

Project Title	Expanding FAIR solutions across EOSC
Project Acronym	FAIR-IMPACT
Grant Agreement No.	101057344
Start Date of Project	2022-06-01
Duration of Project	36 months
Project Website	fair-impact.eu

M5.6 - Practical tests for automated FAIR software assessment in a disciplinary context

Work Package	WP5 - Metrics, certification, and guidelines
Lead Author (Org)	Kara Moraw (UEDIN-SSI)
Contributing Author(s) (Org)	Mario Antonioletti (UEDIN-SSI), Elena Breitmoser (UEDIN-SSI), Neil Chue Hong (UEDIN-SSI), Mike Priddy (KNAW-DANS)
Due Date	2024-03-31
Date	2024-03-28
Version	V1.0
DOI	https://doi.org/10.5281/zenodo.10890043

Dissemination Level

<input checked="" type="checkbox"/>	PU: Public
<input type="checkbox"/>	PP: Restricted to other programme participants (including the Commission)
<input type="checkbox"/>	RE: Restricted to a group specified by the consortium (including the Commission)
<input type="checkbox"/>	CO: Confidential, only for members of the consortium (including the Commission)



Versioning and contribution history

Version	Date	Author	Notes
0.1	08.03.2024	Kara Moraw (UEDIN-SSI)	TOC and V0.1
0.2	22.03.2024	Kara Moraw (UEDIN-SSI), Mario Antonioletti (UEDIN-SSI), Elena Breitmoser (UEDIN-SSI), Neil Chue Hong (UEDIN-SSI), Mike Priddy (KNAW-DANS)	Main sections incorporating feedback.
1.0	28.03.2024	Kara Moraw (UEDIN-SSI), Mario Antonioletti (UEDIN-SSI), Elena Breitmoser (UEDIN-SSI), Neil Chue Hong (UEDIN-SSI), Mike Priddy (KNAW-DANS)	Updated TOC, added details to evaluation.

Disclaimer

FAIR-IMPACT has received funding from the European Commission's Horizon Europe funding programme for research and innovation programme under the Grant Agreement no. 101057344. The content of this document does not represent the opinion of the European Commission, and the European Commission is not responsible for any use that might be made of such content.

Table of Contents

1 Introduction	6
2 Description of the Milestone	7
2.1 Role of the milestone	7
2.1.1 Means of verification	7
3 Background information	8
3.1 Metric structure	8
4 Implementation of practical tests for FAIR Research Software Metrics in F-UJI	10
4.1 Extending F-UJI for FAIR4RS	10
4.2 Test implementations	17
5 Evaluation of tests in a disciplinary context	22
6 Conclusions and next steps	25
7 References	27
8 Appendices	28
8.1 Example of a metric definition in F-UJI using YAML	28
8.2 Repository-level evaluation results	28

TERMINOLOGY

Terminology/Acronym	Description
CESSDA	Consortium of European Social Science Data Archives
FAIR	Findable Accessible Interoperable Reusable
FAIR4RS	FAIR for Research Software
FRSM	FAIR Research Software Metric
FsF	FAIRsFAIR project (predecessor of FAIR-IMPACT)

1 Introduction

FAIR-IMPACT¹ have developed 17 metrics (Chue Hong et al., 2023) that can be used to automate the assessment of research software against the FAIR Principles for Research Software (FAIR4RS Principles) (Chue Hong et al., 2022). These build on the outputs of the RDA/ReSA/FORCE11 FAIR for Research Software Working Group² and existing guidelines and metrics for research software to define metrics for the assessment of the FAIR4RS Principles. FAIR software can be defined as research software which adheres to these principles, and the extent to which a principle has been satisfied can be measured against the criteria in a metric via a set of tests.

This work has been continued by fully implementing two of the metrics, FRSM-13³ (dependencies, build and configuration) and FRSM-15⁴ (licensing information), as well as providing skeletons for many of the other metrics. This was done by extending the F-UJI tool (Devaraju and Huber, 2020), which was originally developed for assessing datasets, to include tests for the research software metrics, and evaluating these discipline-agnostic tests against a collection of reference repositories provided by FAIR-EASE.⁵ Additionally, discipline-specific versions of the metrics were implemented based on the version of the metrics defined by CESSDA, and evaluated against a collection of reference repositories provided by CESSDA.⁶

This milestone comes from Task 5.2 (FAIR metrics for research software) on "*Practical tests for automated FAIR software assessment in a disciplinary context*" and is part of Work Package 5 on "Metrics, Certification and Guidelines" within the FAIR-IMPACT project.

This milestone report summarises the way that F-UJI was extended to accommodate software metrics, describes the implementation of the tests used to check each metric, and the results of FAIR assessments performed on the provided repositories.

¹ <https://fair-impact.eu/>

² <https://www.rd-alliance.org/groups/fair-research-software-fair4rs-wg>

³ FAIR Research Software Metric 13 as proposed by Chue Hong et al., 2023.

⁴ FAIR Research Software Metric 15 as proposed by Chue Hong et al., 2023.

⁵ <https://fairease.eu/>

⁶ Consortium of European Social Science Data Archives. <https://www.cessda.eu/>

2 Description of the Milestone

2.1 Role of the milestone

This milestone provides a proof of concept implementation of the FAIR4RS metric assessments (Chue Hong *et al*, 2023) as practical automatic tests as an extension to the F-UJI tool for FAIRsFAIR research data object assessment (Devaraju and Huber, 2020). The domain-agnostic implementations allow users from any scientific domain to assess the FAIRness of their research software. The discipline-specific implementation for metric tests as proposed by the Consortium of European Social Science Data Archives (CESSDA)⁷ further allows research software from the social sciences to test their compliance with CESSDA guidelines⁸.

2.1.1 Means of verification

The implementation is publicly available on GitHub⁹ and has been merged into the original F-UJI repository¹⁰ maintained by PANGAEA (Felden *et al*, 2023).

⁷ <https://www.cessda.eu/>

⁸ <https://docs.tech.cessda.eu/>

⁹ <https://github.com/FAIR-IMPACT/fuji>

¹⁰ <https://github.com/pangaea-data-publisher/fuji>

3 Background information

The FAIR and FAIR4RS principles are stated as broad principles, which are advisory and not prescriptive. To evaluate compliance with each principle in an automated manner a set of abstract criteria have been or are being developed to assess whether a digital object, such as a dataset or research software, complies with the principle. These abstract criteria are called metrics and each principle might have one or more of these to assess compliance. For instance, for the principle “A1.1: The protocol [used to access the data or research software] is open, free, and universally implementable”, a metric that would satisfy this principle would be to check that the data/research software is accessible using the HTTP(S) protocol. A test could then be designed and implemented to check that the digital object can be accessed using HTTP(S). The results from these tests, and implementations of metrics, can then be collected together and used to determine whether a FAIR principle is being observed or give guidance to the end user as to what changes would improve the level of compliance.

