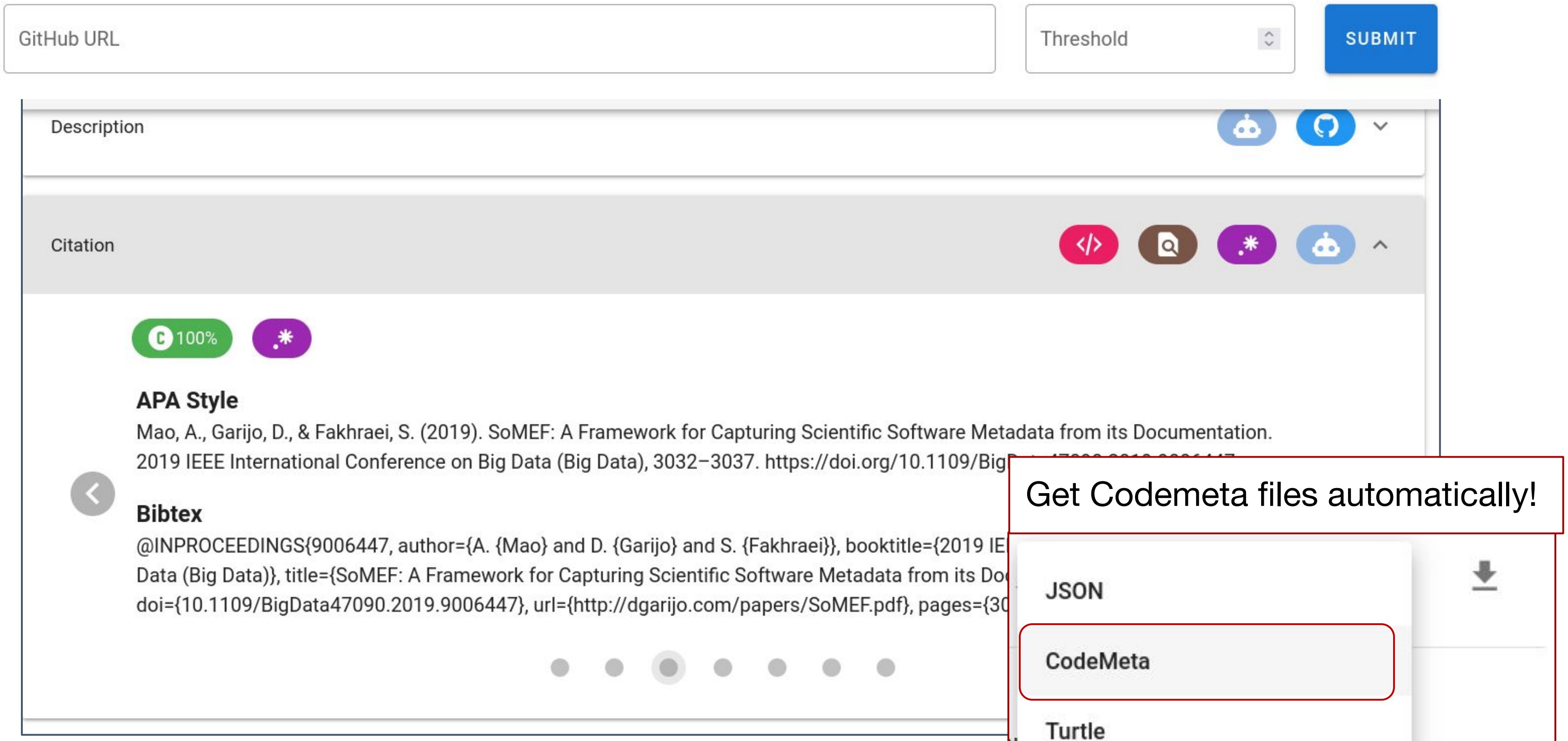
What can we do now?

