Session B: Metrics for FAIR Software

Links

Slides including description of activity: https://doi.org/10.5281/zenodo.7805608

Session B description

- How can metrics can be used to improve the discoverability, access, interoperability and reuse of research software
- How to extend interoperability and improve the quality of metadata through the use of the vocabulary
- What are the criteria for trustworthy research software repositories?
- *Activity: collect indicators that can be used for FAIR research software metrics*

Funded by the European Union

FAIR Metrics

The work of the FAIR metrics suggest that a good FAIR metric should be:

- Clear: anyone can understand the purpose of the metric
- Realistic: it should not be unduly complicated for a resource to comply with the metric
- Discriminating: the metric should measure something important for FAIRness; distinguish the degree to which that resource meets that objective; and be able to provide instruction as to what would maximize that value
- Measurable: the assessment can be made in an objective, quantitative, machine-interpretable, scalable and reproducible manner, ensuring transparency of what is being measured, and how.
- *Universal: The metric should be applicable to all digital resources. [however some types of resources may have specific metrics]*

Funded by the European Union

How will we work during this session? Activity instructions

Times (all UTC)

9.30-10.45 AM : parallel session
11.15-11.30 AM: report back from parallel session

Activity	On site	On line	*Duration*
Sub-group introductions and instructions for activity	Choose rapporteurs and explain notes	Explain how session will work and explain note taking	10’
First task: Indicators of FAIR software (metadata and repositories)	F2. Rich description R1: Plurality of descriptive attributes	Trustworthy repositories for research software	20’
Second task: Indicators of FAIR software (identifiers and community standards)	F1.1 Component IDs F1.2 Version IDs R2 Qualified references	I1: Community standards R3: Community standards	15’
Closure activity: reflection			10’

Instructions

1. Choose a rapporteur - someone who is responsible for summing up. However everyone can add to the notes below.
2. For the topics or FAIR4RS principles assigned to your group, discuss
    a. Are there any existing metrics that apply to this principle?
    b. If not, what would be a good indicator that the principle was being met?
    c. What would count as an improvement if this metric / indicator was being applied?
3. Write down the metrics you have identified in the table below
4. After 20 minutes, your group will move on to the second set of FAIR4RS principles assigned to your group
5. At the end of the second discussion, the session leader will facilitate a reflection on the discussion - e.g. what were the challenges, are there any “easy to implement” metrics?
6. After the break, rapporteurs will report back - please add a summary of your discussion to the main session notes document:
    a. Metrics you identified
    b. Insights you learned from the process (e.g. challenges in defining metrics)

Funded by the European Union

FAIR Metrics (online group)

Examples/ideas of metrics for trustworthy repositories for research software

Notes:

- How do we define 'repository' in this context?
  - Service/organisation where you deposit objects for long term archiving (SoftwareHeritage)
  - Platform where you deposit software for publishing (Github)

  It can be both, maybe we shouldn't use the word 'repository' in this context as it creates confusion. Mechanism? Platform?

Descriptive name for metric	What is being measured?	What counts as success / improvement?
Filter software	The ability of the repository to distinguish software vs other deposits e.g. in a search using an API	The repository enables users to discover software stored in it, and refer to them in a persistent way through proper citation (links to F1). The repository allows versions to be distinguished easily.
Age of the repository of each entry	How long the repository has been available	The longer it has been there the higher the chances it will be available
Capability to capture provenance metadata	Presence of key fields relating to the origins of the software	Increase if detail for the same record. Linkage to other records.
Continuity plan	The repository should have a plan to ensure continuity of service and a succession/disaster plan in case their service ceases	It could be a yes/no, or certain elements of information need to be checked off for it to be complete enough
Use of persistent identifiers	The repository should use widely used recognized persistent identifiers for software (including versioning) and other digital objects	Implementation of PIDs
Openness	Can the content be freely accessed and exported (to avoid lock-in)	The content and its metadata can freely be accessed/exported. For repos like github: the associated artifacts and associated data can also freely