3.1 Metric structure

Metrics are used to translate the FAIR guiding principles into practical tests to measure the FAIRness of a digital object. This report focuses on FAIR4RS metrics (FRSM, Chue Hong *et al*, 2023), developed for research software, and FAIRsFAIR (FsF) metrics (Devaraju *et al*, 2022) for datasets. Both follow a similar structure, as shown in Tables 1 and 2.

Table 1: Structure of a FAIR4RS metric (Chue Hong *et al*, 2023).

Field	Description	
Metric Identifier	The local identifier of the metric (FRSM-XX).	
Metric Name	Metric name in a human readable form.	
Description	The definition of the metric, including examples.	
FAIR4RS principle	The FAIR4RS principle(s) most related to the metric.	
Assessment	Requirements and methods to perform the assessment against the metric with respect to three compliance levels.	
	Compliance level	Metric test
	<i>Essential</i>	Requirements to assess compliance with the metric on an <i>essential</i> level.
	<i>Important</i>	Requirements to assess compliance with the metric on an <i>important</i> level.
	<i>Useful</i>	Requirements to assess compliance with the metric on a <i>useful</i> level.

Table 2: Structure of an FsF metric (Devaraju *et al*, 2022).

Field	Description		
Metric Identifier	The local identifier of the metric following the FsF naming convention.		
Metric Name	Metric name in a human readable form.		
Description	The definition of the metric, including examples.		
FAIR principle	The FAIR principle most related to the metric.		
Assessment	Requirements and methods to perform the assessment against the metric with respect to three compliance levels, and the score given upon passing a test.		
	Compliance level	Metric test	Score
	1	Requirements to assess compliance with the metric on compliance level 1.	
	2	Requirements to assess compliance with the metric on compliance level 2.	
	3	Requirements to assess compliance with the metric on compliance level 3.	

Each metric is assessed by evaluating a series of *metric tests*. Each metric test evaluates the compliance with the metric on a specific level. The FAIR4RS metrics are defined with respect to compliance levels *essential*, *important* and *useful*, which can be mapped directly to the compliance levels 1, 2 and 3 referenced in the FsF metrics.

The FsF metrics explicitly define a score given upon passing a test to allow assigning different weights to a metric test with respect to the overall score achieved by a dataset.

4 Implementation of practical tests for FAIR Research Software Metrics in F-UJI

F-UJI is a tool for the automatic assessment of FAIRness that uses the FAIRsFAIR metrics proposed in Devaraju *et al*, 2022. It was initially developed to assess the FAIRness of research data objects, by Devaraju & Huber, 2021. The automatic assessment process is split into two stages: harvesting and evaluation, as shown in *Figure 1*.

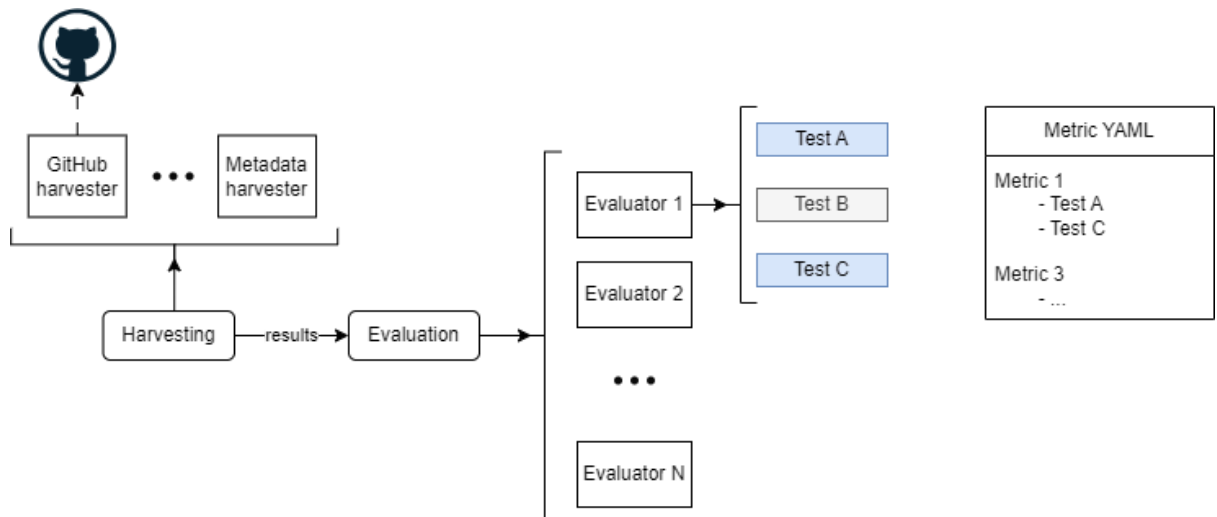


Figure 1: F-UJI workflow. Only tests A and C are listed for metric 1 in the configuration file, so test B is not run.

First, multiple “harvesters” collect data and metadata from various sources. The results are then merged and handed over to the evaluation stage. Each evaluator checks the compliance with a metric by running a series of test functions.

The metric configuration file (Metric YAML) is used to define the relevant details for all metrics. An example of a metric definition is shown in the appendix (*Listing A.1*). Each metric definition contains a list of metric tests with an associated test score and maturity level indicating the level of compliance with the metric. Each evaluator test function is associated with a metric test identifier. In our extension, a test can be mapped to multiple test identifiers from different metric sources to allow the reuse of similar implementations. Each test function starts by verifying that one of its identifiers is listed in the metric configuration file. If no identifier is listed, the test fails immediately. The test score is not added to the overall score. Otherwise, it carries out the implemented test. If the test passes, the configured test score is added to the overall evaluation score, and the maturity level is recorded.

4.1 Extending F-UJI for FAIR4RS

In the F-UJI framework all the information required for the assessment is first collected by harvesters. To parse the necessary data from software repositories, we needed to add a new harvester for the relevant sources. We decided to focus on GitHub for the time being as it is

a widely used platform in research software and we had prior experience with using the GitHub API¹¹. Adding support for other platforms would require only the addition of another harvester, e.g. for Zenodo¹² or GitLab¹³, as the metric evaluation is decoupled from the harvesting. Documentation and standardisation regarding what data is collected and in what structure it should be provided needs to be in would make the addition of new harvesters and reuse of existing evaluators for new metrics more straightforward, avoiding complicated nested harvesting results with duplicated metadata entries.

Chue Hong *et al*, 2023 proposed two sets of FAIR4RS metrics (FRSM), namely a generic set of metrics that are agnostic to any subject domain, and a CESSDA-specific set that takes into account domain-specific requirements and metadata. We added support for both to F-UJI using two new YAML files¹⁴. The metric configuration using YAML files was introduced in F-UJI version 3.0.0. This made it very easy to add our metrics and specify requirements for discipline-specific tests. Especially the option to configure test requirements, e.g. to specify a community standard, was useful for shaping the behaviour of domain-specific tests.