- **Assist**
  - Ease descriptions
  - Ease reuse
  - Augment impact
- Assess (measure practices)

Easing descriptions: SOMEF-Vider

SOMEF Vider

https://somef.linkeddata.es/



Victor, F. A., & Daniel, G. (2021). SOMEF VIDER (Version 0.0.1) [Computer software]. https://github.com/SoftwareUnderstanding/SOMEF-Vider

Easing reuse and explore: Automated software catalogs

Useful for
- Comparison
- Exploring
- Reuse

Alpha available at: https://software.oeg.fi.upm.es/ Github: https://github.com/oeg-upm/soca

Work by Daniel Rodriguez

A software repository at a glance

link

logo

morph-kgc

Powerful RDF Knowledge Graph Generation with [R2]RML Mappings

morph

short description

notebooks

52

21

Usage

Learn quickly with the tutorial in **Google Colaboratory**!
PyPi is the fastest way to install Morph-KGC:
pip install morph-kgc
We recommend to use **virtual environments** to install Morph-KGC.
To run the engine via **command line** you just need to execute the following:
python3 -m morph_kgc config.ini
Check the **documentation** to can see how to generate the configuration **INI file**.
Here you can also see an example INI file.
It is also possible to run Morph-KGC as a **library** with **RDFLib** and **Oxigraph**:

```
import morph_kgc

# generate the triples and load them to an RDFLib graph
g_rdflib = morph_kgc.materialize('/path/to/config.ini')
# work with the RDFLib graph
q_res = g_rdflib.query(' SELECT DISTINCT ?classes WHERE { ?s a ?classes } ')

# generate the triples and load them to Oxigraph
g_oxigraph = morph_kgc.materialize_oxigraph('/path/to/config.ini')
# work with Oxigraph
q_res = graph.query(' SELECT DISTINCT ?classes WHERE { ?s a ?classes } ')

# the methods above also accept the config as a string
config = """
         [DataSource1]
         mappings: /path/to/mapping/mapping_file.rml.ttl
         db_url: mysql+pymysql://user:password@localhos
         """
g_rdflib = morph_kgc.materialize(config)
```

How to use it

python /morph-kgc/oeg-upm_morph-kgc/morph-kgc-main/src/morph_kgc/**main**.py

invocation

{Morph-KGC: Scalable Knowledge Graph Materialization with Mapping Partitions}

License

Apache License 2.0

Description:

A permissive license whose main conditions require preservation of copyright and license notices. Contributors provide an express grant of patent rights. Licensed works, modifications, and larger works may be distributed under different terms and without source code.

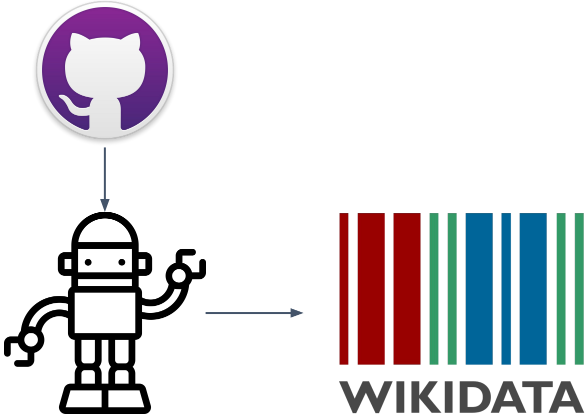
Permissions:

1. Commercial-use
2. Modifications
3. Distribution
4. Patent-use
5. Private-use

Citation

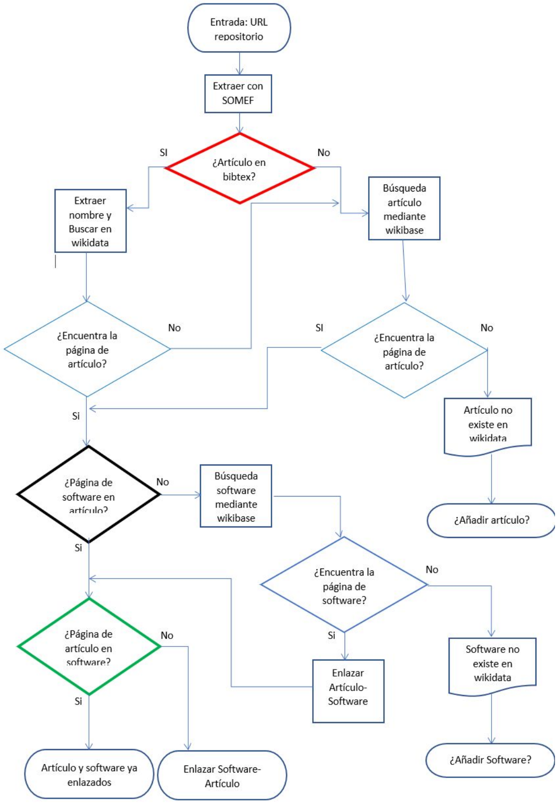
```
@article{arenas2022morph,
  title    = {{Morph-KGC: Scalable Knowledge Graph Materialization with
  author   = {Arenas-Guerrero, Julián and Chaves-Fraga, David and Toledo
  journal  = {Semantic Web},
  year     = {2022},
  url      = {http://www.semantic-web-journal.net/system/files/swj3135.p
}
```