Funded by the European Union

Descriptive name for metric	What is being measured?	What counts as success / improvement?
		be accessed/exported (like contributors, history, issues, etc.).
Number of reused software	The number of software is actually re-used to produce research outputs	Increased number of re-use
Ownership	Reliability of the ownership (background)	Notoriety, reputation of the owner
Versioning	Version 1,2,3, .. of the software	It is clear what counts as a (new) version and this version is clearly indicated
Scientific context	The research for which the software was created	A list of scientific outputs for which the software was created/used
Last modification Granularity and identifiers	Measure of activity, when it was last modifiedIt should be allowed to persistently identify research objects of different granularity, e.g. collections, datasets for richer information about research.	Different levels of granularity are available for citing and it is possible to provide different identifiers to these levels.
Type of repository	Archive for immutable objects (versions of software/documents/data), archive of mutable objects (software being developed)	
Quality checks / curation	The repository should perform certain quality checks when ingesting new software, to ensure it's reusability and preservation for the long term	Certain levels of curation should be defined, depending on the activities carried out and the needed curation. I don't think one specific level of curation would count as a success, but rather knowledge on what curation is needed in which cases
Existence of certification	Classic measures of conventional data repositories (is there a statement of commitment to maintain records, are protocols for depot and retention publicly available etc)	Achievement of certification, incremental improvements inline with certification levels
Number of	The size of the community using	Increasing number of software

Funded by the European Union

Descriptive name for metric	What is being measured?	What counts as success / improvement?
software in the repository	this repository	deposited
Presence of a clear documentation for users	Presence/absence Frequency of updates	Increasing number of view/download - decreased n of days from the last update
Scholarly society endorsement, or complementary policy instruments	Does the scholarly society flag the publication infrastructure as a preferred option	Inclusion in policy
Existence of software data curators	Measure if the software deposited are curated by “professional” software curators.	List of people acting as software curators for the repository. Also clear guidelines on how to become such a curator.
Citation of the software	The repository should support the citation of the software and also count statistics related to citations	Increasing the number of citations
Licensing	The repository should support suitable license options to assign to the software	Different license options should be clearly communicated to depositors and other users
Existence of PIDs	PIDs for different versions and parts of the software(tests, core/libraries, tutorials)	PIDs are assigned by users and curators check that is is enforced
List of dependencies	If software require dependencies they need to be listed (with PIDs?)	Software may not be executable in the absence of specific versions of dependencies (later versions may/may not work).
Funding	A trustworthy repository should be able to show they have funding secured to continue operational, as well as back-up plans when funding fails	In many cases, long term funding cannot be proved in advance, but the focus could maybe be on back-up plans in case funding fails? Having a plan is then a sign of trustworthiness.
Curatorial protocols	The existence of public documentation describing how software is curated (or not)	Existence as a binary feature. Improvements would require met
Clear storage/disaster	Presence/absence of this documentation	

Funded by the European Union

Descriptive name for metric	What is being measured?	What counts as success / improvement?
recovery policy documentation		
Active test framework	The software still works	Whether the software still works
Support policy	If you encounter a problem will you obtain any support or are you on your own	
Support a Software Bill of Materials	Which dependency does the software have? Where can they be found, how trustworthy are those?	Having a clear list of dependencies. Being able to flag trust issues (security, quality, bugs) in software based on its dependencies.
The infrastructure enforces or encourages community standards	Existence of mechanisms in the service or guidance/documentation	Introduction of mechanism/guidance, rewarding uptake, enforcement of standards

Examples/ideas of metrics for I1 & R3 - meeting community standards

I1. Software reads, writes and exchanges data in a way that meets domain-relevant community standards.
R3. Software meets domain-relevant community standards.

Descriptive name for metric	What is being measured?	What counts as success / improvement?
Data standard read/write support	If a community standard exists, then does the software support read/writing/exchanging (as appropriate).	If data is read/written/exchanged by the software and if community standards exist then are they supported.
The object uses a recognised language or format including version or standard observed	For software you would find the code uses one (or more recognised languages) - C, FORTRAN, COBOL, Python, R, etc	Number of software with standard and open languages increases

Funded by the European Union

Descriptive name for metric	What is being measured?	What counts as success / improvement?
Openness of the standards used (for I1)	Can these standards be freely used? Are they open, do they have a license, proper definition, etc.	Standards used have an open license, are well defined, documented, etc.
List of best practices/standards used	If the software has put effort in following community standards and best practices.	
Source code is available	Whether source code is available.	Yes counts as a success
Clear statement of the communities the software supports	Whether software supports multiple communities	
Input requirements/dependencies	To operate data or other in a particular format must be made available	Recognised standards are used
Listing of Open standards that are observed	List of recognised standards that are used	
Contact point and/or link to website containing community standards	Existence of community standards	Yes is success
File format follows community-standards	Or is converted to one that follows community-standards	
Is the software written in a language commonly used by the audiences likely to use it, or is it in a form that enables reuse within the norms of that community (e.g., discrete modular tools for	What are the norms of a community and does the software fit within those.	Must be able to extract measures of norms first, and note that they can change over time. This will be hard! Improvement may be the *codification* of norms.

