S3 School
Sustainable Scientific Software School

FAIRness and Software Publication

Making research software findable, accessible, interoperable, and reusable

Press Space for next page →

eosc | EVERSE
OSCARS
Open Science Clusters' Action for Research & Society
Funded by the European Union

Lecture Overview

Duration: 90min
Target Audience: Research software developers, PhD students, postdocs, researchers

Topics

1. FAIR for Research Software (FAIR4RS) principles
2. The FAIR4RS principles in practice
3. Publishing Research Software

Learning Outcomes:

- Understand FAIR4RS principles
- Assess software quality and maturity
- License software
- Create metadata and citation files
- Version and release software
- Publish and archive software properly

FAIRness of Research Software

Quality Dimensions We've Covered This Week

Development Practices

- **Virtual Environments** → Flexibility, Maintainability
  - Isolated dependencies
  - Reproducible setups
- **Unit Testing** → Maintainability, Functional Suitability
  - Verify correctness
  - Enable refactoring
- **Debugging** → Maintainability
  - Find and fix issues
  - Understand code behavior
- **Documentation** → Maintainability
  - Explain purpose and usage
  - Onboard new contributors

Advanced Topics

- **Profiling/Optimizing** → Performance Efficiency
  - Identify bottlenecks
  - Improve resource usage
- **Containerization** → Flexibility, Sustainability
  - Reproducible environments
  - Easy deployment
- **Security** → Security, Reliability
  - Protect against vulnerabilities
  - Secure data handling
- **Coding with AI** → Interaction Capability, Functional Suitability
  - Accelerate development
  - Generate boilerplate code

💡 An important one still missing: FAIRness

The Missing Dimension: FAIRness

What is FAIR?

4 principles for data objects:

- **F**indable - Easy to discover by humans & machines
- **A**ccessible - Retrievable via standard protocols
- **I**nteroperable - Exchange data through standards
- **R**eusable - Usable and modifiable by others

FAIRness is about discoverability and reusability

FAIR vs Quality

- FAIR ⊂ Quality Software
- FAIR ensures **discoverability** & **reusability**
- Quality includes **correctness**, **performance**, **testing**

FAIR Principles for Research Software (FAIR4RS)

F.indable

Easy for humans and machines to find.

- **F1.** Assigned unique & persistent ID (DOI)
  - **F1.1.** IDs for different components
  - **F1.2.** IDs for different versions
- **F2.** Described with rich metadata
- **F3.** Metadata explicitly points to ID
- **F4.** Metadata are searchable & indexable

A.ccessible

Retrievable via standard protocols.

- **A1.** Retrievable by ID using standard protocols
  - **A1.1.** Open, free & universal protocol
  - **A1.2.** Auth/Auth procedure where needed
- **A2.** Metadata persists even if software is gone

I.nteroperable

Exchange data and interact via APIs.

- **I1.** Meets community standards for exchange
- **I2.** Includes qualified references to other objects

R.eusable

Understandable, modifiable, and buildable.

- **R1.** Rich and accurate attributes
  - **R1.1.** Clear and accessible License
  - **R1.2.** Detailed provenance & history
- **R2.** References to other software
- **R3.** Meets domain-relevant community standards

Chue Hong, N. P. et al. (2022). FAIR Principles for Research Software (FAIR4RS Principles). DOI: 10.15497/RDA/00068

FAIR4RS in Practice

Translating abstract principles into concrete tools and files in your repository.

F.indable

- **Repository & Identifiers**
  Public Git repo + DOI (Zenodo/Figshare) or SWHID
- **Standard Metadata**
  `codemeta.json` and `CITATION.cff` files
- **Indexing**
  Register in PyPI, Conda-forge, or domain registries

A.ccessible

- **Software Access**
  HTTPS/SSH for clones, `pip install` for users
- **Metadata Longevity**
  Archiving in Zenodo ensures metadata stays even if repo disappears

I.nteroperable

- **Standard Formats**
  Use CSV, JSON, HDF5, or community-specific standards
- **Qualified References**
  Reference other tools/data using their DOIs
- **Controlled vocabularies**
  Standard terminology/Domain ontologies

R.eusable

- **Documentation**
  Rich `README.md` , usage examples, and API docs
- **Legal Terms**
  Include a `LICENSE` file (MIT, Apache, GPL)
- **Community & Provenance**
  `CONTRIBUTING.md` and `CHANGELOG.md`

FAIRness Assessment Tools

Available Tools

- **FAIR Software Checklist** - Self-assessment
- **howfairis** - Command-line and online tool

Purpose

- 🎯 **Diagnostic**, not evaluative
- 📊 Make quality aspects visible
- 🔍 Identify strengths & areas for improvement
- 📈 Guide reflection and learning

⚠️ Not meant to criticize - but to help improve!