Wikidata **bot** to link code repositories with Wikidata articles

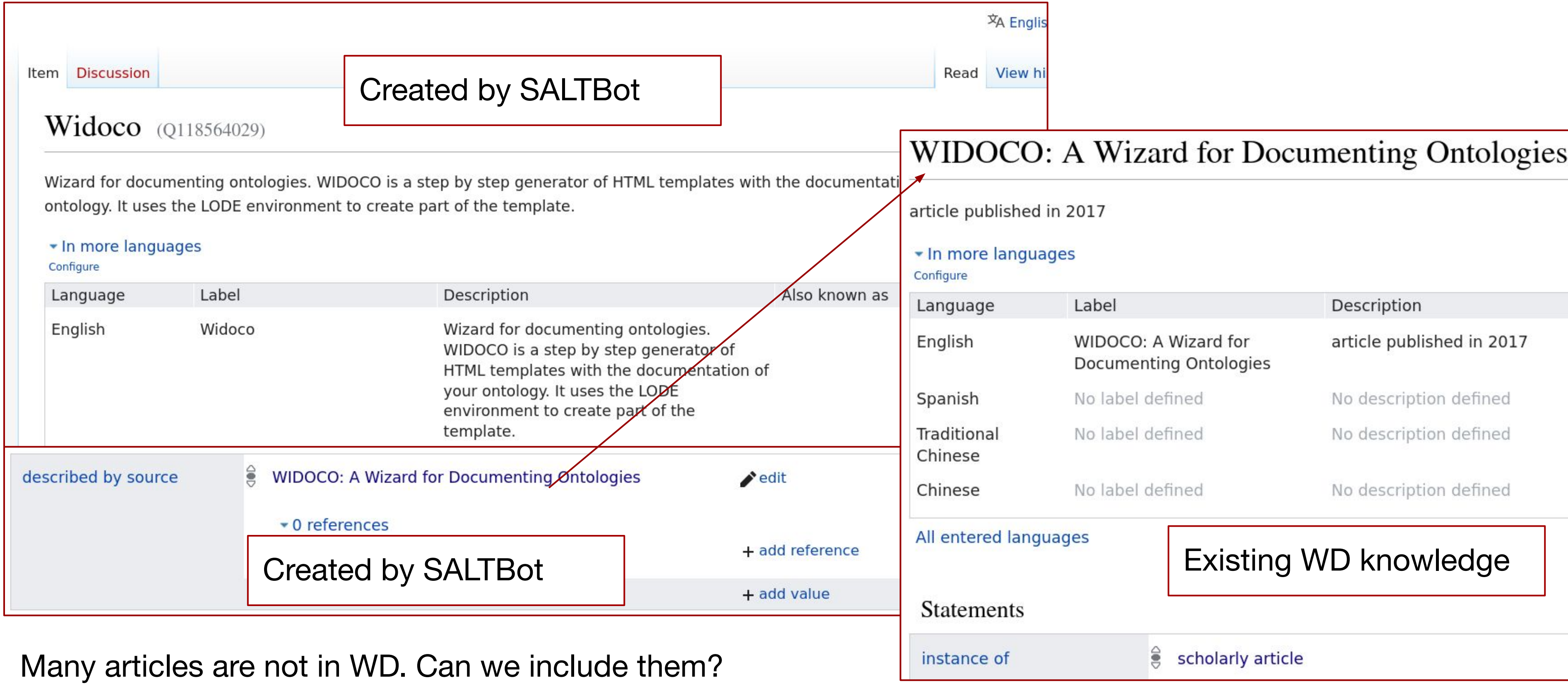


Does the repository have a link to a paper?
Does the paper exist in Wikidata?
Connect them! Create a software entry if it does not exist