Without our extension, F-UJI only recognises metrics with identifiers that start with “FsF”¹⁵. This is tested with regular expressions in several places throughout the code, which all needed to be updated to recognise our metric identifier scheme starting with “FRSM”. To simplify the addition of future metric schemes, it would be beneficial to refactor this, for example by making the metric identifier pattern configurable. The regular expressions are also used to extract the FAIR principle; the same information is also included in the metric configuration file and can be read from there. This suggests that there is potential to ease the addition of other metric sets simply by leveraging the already existing YAML files.

The next step was to add capability for evaluating the new metrics. Linking our metrics to evaluator objects was not possible through the metrics file. Instead, every existing evaluator that we wanted to reuse required additional logic to allow more than one metric identifier to be associated with it (e.g. FsF-F2-01M and FRSM-04-F2 for minimal metadata). Moreover, test functions are also associated with metric test descriptions in the code base. This mapping cannot be set from the configuration file, which would make test reuse much easier for less technical users. Again, we were able to reuse some test functions and added the remaining ones.

Some of the FAIR4RS metrics required the addition of new evaluators, for example FRSM-13 (software requirements). Adding a new evaluator class is straightforward based on the parent class FAIREvaluator. However, in order for it to be called, additional code changes are required in multiple classes. We added documentation detailing these required changes, but ideally, the addition could be streamlined. One possibility would be refactoring to a factory pattern, where each new evaluator is registered with a factory class which ensures that they all get called and return the results as expected.

¹¹ <https://docs.github.com/en/rest>

¹² <https://zenodo.org/>

¹³ <https://about.gitlab.com/>

¹⁴ YAML is a human readable data serialisation language. (<https://yaml.org/>)

¹⁵ FsF is the acronym used for the FAIRsFAIR metrics that F-UJI was originally developed for.

The F-UJI implementation expects each metric to be mapped to one FAIR principle. This is not always the case for the FAIR4RS metrics, they often relate to multiple principles. In these cases, we decided to choose one principle as the main guiding principle for the metric.

Table 3 lists the mapping from FAIR4RS metrics to their main principle, as well as the F-UJI evaluator class that was reused or added. For those that were reused, the FsF metric for which the evaluator was originally designed is stated.

Table 3: Mapping of FAIR4RS metrics to F-UJI evaluators and their FsF metrics.

FRSM metric	FAIR4RS principle (main in bold)	F-UJI evaluator (reused or added)	FsF metric
FRSM-01: Does the software have a globally unique and persistent identifier?	F1 , R3	FAIREvaluatorUniquePersistentIdentifierSoftware (added)	n/a
FRSM-02: Do the different components of the software have their own identifiers?	F1.1 , F1	FAIREvaluatorSoftwareComponentIdentifier (added)	n/a
FRSM-03: Does each version of the software have a unique identifier?	F1.2 , F1	FAIREvaluatorVersionIdentifier (added)	n/a
FRSM-04: Does the software include descriptive metadata which helps define its purpose?	F2 , R1, R3	FAIREvaluatorCoreMetadata (reused)	FsF-F2-01M: Metadata includes descriptive core elements (creator, title, data identifier, publisher, publication date, summary and keywords) to support data findability.
FRSM-05: Does the software	R1 , F2, R3	FAIREvaluatorDevelopmentMetadata (added)	n/a

include development metadata which helps define its status?			
FRSM-06: Does the software include metadata about the contributors and their roles?	F2 , R3	FAIREvaluatorDataProvenance (reused)	FsF-R1.2-01M: Metadata includes provenance information about data creation or generation.
FRSM-07: Does the software metadata include the identifier for the software?	F3 , R3	FAIREvaluatorDataIdentifierIncluded (reused)	FsF-F3-01M: Metadata includes the identifier of the data it describes.
FRSM-08: Does the software have a publicly available, openly accessible and persistent metadata record?	F4 , A2, R3	FAIREvaluatorMetadataPreserved (reused)	FsF-A2-01M: Metadata remains available, even if the data is no longer available.
FRSM-09: Is the software developed in a code repository / forge that uses standard communications protocols?	A1 , A1.1, A1.2, R3	FAIREvaluatorStandardisedProtocolData (reused)	FsF-A1-03D: Data is accessible through a standardized communication protocol.
FRSM-10: Are the formats used by the	I1 , I2	FAIREvaluatorFileFormat (reused)	FsF-R1.3-02D: Data is available in a file format recommended

data consumed or produced by the software open and a reference provided to the format?			by the target research community.
FRSM-11: Does the software use open APIs that support machine-readable interface definition?	I1	FAIREvaluatorAPI (added)	n/a
FRSM-12: Does the software provide references to other objects that support its use?	I2	FAIREvaluatorRelatedResources (reused)	FsF-I3-01M: Metadata includes links between the data and its related entities.
FRSM-13: Does the software describe what is required to use it?	R1, R2	FAIREvaluatorRequirements (added)	n/a
FRSM-14: Does the software come with test cases to demonstrate it is working?	R1	FAIREvaluatorTestCases (added)	n/a
FRSM-15: The software source code includes licensing information for the software and any bundled	R1.1	FAIREvaluatorLicenseFile (added)	n/a

external software.			
FRSM-16: Does the software metadata record include licensing information?	R1.1	FAIREvaluatorLicense (reused)	FsF-R1.1-01M: Metadata includes licence information under which data can be reused.
FRSM-17: Does the software include provenance information that describe the development of the software?	R1.2	FAIREvaluatorCodeProvenance (added)	n/a

With these changes, all FAIR4RS metrics have a corresponding evaluator and set of test functions in F-UJI. Not all tests are implemented. Those that are not implemented are present in the tool as skeleton functions and add a warning message indicating this to the user as shown in *Figure 2*.

FRSM-13-R1 - Does the software describe what is required to use it?
✓

FRSM-14-R1 - Does the software come with test cases to demonstrate it is working?
?

FAIR level:

0 of 3

Score:

0 of 3

Output:

[]

incomplete

Metric tests:

Test:	Test name:	Score:	Maturity:	Result:
FRSM-14-R1-1	Tests and data are provided to check that the software is operating as expected.			?
FRSM-14-R1-2	Automated unit and system tests are provided.			?
FRSM-14-R1-3	Code coverage / test coverage is reported.			?