Source: RSQKit - FAIR Research Software

Exercise

Run `howfairis` on `pkoffee` and discuss results

```
pip install howfairis
howfairis https://github.com/<username>/pkoffee
```

```
docker run --rm fairsoftware/howfairis https://github.com/s3-school/pkoffee
```

or go to https://www.howfairis.com/, connect your GitHub account and run on your `pkoffee` fork.

Example Output

```
(1/5) repository
      ✓ has_open_repository
(2/5) license
      × has_license
(3/5) registry
      × in_package_registry
(4/5) citation
      × has_citation_file
(5/5) checklist
      × has_checklist
```

Let's try to improve that evaluation together →

Software metadata and Essential Files

Software Licensing

Why License?

- Defines what others can do
- Required for legal reuse
- Part of FAIR principles (R1.1)

❌ **No license = No one can legally use your code**

Even if it's on GitHub!

License Categories

1. **Public Domain** - No restrictions
   - CC0, Unlicense
2. **Permissive** - Minimal restrictions
   - MIT, Apache 2.0, BSD
3. **Copyleft** - Share-alike required
   - GPL v3, AGPL, LGPL
4. **Creative Commons** - For non-code
   - CC-BY, CC-BY-SA (not for software!)

Source: RSQKit - Licensing Software

Popular Licenses for Research

MIT License ⭐

Most Popular

✅ Commercial use
✅ Modification
✅ Distribution
✅ Private use
✅ No liability

⚠️ Derived work must include license

Short, simple, permissive

Apache 2.0

Patent protection

✅ Same as MIT
✅ Patent grant
✅ Trademark protection

⚠️ Must state changes
⚠️ Include NOTICE file

Better for large projects

GPL v3

Strong copyleft

✅ Derivatives must be open
✅ Anti-tivoization
✅ Patent grant

⚠️ Can limit adoption
⚠️ Incompatible with some licenses

Ensures freedom

👉 choosealicense.com

💡 For research software: MIT or Apache 2.0 are most common. Use GPL if you want to ensure derivatives stay open.

Source: RSQKit - Licensing Software

Exercise: Add a License to Your Project

Steps

1. **Choose a license**
   - Use choosealicense.com
   - Consider your goals
   - Check funder requirements
2. **Add LICENSE file**
   - Create `LICENSE` or `LICENSE.txt`
   - Copy license text
   - Fill in year and copyright holder
3. **Add to metadata**
   - Update `pyproject.toml`
   - Add to `codemeta.json`
   - Include in `CITATION.cff`
4. **Add license headers** (optional)
   - Add to source files
   - Use SPDX identifiers

Example: MIT License

```
Copyright 2026 Thomas Vuillaume

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated document
to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, pub
and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FO
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWA
DEALINGS IN THE SOFTWARE.
```

In pyproject.toml

```toml
[project]
name = "my-package"
license = {text = "MIT"}
# or
license = {file = "LICENSE"}
```

SPDX Header (optional)

```
# SPDX-License-Identifier: MIT
# Copyright (c) 2026 Thomas Vuillaume
```

Source: RSQKit - Licensing Software

Software Metadata

What is Metadata?

Structured data describing your software:

- 📝 Name, version, description
- 👥 Authors, contributors
- ⚖️ License
- 🔗 Repository URL
- 🐍 Programming language
- 📦 Dependencies
- 📄 Documentation links

💡 Machine-readable metadata enables discoverability & automation

Why It Matters

- 🔍 **Findability** - Search engines can discover it
- 🤖 **Automation** - Tools can process it
- 🔄 **Interoperability** - Different platforms understand it
- 📚 **Archives** - Zenodo, Software Heritage can ingest it
- 📖 **Citation** - Automatic citation generation

Different use cases need different metadata:

- Citation: Authors, DOI
- Replication: Dependencies, versions
- Discovery: Keywords, description

Source: RSQKit - Software Metadata

Metadata Standards

Common Standards

pyproject.toml

- Package manager metadata
- Language-specific

CodeMeta

- JSON-LD format
- Based on Schema.org
- `codemeta.json`
- Widely supported (Zenodo, Software Heritage)

Citation File Format (CFF)

- YAML format
- Academic citation
- `CITATION.cff`
- GitHub native support (Shows a button "Cite this repository" automatically)
- Zenodo support
- Specifies prefeffed citation

Comparison

Feature	CodeMeta	CFF
Format	JSON-LD	YAML
Purpose	General	Citation
GitHub Support	Via API	Native
Human Readable	Medium	High
Machine Readable	✔	✔

Best Practice

Use both!