Funded by the European Union

Descriptive name for metric	What is being measured?	What counts as success / improvement?
use in workflows, rather than monolithic frameworks)		
Existence of software management plan and templates	Measure if the community agreed on the tools to use and offer templates for software management plan.	
Existence of contributing guidelines	Measure the openness (not technically) and if there is any onboarding procedure for new contributors	CONTRIBUTING file in each software repo
Domain specific certification for the software.	If the software has received the relevant certifications. For example, for medical software, security related software, AI regulations (introduced in EU), etc.)	It has the standards.
The output of the software uses a recognised standard or the outputs are well specified.	In order for the outputs to be reusable they should observe a recognised standard or the output format should be well specified.	
Endorsement from specific communities/users	A star based system as in GitHub or dare I say it as in Amazon?	Produce a standard way for encoding this that repository providers could use
Software is linked to a relevant community or communities	As a first step is their evidence that the software is cognizant of community needs or norms. This could be references to a broad array of “communities” (user/creator/developer/maintainer/ contributor, researcher communities, methods communities, general public norms, policy makers)	A mention of this detail first, possibly with more formal measures over time.
Workflow integration	The software can be used in workflows commonly used by the community	

Funded by the European Union

Funded by the European Union

FAIR Metrics (on-site / in-person group)

Examples/ideas of metrics for F2 & R1 - descriptive metadata and attributes

F2. Software is described with rich metadata.
R1. Software is described with a plurality of accurate and relevant attributes.

Descriptive name for metric	**What is being measured?**	**What counts as success / improvement?**
Clear software license	Is there an easily identifiable license for the software	Inclusion of license files with standard names Use of SPDX
Software purpose	Is it easy to understand the domain and problem being solved by the software within a couple of sentences	- README with clear first paragraph - Language specific way of describing software
Programming Language	The language is identified,	Language includes version, platform, dependencies e.g., in codemeta field “programminglanguage”. Related to “software components” metric
Uses a standard API	The software uses a community-standard API	Uses a standard API; the API that is used is named
Software components (libraries, middlewares)	Description of the software's components	A list of external libraries used in the software with, ideally, versions used
Attributes are ontological, and based on a published vocabulary	The domain of the software’s applicability/use needs to be described using machine actionable concepts	Utilisation of ontological concepts as attributes
Clear documentation	Whether there are installation instructions, usage instructions. Does the software component have a link to a documentation document?	A documentation file (e.g., readme, install.md) contains clear sections stating how to install and invoke a software component
Version/Versioning concept	Is there a clear version with release notes	Version with attached release notes and a definition of version

Funded by the European Union

		scheme documentation exist and linkages between versions exist
Authors	A list of the authors of the software	Are all names available. Are real names available Are ORCIDs included Associated with their role / contribution
Build/install information	Human and machine readable information about building or installing the software is provided	The software can be built/installed by a human; the software can be built/installed by a machine without detailed human action
Clear reference to related outputs	All outputs/data related with the software to be retrievable	To be linked back and forth
Clear workflow to report issues / bugs etc	Is there a place to add newly found issues? Timeliness of responses in that system	Existence of an issue tracker of some sort (github/gitlab etc; redmine, …) Past time until first triaging new issues
Link to software from a registry/catalog	Working, verifiable link to the software	
Purpose of the research software	In what context is the research software developed. PhD, research project, infrastructure	Link to grant proposal, publication,
Creation date is present	Code metadata includes a creation date	Basic: in the readme. More advanced: in codemeta field
Requirements file is available	Documentation/machine readable file exists to state the libraries the component depends on	A requirement file (e.g., requirements.txt, pom.xml), a section (human-readable) in readme stating this information
Governance	A group of responsible persons or a committee is provided with their responsibilities	documented governance structure
Citation for software component is clear	A citation is available, stating how the software component should be provided credit	Link to citation file, or section in readme stating what the citation should be. Ideally, cff file
Example data is	Code includes sample data or link to	Minimum: description of the data

Funded by the European Union

available	dataset (input or output)	Better: sample table Best DOI for a dataset
Original use is described	The original research project is described,	minimum, in readme. Better: keywords Best: semantic
Use of design patterns	Design patterns	
Contribution instructions	Are there any descriptions on how to help improve/report bugs etc?	CONTRIBUTING file Field in the README with instructions on contribution
Constraints	Clear description of what software can or can't be used for	Intellectual constraints Legal constraints
Use potential	What could the software (perhaps also) be used for?	Gauging re-use of software in new domains
Software has been vetted by a language community	Languages with user-groups have vetting systems (e.g., CRAN for R, Matlab file exchange(?)) with some level of vetting or quality control	Code or package is available thru a user-group

Examples/ideas of metrics for F1.1, F1.2 & R2 - identifiers and references

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.
R2. Software includes qualified references to other software.