Jorge Bolinches (work in progress)

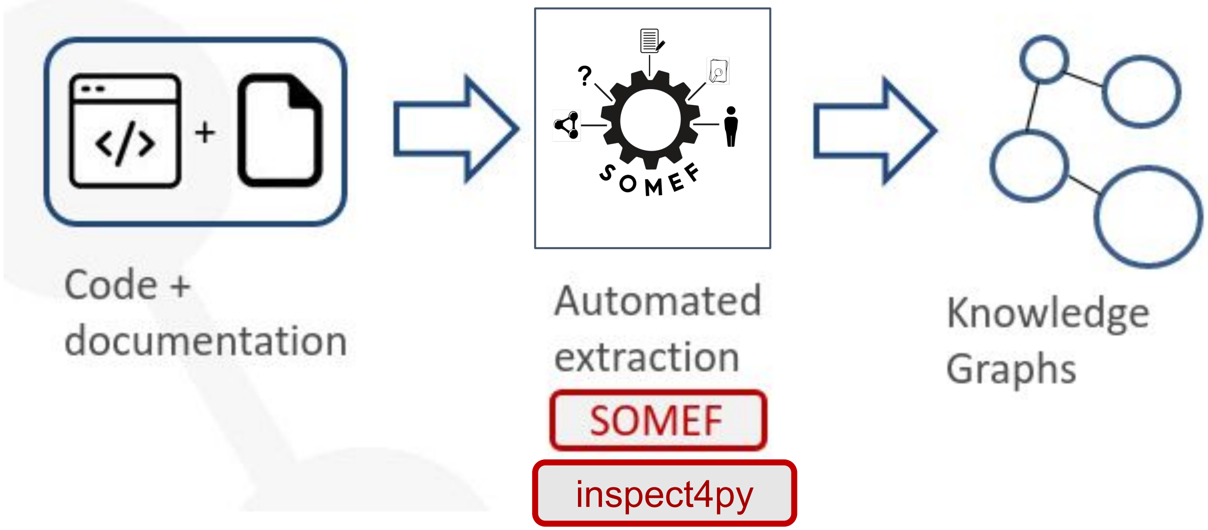


(included by Saltbot, from a CFF file)



Many articles are not in WD. Can we include them?

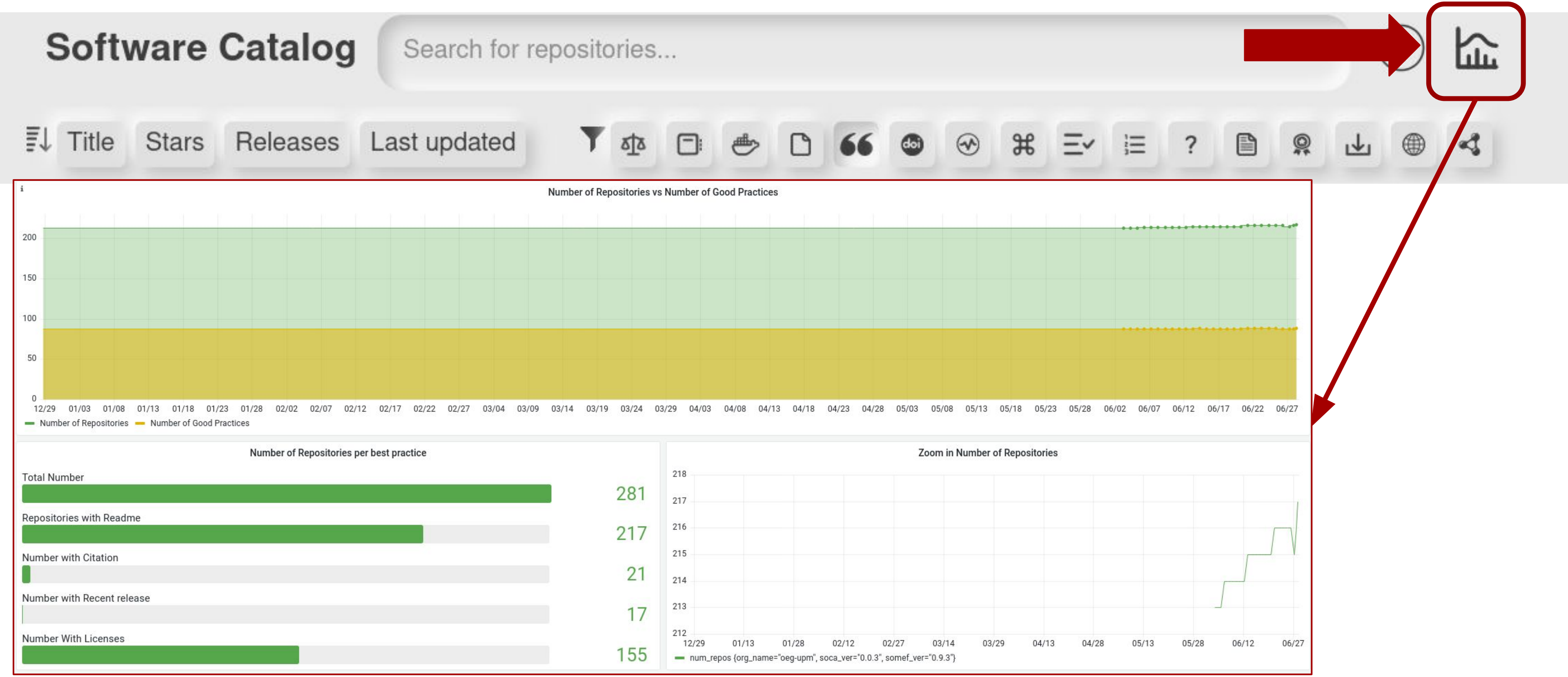
So what?