- `codemeta.json` for comprehensive metadata
- `CITATION.cff` for citation
- Plus language-specific files

Source: RSQKit - Software Metadata

CodeMeta example

`codemeta.json`

```
{
  "@context": "https://doi.org/10.5063/schema/codemeta-2.0",
  "@type": "SoftwareSourceCode",
  "name": "My Research Software",
  "description": "A tool for scientific data analysis",
  "version": "1.0.0",
  "author": [{
    "@type": "Person",
    "givenName": "Jane",
    "familyName": "Doe",
    "email": "jane@example.org",
    "affiliation": {
      "@type": "Organization",
      "name": "University of Example"
    }
  }],
  "license": "https://spdx.org/licenses/MIT",
  "programmingLanguage": "Python",
  "codeRepository": "https://github.com/user/repo"
}
```

Tools:

- CodeMeta Generator - Web form
- SOMEF - Automatic extraction
- autocodemeta - Automatic extraction as web service
- CodeMeta Lookup - Crosswalks

Source: https://codemeta.github.io/

Citation File Format (CFF) example

`citation.cff`

```yaml
cff-version: 1.2.0
message: "If you use this software, please cite it as below."
title: "My Research Software"
version: 1.0.0
date-released: 2024-01-15
authors:
  - family-names: "Doe"
    given-names: "Jane"
    orcid: "https://orcid.org/0000-0000-0000-0000"
    affiliation: "University of Example"
repository-code: "https://github.com/user/repo"
license: MIT
keywords:
  - research software
  - data analysis
preferred-citation:
  type: article
  title: "Software Paper Title"
  authors:
    - family-names: "Doe"
      given-names: "Jane"
  doi: "10.1234/example.doi"
  journal: "Journal of Open Source Software"
  year: 2024
```

Tools:

- cffinit - Web form
- CFF Validator - Check syntax

Source: RSQKit - Software Metadata

Exercise: Create Metadata Files

⏱️ 15 minutes hands-on

Your Task

Create both metadata files for your `pkoffee` project:

1. codemeta.json

- Use autocodemeta
- Fill in your repository URL
- Add missing information
- Download `codemeta.json`

2. CITATION.cff

- Use CFF Initializer
- Add your author information
- Include repository URL
- Download and validate

3. Add to Repository

- Place files in repository root
- Commit and push
- Verify GitHub recognizes them -> Cite button appears

Bonus

- Try `howfairis` again - did your score improve?

💡 Use journal preprint as related paper

💡 These files will be used when archiving to Zenodo!

Summary: Essential Files for Publication

📄 README.md

- Project description
- Installation instructions
- Usage examples
- Dependencies
- Citation information
- Contact details

```
# My Research Software

## Description
Brief description of what it does

## Installation
\```bash
pip install my-software
\```

## Usage
\```python
import my_software
result = my_software.analyze(data)
\```

## Citation
If you use this software, please cite:
[DOI or paper reference]
```

⚖️ LICENSE

Without a license, code cannot be legally reused!

📋 Metadata & Citation

- `codemeta.json` (General metadata)
- `CITATION.cff` (Academic citation)

🤝 CONTRIBUTING.md

- How to contribute
- Code of conduct
- Development setup

📝 CHANGELOG.md

- Version history
- What changed between releases

📚 docs/

- Detailed documentation
- API reference
- Tutorials

Source: RSQKit - Publishing Software

Publishing Research Software

Software Publication ≠ Code Hosting

Code Hosting (GitHub/GitLab)

- ✅ Version control
- ✅ Collaboration
- ✅ Issue tracking
- ✅ Code review

⚠️ This is a great start, but not enough!

Full Publication Includes

- 📄 **Documentation** - README, guides
- ⚖️ **License** - Legal reuse terms
- 📋 **Metadata** - Findability
- 🏷️ **Citation** - Academic credit
- 📦 Packaging - Easy installation
- 🏷️ Releases - Version management
- 🏛️ Archiving - Long-term preservation

💡 Publishing is the finale touch to make your software FAIR

Source: RSQKit - Publishing Software

Software Releases and Versioning

What is a Release?

A snapshot of your software at a specific point in time, made available to users.