Debug messages:

Level:	Message:
WARNING	Test for presence of tests and test data is not implemented.
WARNING	Test for Automated unit and system tests is not implemented.
WARNING	Test for code coverage is not implemented.
INFO	This test is not defined in the metric YAML and therefore not performed: FRSM-14-R1-CESSDA-1
INFO	This test is not defined in the metric YAML and therefore not performed: FRSM-14-R1-CESSDA-2
INFO	This test is not defined in the metric YAML and therefore not performed: FRSM-14-R1-CESSDA-3
WARNING	Failed to check the software version identifier.

FRSM-15-R1.1 - The software source code includes licensing information for the software and any bundled external software.
✓

FAIR level:

3 of 3

Score:

2 of 3

Output:

[
{
"license": "Apache License 2.0",
"os_i_approved": true,
"details_url": "http://\V\spdx.org/licenses/Apache-2.0.html"
}
]

advanced

Metric tests:

Test:	Test name:	Score:	Maturity:	Result:
FRSM-15-R1.1-1	License file is included.	1	1	✓
FRSM-15-R1.1-2	The source code includes licensing information for all components bundled with that software.			?
FRSM-15-R1.1-3	Recognized licence is in SPDX format.	1	3	✓

Debug messages:

Level:	Message:
INFO	License verification name through SPDX registry -: Apache License 2.0
INFO	Found SPDX license representation -: http://spdx.org/licenses/Apache-2.0.json
SUCCESS	Found SPDX license representation (spdx url, osi_approved)
SUCCESS	Found licence file: ['LICENSE'].
INFO	Will consider all SPDX licenses as community specific licenses for FRSM-15-R1.1
INFO	This test is not defined in the metric YAML and therefore not performed: FRSM-15-R1.1-CESSDA-1
WARNING	Test for license information of bundled components is not implemented (FRSM-15-R1.1-2).
INFO	This test is not defined in the metric YAML and therefore not performed: FRSM-15-R1.1-CESSDA-3
INFO	This test is not defined in the metric YAML and therefore not performed: FRSM-15-R1.1-CESSDA-2

FRSM-16-R1.1 - Does the software metadata record include licensing information?
?

Figure 2: Web client view including tests that are not implemented.

4.2 Test implementations

As a proof-of-concept, we fully implemented the FAIR4RS metrics 13 and 15, both the generic tests and the CESSDA-specific tests. The metrics and corresponding tests, proposed in Chue Hong *et al*, 2023, are included below for reference, and implementation notes have been added (see *Tables 4 to 7*).

Some of the tests were straightforward to implement, like the generic test FRSM-15-useful (*‘The recognised licence is in SPDX format’*). The required data are clearly stated (the licence name), and the condition for passing the test is specific: the licence name must be on the SPDX licence list.

Other tests were vague in terms of scope or strictness, or impossible to check automatically. For example, we were not able to implement the generic test FRSM-15-important (*‘The source code includes licensing information for all components bundled with that software’*), as it required solving two hard problems: automatically recognising bundled components, and mapping the content of the software licence to see whether it covers all those bundled components.

The CESSDA-specific test FRSM-15-CESSDA-useful asks that *‘The build script (Maven POM, where used) checks that the standard header is present in all source code files.’*. Maven POM checks this using a plugin, *license-maven-plugin*, which can be configured to fail if not all code files contain the standard header. Otherwise, it just displays a warning. The expected strictness of the metric test is unclear: is it enough for the plugin to be included for the test to pass, or is the additional configuration expected? On the other hand, FRSM-15-CESSDA-essential (*‘Include a LICENSE.txt file in the root of the repository.’*) is unnecessarily strict for evaluating GitHub repositories. The licence files generated by GitHub do not have a *‘.txt’* file extension, but arguably, the compliance with the metric isn’t increased by adding that extension after creation.

When the scope of a metric test is unclear, it is difficult to decide how many aspects need to be considered. For example, the CESSDA-specific test FRSM-13-CESSDA-essential asks that *‘linting and other relevant checks are present in the automated build and test process (e.g. via the Jenkinsfile)’*. As the *‘other checks’* are not specified, we did not consider additional checks apart from linting.

The ability to configure test requirements in the metrics YAML file makes it easy to extend the scope of such vague tests as they become more specific over time, for example, to cater to a variety of tools and programming languages.

Table 4: Description of the generic FSM-13.

Field	Description	
Metric Identifier	FRSM-13	
Metric Name	Does the software describe what is required to use it?	
Description	Software is made more reusable by providing suitable machine-actionable information on dependencies, build and configuration.	
Metric Tests	<i>Essential</i>	The software has build, installation and/or execution instructions

	<i>Important</i>	Dependencies are provided in a machine-readable format and the building and installation of the software is automated.
	<i>Useful</i>	N/A
Notes from Implementation	<p>FRSM-13-Essential:</p> <ul style="list-style-type: none"> - Scope: <ul style="list-style-type: none"> - Where in a repository are instructions expected? The current implementation considers the README file and files in a 'docs/' folder. More locations can be configured. - How can instructions be recognised? The current implementation looks for a list of configurable keywords like 'build'. <p>FRSM-13-Important:</p> <ul style="list-style-type: none"> - Scope: <ul style="list-style-type: none"> - There is a wide variety of machine-readable dependency formats. The current implementation checks the presence of files from a configurable list of such formats, for example a "requirements.txt" file as commonly used in Python projects. - A variety of automation tools is available. The current implementation considers Jenkins and GitHub actions. - Strictness: It would be difficult to check whether the dependencies are up to date and whether the automation tool covers the entire piece of software. 	

Table 5: Description of the CESSDA-specific version of FSM-13.

Field	Description	
Metric Identifier	FRSM-13-CESSDA	
Metric Name	Does the software describe what is required to use it?	
Metric Tests	<i>Essential</i>	Dependency information and build instructions are included in the README file. Linting and other relevant checks are present in the automated build and test process (e.g. via the Jenkinsfile).
	<i>Important</i>	The README file includes a badge that links to the automated build tool (Jenkins). Deployment to development and staging environments is automated (conditional on test results).
	<i>Useful</i>	The build badge indicates the status of the latest build (passing or failing)
Notes from Implementation	<p>Both FRSM-13-CESSDA-Important and FRSM-13-CESSDA-Useful both test for badges. The difference in the current implementation is:</p> <ul style="list-style-type: none"> - FRSM-13-CESSDA-Important looks for a badge that links to some Jenkins (or other automated tool) page. 	

	<ul style="list-style-type: none"> - FRSM-13-CESSDA-<i>Useful</i> looks for a badge that links to a job status. <p>FRSM-13-CESSDA-<i>Essential</i>:</p> <ul style="list-style-type: none"> - Difficult to implement: It is unclear how linting checks are defined in a Jenkinsfile. We could not find documentation on whether this test should look for a specific plugin. - Scope: What are “other relevant checks”? The current implementation does not consider checks other than linting, but keywords can be added to the test requirements later on. <p>FRSM-13-CESSDA-<i>Useful</i>:</p> <ul style="list-style-type: none"> - Difficult to implement: The test requires distinguishing between multiple environments. Searching for a section in the tool configuration that implements that is not straightforward as there are various ways to do so. Instead, the test currently just looks for the keyword “<i>deploy</i>” in an automation tool configuration file (again using test requirements for keyword search).
--	---

Table 6: Description of the generic FSM-15.