What can we do now?

- Assist
    - Ease descriptions
    - Ease reuse
    - Augment impact
- **Assess (measure practices)**

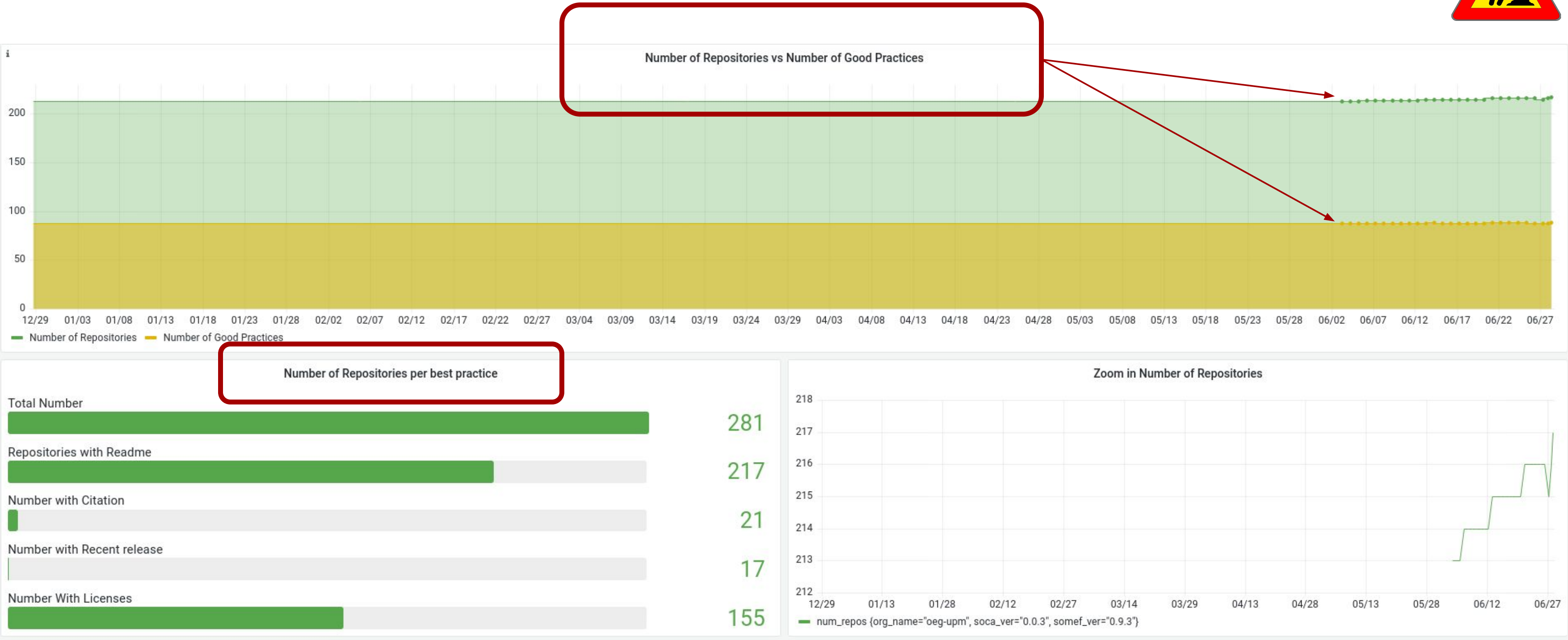
Which best practices are followed in an organization?