Components:

- 🏷️ **Version number/name**
- 📝 **Changelog**
- 📄 **Release notes**
- 📦 **Artifacts** (binaries, packages)

> 💡 Releases provide stable reference points for users and citations

Versioning Schemes

Semantic Versioning (SemVer)

```
MAJOR.MINOR.PATCH
  1   .   2   .   3

MAJOR: Breaking changes
MINOR: New features (backward compatible)
PATCH: Bug fixes
```

Examples

- `1.0.0` → First stable release
- `1.1.0` → Added new feature
- `1.1.1` → Fixed bug
- `2.0.0` → Breaking change

Source: RSQKit - Releasing Software

Automated Versioning (advanced users)

🏷️ setuptools_scm

Infer version automatically from Git tags.

▶ See pyproject.toml content
▶ See pixi.toml content

Version format:

- On tag v0.1.0 → version is 0.1.0
- Between tags → 0.1.0.dev3+g1234567 (dev version with commit info)

No manual version bumping needed - just create git tags when you want to release. The version is computed at build time from your git history.

🤖 Python Semantic Release

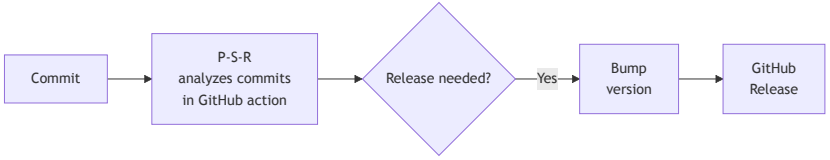
Automates versioning, changelog, and tagging based on commit history.

Requirement: Uses Conventional Commits

- `feat: ...` → **Minor**
- `fix: ...` → **Patch**
- `BREAKING CHANGE: ...` → **Major**

In GitHub Actions:

```
- name: Release
  uses: python-semantic-release/python-semantic-release@v
  with:
    github_token: ${{ secrets.GITHUB_TOKEN }}
```



Creating a GitHub Release

Steps

1. **Prepare**
   - Update version in code: `pyproject.toml`, `codemeta.json`, `citation.cff`
   - Ensure tests pass
2. **Create release on GitHub**
   - Go to Releases → "Draft a new release"
   - Create a new tag
   - Write release notes
     - Note: you can generate them based on past PRs
     - It's good to add a summary at the beginning
   - Attach binaries if needed
3. **Publish**
   - Review everything
   - Click "Publish release"
   - Zenodo integration triggers (if enabled, see after)

Release Notes Template

```
## What's New in v1.0.0

### Features
- Added support for new data format (#42)
- Improved performance by 50% (#38)

### Bug Fixes
- Fixed crash on empty input (#45)
- Corrected calculation error in module X (#41)

### Breaking Changes
- Removed deprecated function old_api()
- Changed default behavior of process()

### Dependencies
- Updated numpy to 1.24+
- Added new requirement: pandas >= 1.5

## Contributors
Thanks to @user1, @user2 for contributions!
```

Python Packaging and Distribution (PyPI)

🛠️ Building Your Package

Ensures your code is packaged correctly for distribution.

1. **Ensure `pyproject.toml` is complete**
   - Metadata, dependencies, build-system
2. **Install build tools**

```
pip install build twine
```

3. **Build the package**

```
python -m build
```

This creates `dist/` with `.whl` and `.tar.gz` files.

🚀 Publishing to PyPI

Makes your software installable via `pip install`.

0. **Setup Account**
   - Create account on PyPI and TestPyPI
   - Generate an **API Token** in Account Settings
1. **Upload to TestPyPI first** (Recommended)

```
python -m twine upload --repository testpypi dist/*
```

2. **Upload to PyPI**

```
python -m twine upload dist/*
```

❌ for pkoffee you won't be able to publish because it already exists on pypi

💡 Automate with GitHub Actions to publish a new package version at each release

Why Archive Software?

The Problem

GitHub/GitLab are NOT archives:

- Commercial platforms
- Can change policies
- Repositories can be deleted
- URLs can break
- No guarantee of permanence

⚠️ What happens to your research software in 10 years?

A Solution: Archiving

True archives provide:

- 🏛️ **Long-term preservation** (decades)
- 🔒 **Persistent identifiers** (DOIs)
- 📋 **Metadata preservation**
- 🔍 **Discoverability** in academic systems
- ✅ **Trustworthy** repositories
- 🌐 **Integration** with citation systems

Source: RSQKit - Archiving Software

Software Archives

Zenodo

- **General-purpose** archive
- CERN-hosted (Europe)
- **Free** and open
- **DOI** for each version
- **GitHub integration**
- Supports all file types
- Part of OpenAIRE

Good For:

- Software
- Datasets
- Supplementary materials

Software Heritage

- **Universal** software archive
- UNESCO-supported
- Preserves all public source code
- **Software Heritage identifier** (SWHID)
- Automatic archiving
- link from HAL
- Complete Git history preserved -> better granularity of identifiers