Field	Description	
Metric Identifier	FRSM-15	
Metric Name	Does the software source code include licensing information for the software and any bundled external software?	
Description	Clear software licensing enables reuse.	
Metric Tests	<i>Essential</i>	The software includes its LICENCE file.
	<i>Important</i>	The source code includes licensing information for all components bundled with that software.
	<i>Useful</i>	The software licensing information is in SPDX format.
Notes from Implementation	<p>We initially extended FsF-R1.1-01M, but found this was more aligned with FRSM-16 (licence in metadata).</p> <p>FRSM-15-<i>Important</i>:</p> <ul style="list-style-type: none"> - Difficult to implement: Checking component licence requires an automated way of checking for components (=bundled dependencies), which is less straightforward. 	

Table 7: Description of the CESSDA-specific version of FSM-15.

Field	Description	
Metric Identifier	FRSM-15-CESSDA	
Metric Name	Does the software source code include licensing information for the software and any bundled external software?	
Metric Tests	<i>Essential</i>	Include a LICENSE.txt file in the root of the repository.

	<i>Important</i>	Include licensing information in the source code header.
	<i>Useful</i>	The build script (Maven POM, where used) checks that the standard header is present in all source code files.
Notes from Implementation	<p>These tests do not check bundled components, even though they are explicitly mentioned in the metric name. Is it important that bundled external software licensing is known to be FAIR? Do the principles apply to software as well or just generally to digital objects?</p> <p>FRSM-15-CESSDA-<i>Essential</i>:</p> <ul style="list-style-type: none"> - Strictness: The TXT suffix is not present in licence files automatically generated by GitHub. The current implementation does therefore not fail when the suffix is missing. However, the CESSDA guidance does include this requirement. Note also that alternative spellings like LICENCE are not recognised in this test. <p>FRSM-15-CESSDA-<i>Important</i>:</p> <ul style="list-style-type: none"> - Strictness: <ul style="list-style-type: none"> - What should be recognised as a licence header? The current implementation checks the first 30 lines of source code files for the word “license”. This is reasonably meaningful, but a more explicit definition after prior investigation about what licence headers usually look like would be preferable. - Do all source code files need to be checked? The current implementation parses a sample of five source code files. <p>FRSM-15-CESSDA-<i>Useful</i>:</p> <ul style="list-style-type: none"> - Strictness: Should the test fail if the Maven plugin is not configured to stop the build if headers are missing? - Scope: What other tools are to be expected and tested? 	

Each metric test is assigned a test score. For the FAIR4RS metrics, we decided to assign each test the same score of 1 so as not to give some metric tests a higher weight. F-UJI also expects a maturity level for each metric test, indicating the level of compliance with the metric assured by the test. We mapped this to the FRSM compliance levels as outlined in *Table 8*. In F-UJI, the overall level of maturity reached for a metric after the evaluation is determined as the highest maturity level from all passed tests of that metric. However, the metric tests both in the FRSM and the FsF framework are defined with respect to compliance levels, not maturity levels. A test for the level ‘*useful*’ can pass even when the tests for ‘*essential*’ and ‘*important*’ fail, as the requirements for each level do not have to build on each other. This results in unexpected behaviour during evaluation with F-UJI: The overall reported maturity level for that metric will be 3 (*advanced*), even though the test for compliance level ‘*essential*’ failed. We did not modify this behaviour as we believe a wider discussion is necessary to determine how maturity levels should be assessed.



Table 8: Mapping of compliance and maturity levels.

Compliance level as used in FAIR4RS metrics	Maturity level as expected in F-UJI
Essential	1 (initial)
Important	2 (moderate)
Useful	3 (advanced)

The focus of the domain-specific tests for CESSDA often differs from that of the generic tests, as is expected for the discipline-specific versions because of the underlying varying standards in different communities. Sometimes, this is expressed as the requirement of a specific software stack, for example tests referring to Jenkins where the generic tests look at automation tools in general, including but not limited to Jenkins. We also encountered different interpretations of the metric. For example, FRSM-15 looks for licensing information in both the software and any bundled external software components. The generic tests broadly cover bundled components in the ‘important’ test. The CESSDA-specific tests instead check for a licence header in all source code files. This might cover components bundled as source code files, but not components bundled as binary files. While the aim of the FAIR4RS metrics is to provide a practical translation of the principles, these kinds of discrepancies might hinder the understanding of how to comply with the FAIR principles.

Overall, the CESSDA tests tend to be more specific, which makes them easier to assess automatically. The generic tests on the other hand are designed to leave room for various processes and tools, therefore requiring automatic assessment to look for a variety of different artefacts. The possibility of defining test requirements in the metric configuration file is helpful here, as the accepted artefacts can grow over time, and the selection can be narrower in a disciplinary context.

A difficulty in both the generic tests and the CESSDA tests is that the strictness is often unclear. Should a test pass when keywords are found, or should the content be checked more rigorously? When formulating tests for a disciplinary context, it could be valuable to specify technical details, for example the keywords, plugins or code snippets to look for in a file. In our implementation, we decided to opt for the least strict interpretation for each test, as we are aiming to encourage software developers to comply with the metrics. Ensuring that every deviation from the metrics is found complicates the automatic assessment implementation and might obfuscate clear actions that developers can take to improve the FAIRness of their software.

5 Evaluation of tests in a disciplinary context

We tested the automatic assessment of research software in a disciplinary context using a set of 33 repositories provided by CESSDA. We evaluated them against the two implemented FAIR4RS metrics described above, FRSM-13 and FRSM-15, using the CESSDA-specific tests. The results of the automatic assessment are shown in *Figure 3*. Note that the test identifier nomenclature used here is adapted from the identifier scheme F-UJI expects: it includes the FAIR principle and ends with a test number instead of the compliance level. For each metric, test number 1 corresponds to the test for “*essential*”, number 2 corresponds to “*important*”, and number 3 to “*useful*”. This is also compliant with the maturity level mapping.