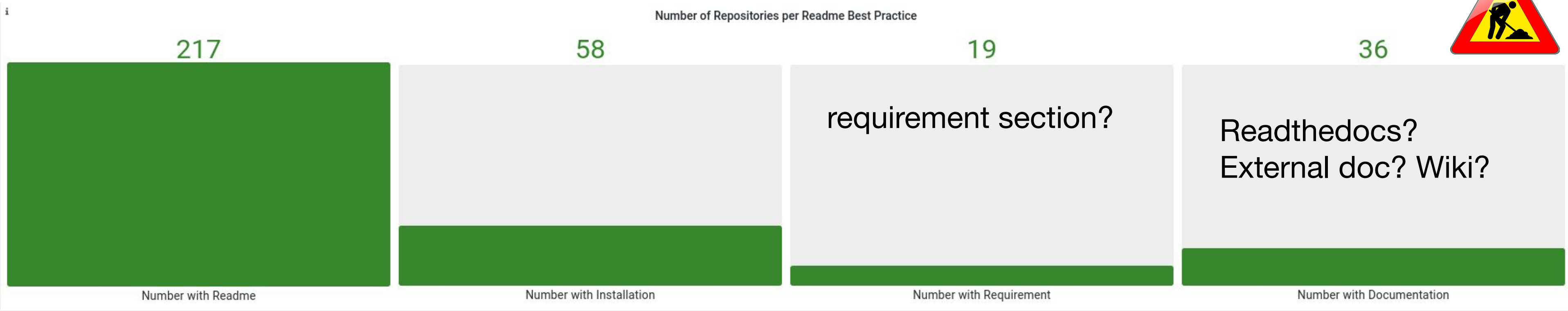


Tracking number of best practices across time

Work by Miguel Arroyo

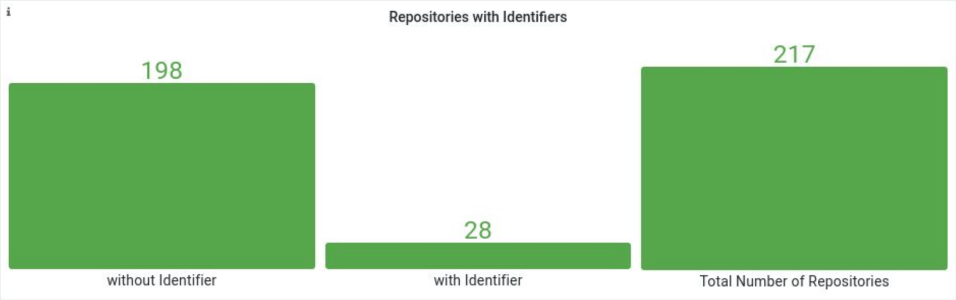
Assessing best practices

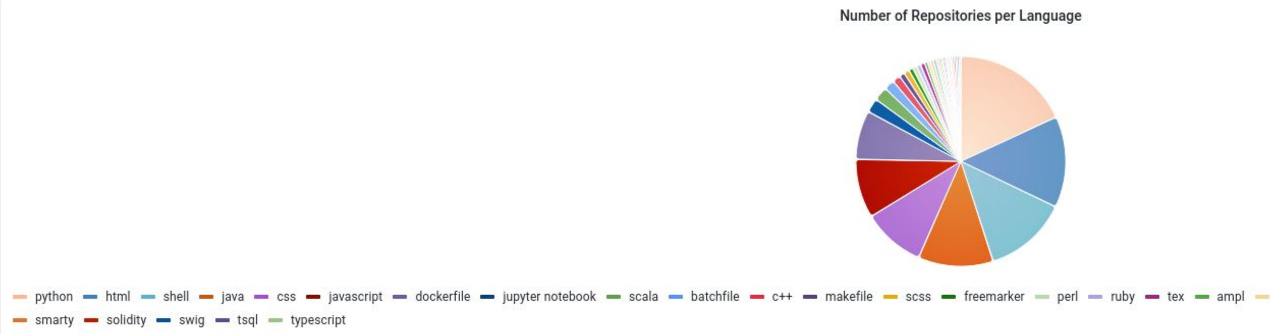




Additional insight on licenses used

Zenodo DOI? Paper DOI? Software heritage?