Good For:

- Software
- Being able to cite a specific part or commit of a software

💡 Recommendation: Use at least one

Zenodo + GitHub Integration

Setup Steps

1. **Create Zenodo account**
   - Visit zenodo.org (or sandbox.zenodo.org for the exercise)
   - Log in with GitHub
2. **Enable repository**
   - Go to GitHub settings in Zenodo
   - Toggle on your repository
3. **Create a release**
   - Tag and release on GitHub
   - Zenodo automatically archives
   - DOI is minted
4. **Update metadata if necessary**
   - Edit metadata on Zenodo
   - Add keywords, description
   - Save changes
5. **Add DOI badge**
   - Copy badge markdown
   - Add to README

What Gets Archived

- Complete repository snapshot
- Release artifacts
- Metadata from GitHub or `codemeta.json` or `CITATION.cff` (if present)

DOI Badge

```
[![DOI](https://zenodo.org/badge/DOI/10.5281/zenodo.17814297.svg)](https://doi.org/10.5281/zen
```

Displays as:

💡 Each release gets a separate DOI. Zenodo also creates a "concept DOI" for all versions.

Exercise: do it using zenodo sandbox (exact replicate of zenodo but gets empited regularly)

Software Heritage Demo

Save your code

https://archive.softwareheritage.org/save/

An example of saved code: gammapy

https://archive.softwareheritage.org/browse/origin/directory/?origin_url=https://github.com/gammapy/gammapy

Publication Checklist

Before First Release

- [ ] **LICENSE file** added
- [ ] **README.md** complete
  - Description
  - Installation
  - Usage examples
  - Citation
- [ ] **codemeta.json** created
- [ ] **CITATION.cff** created
- [ ] **Tests** written and passing
- [ ] **Documentation** available
- [ ] **Code formatted** and linted
- [ ] **Security scan** passed
- [ ] **CHANGELOG.md** started

For Each Release

- [ ] **Version bumped** (following SemVer)
- [ ] **CHANGELOG updated**
- [ ] **Tests passing**
- [ ] **Documentation updated**
- [ ] **Git tag created**
- [ ] **GitHub release** created
- [ ] **Release notes** written
- [ ] **Archived** (Zenodo/Software Heritage)
- [ ] **DOI obtained**
- [ ] **README updated** with DOI
- [ ] **Announced** to users

💡 Add this checklist to your GitHub repository wiki to keep it closeby when doing a release

Exercise

- Try `howfairis` one more time
- Compare before/after scores
- What improved?

⏱️ 20 minutes hands-on

Summary and Resources

Key Takeaways

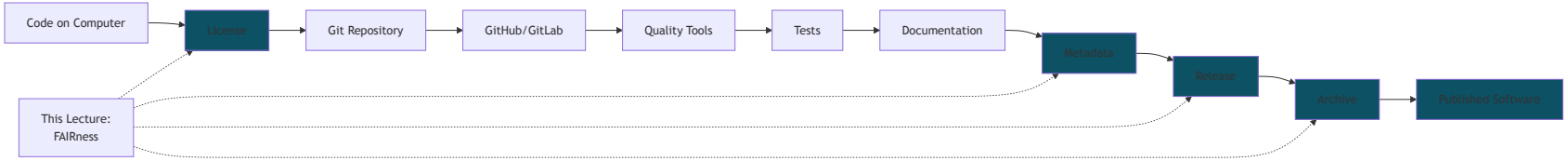
FAIR Principles

- **F**indable through metadata and identifiers
- **A**ccessible via standard protocols
- **I**nteroperable with standard formats
- **R**eusable with clear licenses

Essential Components

- License (MIT, Apache 2.0)
- Metadata (codemeta.json, CITATION.cff)
- Documentation (README, docs)
- Releases (semantic versioning)
- Archiving (Zenodo, Software Heritage)

From Code to Published Software



You now have a complete framework for creating high-quality, FAIR research software! 🎉

Resources and Further Learning

EVERSE RSQKit

- RSQKit Home
- FAIR Research Software
- Publishing Software
- Software Metadata
- Licensing
- Archiving

Tools

- Choose a License
- CodeMeta Generator
- CFF Initializer
- howfairis
- Zenodo

Guides & Documentation

- FAIR4RS Principles
- Software Citation Principles
- Zenodo Help
- Software Heritage
- Semantic Versioning

Community

- Research Software Engineers (RSE)
- EVERSE Project
- Software Sustainability Institute
- US-RSE

Questions?

thomas.vuillaume@lapp.in2p3.fr

Thank you!