Descriptive name for metric	**What is being measured?**	**What counts as success / improvement?**
Software Identifier	The software has an identifier that is globally unique and persistent	The identifier is unique and can be used to obtain information about the software, even if the software is no longer available The identifier is in identifiers.org (or provided by some other service that meets some identifier standards)

Funded by the European Union

Software releases	Snapshots of software at a point in time have a clear identifier	
Dependencies are clear	There is a requirements file that includes a way of identifying and understanding all dependencies	This file is machine actionable The role of a dependency is clear (dev vs required vs optional) And also reverse dependency tree (who relies on your software again)
Research software graph	Amount of relations	Basic: Better: components have pids
Software snapshot is archived	Whether a public archive contains one or more software version	The software component is archived in a public archive like Zenodo, Software Heritage, etc.
Reference to paper	Optional: In case a paper is referenced, it bears a PID	Additional information what is documented in the paper, e.g. method or evaluation or software paper,... Understanding the context of how the reference was used - e.g. is it a comparison, the citation for paper that contains the method, the citation for the software paper (Note Datacite has a vocabulary for types of references, and their relationship to data. Might be adaptable for software refs)
Different version are assigned a PID		
PID build file	Where dependencies have identifiers, these are published as a PID dependency tree	This is published so that it can be linked to others
Identifier and relationship resolution	What part of software is being referenced? How is it related to other parts of the software, including the complete software "unit"	All parts of the software can be uniquely identified (e.g. lines of source code, routines, files, packages, project, executables, containers, services, etc.), and the relationships between parts is clear

Funded by the European Union

The datatypes used in inputs and outputs uses is semanticallty described and identified		FAIRCORE4EOSC is developing a data type PID registry Also enables discoverability (find me software that processes these datatypes)
Input data has quantity identified	Input data is described well enough for a) code reuse or b) reproducibility.	Input data is tagged with a recognizable unit, quantity and/or type, e.g., with QUDT URI
Standard benchmark	Declare identifiers for the reference implementation / benchmark, and data sets used to compare work (e.g. In ML research)	

Difficulties encountered during exercise

- Hard to think about things that aren't open source
    - How do we apply these indicators for closed source software
- How to engage with the people who have to produce identifiers for their software - how to persuade them to use a different "better" identifier?
- For FAIR data, a lot relies on the repository you're using. For software, where to tell them they can share their software to make it more FAIR
    - Metrics for different types of reusability e.g. collaboration vs preservation
- Metrics for the repositories storing the software
- Derivatives of a PID, for versions of software that use each other, and the impact of provenance when we have copies of an original instance

Next steps

- Participants preferred the use of a Google Doc for providing further input / feedback
- FAIR-Impact should highlight areas in the document we are looking for clarification
- Provide regular email to highlight what's changed

Funded by the European Union

Facilitator Guidance

We are recommending that sessions are facilitated using a guided “parallel silent work” approach that makes use of Google Docs. This approach scales well, and is appropriate for both online and in-person sessions, but relies on participants having access to a device that allows them to view and contribute to the session notes document. If in-person participants do not have access to such a device, please see below for facilitation suggestions.

Instructions

- Introduce the activity and make sure participants are able to access and edit the notes document
- Ideally, project/share the table in the document so that all participants can see it being updated in real time
- Ask participants to separately and in parallel enter suggestions in the rows of the tables
- As these suggestions come in, comment on any interesting things you see in them, e.g. trends, contradictions.
- Once you have a reasonable number of suggestions (normally about 5-10 minutes) and the rate has slowed down, take a few interesting ones and invite the person who entered it to talk about why they suggested it, and facilitate some discussion. Ensure that all people have a chance to contribute by encouraging specific people to speak and keeping a watch on whether anybody is dominating discussion.
- At the end of the session, leave some time for facilitated discussion reflecting on any challenges or unexpected outcomes from doing this exercise

If a participant does not have access to the doc, you can get them to say their suggestion and add it directly. However, during the initial period of adding suggestions try to avoid the group breaking into discussion, and for any one person dominating the audio.

Funded by the European Union