CFF file? Bibtex? Plain text citation?

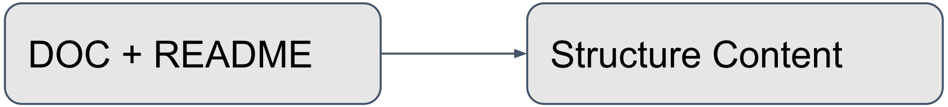
This is work in progress! We are including Containerization, package managers, individual repository assessment, etc.

Current life cycle requires researchers to:

- Create separate metadata files
- Curate them and maintain them
- Re-introduce metadata manually in different registries

Current aim:

- Improve and maintain high quality readme files
- Let the extraction tools do the work
- Maximize benefits from metadata extraction
    - Move **towards assisting researchers** produce FAIR software



Research software is a critical asset for **Open Science**

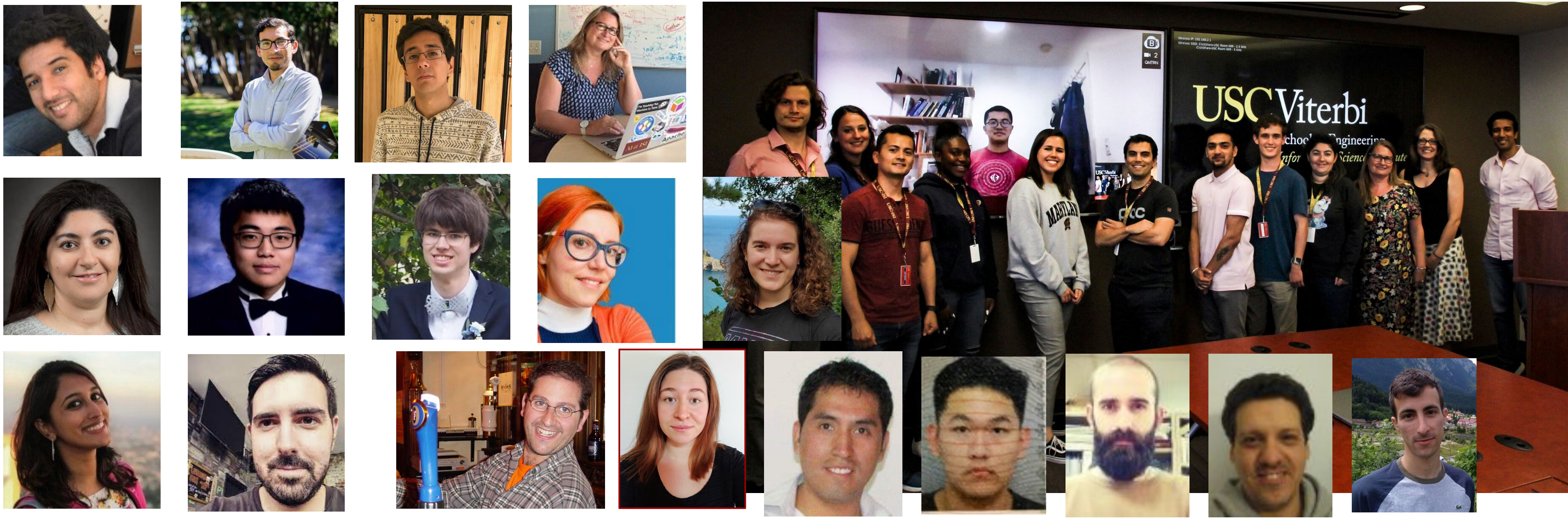
Software metadata **is key**:

- Reusability, Comparison, Search
- FAIR research software
- Propagate impact of research