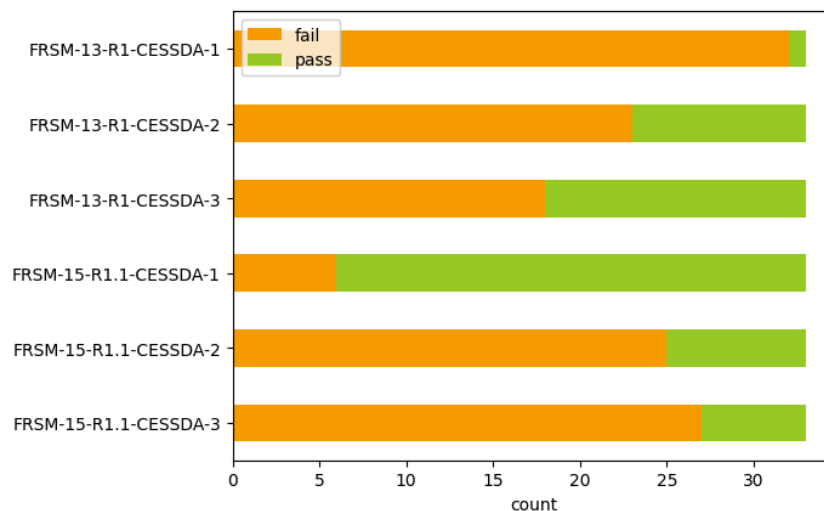


Figure 3: Evaluation of 33 CESSDA repositories against CESSDA-specific metric tests.

Interestingly, the number of repositories that pass the FRSM-13 tests increases with a higher compliance level. This might be because, as outlined in section 3, the scope of the lower compliance level tests was in parts vague. The corresponding proof-of-concept implementation might be too strict, disregarding other options for complying with the test.

We manually verified the assessment results for two repositories. We found that the results for FRSM-15 were as expected both with respect to the metric test definition as proposed by Chue Hong *et al*, 2023, as well as with respect to the practical implementation of these tests, confirming that the implementations appropriately match the definition. However, we observed that one of the repositories did not match the expected software stack (Java and Maven), so any checks for licence headers in the build script would not be recognised by F-UJI. We could not determine whether any such checks were included in the build process. Further investigation would be needed to add support for other such checks, as well as a discussion whether they should be recognised as in line with the CESSDA guidelines.

The verification of the results for FRSM-13 showed that the implementation for FRSM-13-CESSDA-*essential* (FRSM-13-R1-CESSDA-1) does not use a large enough vocabulary to recognise information about dependencies in the README file, as it did not pick up on keywords such as “prerequisites”. This metric test also requires linting checks in the Jenkinsfile. Both repositories use SonarQube in their Jenkinsfile, a code review tool which

can provide linting information. It is unclear whether the call in the Jenkinsfile does indeed perform linting checks. Further input from CESSDA might clarify whether this is a community standard that should be recognised to conform with the metric. The verification of FRSM-13-CESSDA-*useful* (FRSM-13-R1-CESSDA-3) highlighted an ambiguity in the metric test definition. In both cases, the links from the README file to the Jenkins build job lead to an inaccessible web page, and one of the badges displayed the status “not run” instead of “fail” or “pass”. The metric test definition does not clearly state whether the badge should lead to an accessible web page.

These results can inform further improvements of the implemented FRSM tests and inform any potential refinement of the metric test definitions. Detailed notes are included in the appendix (Table A.3).

We also evaluated a small set of repositories against the domain-agnostic metric tests. FAIR-EASE provided us with a list of nine software objects of which four hosted their source code on GitHub. The results of evaluating these four repositories against FRSM-13 and FRSM-15 are shown in Figure 4. Note that the domain-agnostic metric tests for FRSM-13 do not include a test of level “*useful*,” so only two tests are listed. FRSM-15-*important* (FRSM-15-R1.1-2) was not implemented as explained above, so it automatically fails.

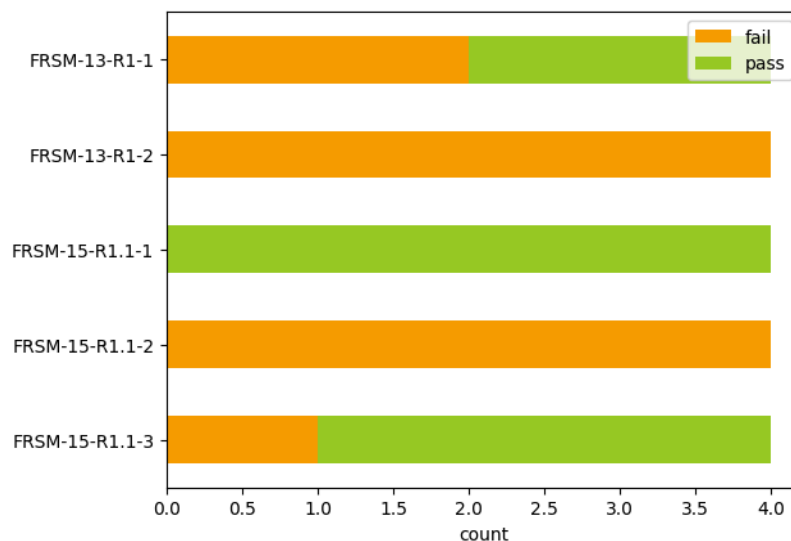


Figure 4: Evaluation of 4 FAIR-EASE repositories against domain-agnostic metric tests.

Notably, FRSM-15-*essential* (FRSM-15-R1.1-1), which checks the inclusion of a licence file, is passed for all repositories. On the other hand, FRSM-13-*important* (FRSM-13-R1-2) was not passed for any of them, which could be due to the narrow implementation. There are a multitude of machine-readable formats for dependencies, and the current implementation only considers a small set as proof of concept.

Again, we manually verified the results for two of the repositories. This confirmed that further machine-readable formats should be taken into account for FRSM-13-*important* as one of the repositories included a Maven POM file that is not recognised by the current implementation.

The evaluation results for each software repository are listed in the appendix of this document (*Tables A.1 and A.2*).

6 Conclusions and next steps

We extended F-UJI to automatically assess FAIR4RS metrics. The process was mostly straightforward due to the modularity of the tool design. Our contributions to the documentation specifically address the addition of new metrics and expanding existing test functions, which will lower the entry barrier for other contributors. Small refactoring steps in F-UJI, as well as keeping this documentation updated and easily accessible, will allow a wider community to contribute their own discipline-specific metric implementation.

We found that defining metric tests with clear technical requirements with respect to scope and strictness eases the implementation. Discipline-specific tests tend to have a more narrow definition, for example requiring specific software stacks, whereas the generic tests are expected to allow for a variety of possible tools and techniques. The discipline-specific tests are thus easier to implement. As our implementation is a proof-of-concept however, the practical implementations of the generic tests don't always cover the full scope allowed in the metric test description. The returned score is thus often lower than it would be if support for more tools or file formats was added. Input from the wider community on such tools and formats will be vital to the future improvement of these practical implementations. Fortunately, due to the high degree of configurability in F-UJI, the implementation can be extended to cover a wider range without major code changes.