From ASSESSing, we are working towards ASSISTing using metadata

- Highlight the **added benefit** to researchers and developers

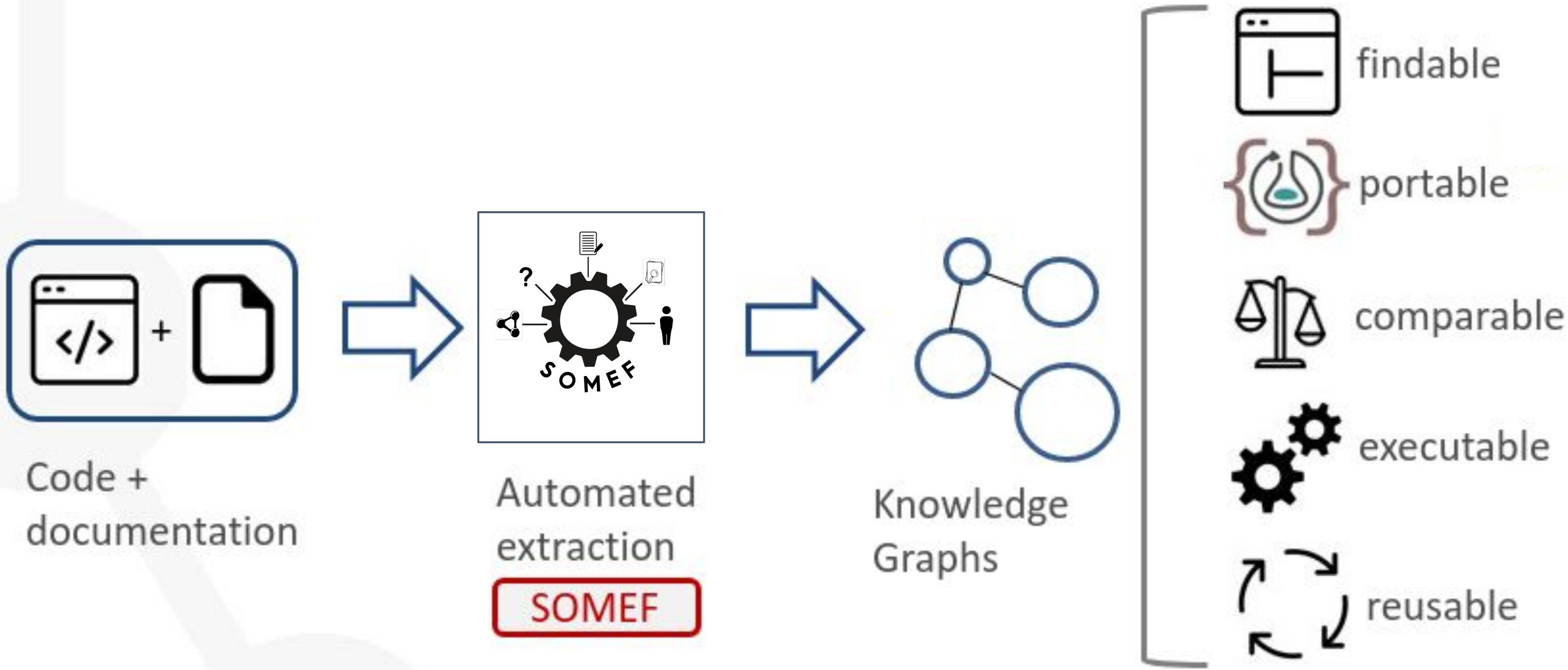
Acknowledgements



Thanks to Yolanda Gil, Varun Ratnakar, Maximiliano Osorio, Hernán Vargas, Deborah Khider, Allen Mao, Aidan Kelley, Haripriya Dharmala, Jiajing Wang, Rosa Filgueira, Pablo Calleja, Oscar Corcho, Laura Camacho, Jhon Toledo, Miguel Angel García, Esteban Gonzalez & all the students at UPM and USC who participated in the initiatives mentioned in this presentation

This work has been supported by the Madrid Government (Comunidad de Madrid-Spain) under the Multiannual Agreement with Universidad Politécnica de Madrid in the line Support for R&D projects for Beatriz Galindo researchers, in the context of the V PRICIT (Regional Programme of Research and Technological Innovation)

Questions?



Let's create **machine-actionable** software metadata