The strictness of discipline-specific metric tests raises a question about the expected next steps for the developers of the research software. If they achieve a low score with respect to the discipline-specific metric tests because they chose a different mechanism to comply with a metric than widely used in the discipline, should they switch to the expected mechanism? Or should the discipline-specific metric tests act as recommendations, to point developers to commonly used mechanisms in case the software object is not at all compliant with the metric? The former might be discouraging for the developers, instead of helping them improve the FAIRness of their software. In the latter case, it might be more helpful to use the discipline-specific tests for discipline-tailored feedback rather than for assessment. These considerations are to be discussed in the wider FAIR-IMPACT project and inform future activities.

Our contributions have been merged back into the original F-UJI repository, maintained by PANGAEA. The next version release will include our changes and make the implemented tests available through their web client. Our extension will be presented in a workshop session at Collaborations Workshop 2024¹⁶ followed by a discussion about the practical implementation of the metric tests, addressing some of the challenges we observed with a wider community. The updated version of F-UJI will also be used in the FAIR-IMPACT Support Action "Assessing and improving existing research software"¹⁷, which, depending on the

¹⁶ Organised by the Software Sustainability Institute (SSI), more information available at <https://www.software.ac.uk/workshop/collaborations-workshop-2024-cw24>

¹⁷ <https://fair-impact.eu/support-offer-1-assessment-and-improvement-research-software>

participants' research backgrounds, should allow a broader evaluation on a wider range of research software objects. Both events will provide valuable insights into existing mechanisms and the expectations developers have regarding the automated assessment. The results will inform future improvements.

7 References

- Chue Hong, N. P., Katz, D. S., Barker, M., Lamprecht, A.-L., Martinez, C., Psomopoulos, F. E., Harrow, J., Castro, L. J., Gruenpeter, M., Martinez, P. A., Honeyman, T., Struck, A., Lee, A., Loewe, A., van Werkhoven, B., Jones, C., Garijo, D., Plomp, E., Genova, F., ... RDA FAIR4RS WG. (2022). FAIR Principles for Research Software (FAIR4RS Principles) (1.0). Zenodo. <https://doi.org/10.15497/RDA00068>
- Chue Hong, N., Breitmoser, E., Antonioletti, M., Davidson, J., Garijo, D., Gonzalez-Beltran, A., Gruenpeter, M., Huber, R., Jonquet, C., Priddy, M., Shepeherdson, J., Verburg, M., & Wood, C. (2023, October 27). *D5.2 - Metrics for automated FAIR software assessment in a disciplinary context*. Zenodo. <https://doi.org/10.5281/zenodo.10047401>
- Devaraju, A., & Huber, R. (2020). F-UJI - An Automated FAIR Data Assessment Tool. Zenodo. <https://doi.org/10.5281/zenodo.6361400>
- Devaraju, A., & Huber, R. (2021). An automated solution for measuring the progress toward FAIR research data. *Patterns*, 2(11), 100370. <https://doi.org/10.1016/j.patter.2021.100370>
- Devaraju, A., Huber, R., Mokrane, M., Herterich, P., Cepinskas, L., de Vries, J., L'Hours, H., Davidson, J., & Angus White. (2022). FAIRsFAIR Data Object Assessment Metrics (0.5). Zenodo. <https://doi.org/10.5281/zenodo.6461229>
- Felden, J., Möller, L., Schindler, U. et al. *PANGAEA - Data Publisher for Earth & Environmental Science*. *Sci Data* 10, 347 (2023). <https://doi.org/10.1038/s41597-023-02269-x>

8 Appendices

8.1 Example of a metric definition in F-UJI using YAML

```
- metric_identifier: FRSM-15-R1.1
  metric_number: 15
  metric_short_name: Software Source Code License
  metric_name: The software source code includes licensing
    information for the software and any bundled external software.
  description: It is important that software licences are included
    with the source code as many tools and processes look for licensing
    information there to determine licence compatibility.
  fair_principle: R1.1
  target: Software
  evaluation_mechanism: Metric evaluation is based on the presence of
    a machine readable license file.
  test_scoring_mechanism: cumulative
  metric_tests:
    - metric_test_identifier: FRSM-15-R1.1-1
      metric_test_name: License file is included.
      metric_test_score: 1
      metric_test_maturity: 1
    - metric_test_identifier: FRSM-15-R1.1-2
      metric_test_name: The source code includes licensing
        information for all components bundled with that software.
      metric_test_score: 1
      metric_test_maturity: 2
    - metric_test_identifier: FRSM-15-R1.1-3
      metric_test_name: Recognized licence is in SPDX format.
      metric_test_score: 1
      metric_test_maturity: 3
  created_by: FAIR4RS
  date_created: 2023-11-10
  date_updated: 2023-12-13
  version: 0.1
  total_score: 3
```

Listing A.1: YAML definition of a metric.

8.2 Repository-level evaluation results

Table A.1: Results for domain-agnostic evaluation of FAIR-EASE software repositories.

URL	FRSM-13 -R1-1	FRSM-13 -R1-2	FRSM-15 -R1.1-1	FRSM-15 -R1.1-2	FRSM-15 -R1.1-3
https://github.com/HCBScienceProducts/CANY ON-B	fail	fail	pass	fail	pass
https://github.com/ESSI-Lab/DAB	pass	fail	pass	fail	pass

https://github.com/gher-uliege/DIVAnd.jl	fail	fail	pass	fail	pass
https://github.com/Geomatys/examind-community	pass	fail	pass	fail	fail

Table A.2: Results for CESSDA-specific evaluation of CESSDA software repositories.

URL	FRSM-13-R1-CESSDA-1	FRSM-13-R1-CESSDA-2	FRSM-13-R1-CESSDA-3	FRSM-15-R1.1-CESSDA-1	FRSM-15-R1.1-CESSDA-2	FRSM-15-R1.1-CESSDA-3
https://github.com/cessda/cessda.cvs.contentguide	fail	fail	fail	pass	fail	fail
https://github.com/cessda/cessda.metadata.profiles	fail	fail	fail	pass	fail	fail
https://github.com/cessda/cessda.cdc.versions	fail	fail	fail	pass	fail	fail
https://github.com/cessda/cessda.cafe.waiter	fail	pass	pass	pass	fail	fail
https://github.com/cessda/main	fail	fail	fail	fail	fail	fail
https://github.com/cessda/cessda.cdc.aggregator.oai-pmh-repo-handler	fail	fail	pass	pass	fail	fail
https://github.com/cessda/cessda.cafe.coffee.carsten	fail	fail	fail	pass	fail	fail
https://github.com/cessda/cessda.cdc.aggregator.doc-store	fail	fail	pass	pass	fail	fail
https://github.com/cessda/cessda.documentation.theme	fail	fail	fail	pass	fail	fail
https://github.com/cessda/cessda.metadata.harvester	fail	pass	pass	pass	pass	pass
https://github.com/cessda/cessda.training-resources.issues	fail	fail	fail	fail	fail	fail
https://github.com/cessda/cessda.cmv	fail	pass	pass	pass	fail	fail
https://github.com/cessda/cessda.code.nesstar	fail	fail	fail	pass	fail	fail
https://github.com/cessda/cessda.twitter.api.stats	fail	fail	pass	pass	fail	fail

https://github.com/cessda/cessda.cmv.console	fail	pass	pass	pass	pass	pass
https://github.com/cessda/cessda.cdc.aggregator.client	fail	fail	pass	pass	pass	fail
https://github.com/cessda/cessda.resource-directory.issues	fail	fail	fail	fail	fail	fail
https://github.com/cessda/cessda.cvs.userguide	fail	fail	fail	pass	fail	fail
https://github.com/cessda/cessda.cmv.documentation	fail	fail	fail	pass	fail	fail
https://github.com/cessda/cessda.cdc.osmh-indexer.cmm	fail	pass	pass	pass	pass	pass
https://github.com/cessda/cessda.code.dataverse	fail	fail	fail	pass	fail	fail
https://github.com/cessda/cessda.cmv.core	fail	pass	pass	pass	pass	pass
https://github.com/cessda/cessda.cdc.fuji.runner	fail	pass	pass	pass	fail	fail
https://github.com/cessda/cessda.cdc.aggregator.shared-library	fail	fail	pass	pass	fail	fail
https://github.com/cessda/cessda.cdc.userguide	fail	fail	fail	pass	pass	fail
https://github.com/cessda/eqb.colectica.issues	fail	fail	fail	fail	fail	fail
https://github.com/cessda/cessda.cdc.aggregator.devguide	fail	fail	fail	fail	fail	fail
https://github.com/cessda/cessda.cvs.two	fail	pass	pass	pass	pass	pass
https://github.com/cessda/cessda.cdc.searchkit	pass	pass	pass	pass	fail	fail
https://github.com/cessda/cessda.cafe.cashier	fail	fail	fail	pass	fail	fail
https://github.com/cessda/cessda.dmeg-dag.issues	fail	fail	fail	fail	fail	fail
https://github.com/cessda/cessda.cmv.server	fail	pass	pass	pass	pass	pass

https://github.com/cessda/cessda.guidelines.public	fail	fail	fail	pass	fail	fail
---	------	------	------	------	------	------

Table A.3: Manual verification notes for CESSDA-specific evaluation. Discrepancies between the expected and observed outcome are highlighted in bold.

Software Object	Metric Test Identifier	Metric Test Outcome	Expected metric test outcome	Comment
CESSDA Café: Cashier ¹⁸	FRSM-13-R1-CESSDA-1	fail	unclear	Instead of “dependencies”, the README file lists “prerequisites”, which is not recognised during automatic assessment. It possibly should be recognised. The Jenkinsfile does not explicitly define a linting check, but it uses SonarQube, a code review tool that might include linting checks. This should possibly be recognised in the assessment. It is unclear what other checks should be included.
	FRSM-13-R1-CESSDA-2	fail	fail	The README includes a link to the Jenkins job, but not through a badge. Incidentally, the link leads to a “Server not found” error, which is not considered in the test definition but should possibly be. A Docker image is deployed automatically regardless of the results of tests. There is no distinction between different environments, e.g. development and staging. This part of the test implementation passes (though it should not), as it can only evaluate whether the Jenkinsfile includes a deployment stage.
	FRSM-13-R1-	fail	fail	There is no build badge.

¹⁸ <https://github.com/cessda/cessda.cafe.cashier>

	CESSDA-3			
	FRSM-15-R1.1-CESSDA-1	pass	pass	LICENSE.txt was found.
	FRSM-15-R1.1-CESSDA-2	fail	fail	No licence headers in the source code.
	FRSM-15-R1.1-CESSDA-3	fail	unclear	The project uses .NET, not Maven, so the implementation would not recognise checks for a licence header if there are any. We cannot determine if there are any.
CESSDA Metadata Validator Core ¹⁹	FRSM-13-R1-CESSDA-1	fail	fail	Dependencies are included but not recognised by the test as they are listed as “Dependency information”. There are no explicit build instructions, only instructions for test execution. The Jenkinsfile does not explicitly define a linting check, but it uses SonarQube, a code review tool that might include linting checks. This should possibly be recognised in the assessment. It is unclear what other checks should be included.
	FRSM-13-R1-CESSDA-2	pass	pass	There is a build badge.
	FRSM-13-R1-CESSDA-3	pass	unclear	The build badge points to an inaccessible Jenkins job (“Not found”), and it indicates that the build job has “not run”. It is unclear whether this qualifies for the required indication of the status of the latest build.
	FRSM-15-R1.1-CESSDA-1	pass	pass	LICENSE file was found (though not .txt).

¹⁹ <https://github.com/cessda/cessda.cmv.core>

	FRSM-15-R1.1-CESSDA-2	pass	pass	Licence headers are included.
	FRSM-15-R1.1-CESSDA-3	pass	pass	The Maven POM file includes the license-maven-plugin, and the build fails when headers are missing.

Table A.4: Manual verification notes for generic evaluation. Discrepancies between the expected and observed outcome are highlighted in bold.

Software Object	Metric Test Identifier	Metric Test Outcome	Expected metric test outcome	Comment
CANYON-B ²⁰	FRSM-13-R1-1	fail	fail	The README file does not include any instructions.
	FRSM-13-R1-2	fail	fail	The dependencies are not provided in a separate machine-readable format. Building and installation are not automated.
	FRSM-15-R1.1-1	pass	pass	A licence is included.
	FRSM-15-R1.1-2	fail	unclear	There are no bundled components. Possibly the test should pass. Currently, the test implementation is not able to recognise bundled components so the test fails automatically.
	FRSM-15-R1.1-3	pass	pass	The included GPL-3.0 licence is on the SPDX licence list. ²¹
Discovery and Access Broker (DAB) Community Edition (CE) ²²	FRSM-13-R1-1	pass	pass	The README includes instructions for installation, compilation, launching and building a Docker image.
	FRSM-13-R1-2	fail	fail	The repository includes a Maven POM file including

²⁰ <https://github.com/HCBScienceProducts/CANYON-B>

²¹ <https://spdx.org/licenses/>

²² <https://github.com/ESSI-Lab/DAB>

				<p>machine-readable dependencies. This is not recognised by the generic metric test implementation, but it should be.</p> <p>Build and installation do not seem to be automated.</p>
	FRSM-15-R1.1-1	pass	pass	Two licence files are included (LICENSE and LICENSE.txt). They seem to be identical.
	FRSM-15-R1.1-2	fail	unclear	It is unclear whether the software contains bundled components.
	FRSM-15-R1.1-3	pass	pass	The included AGPL-3.0 licence is on the SPDX licence list.