Knowledge Graphs in the Age of Language Models and Neuro-Symbolic AI
A. Salatino et al. (Eds.)
© 2024 The Authors.
This article is published online with Open Access by IOS Press and distributed under the terms
of the Creative Commons Attribution License 4.0 (CC BY 4.0).
doi:10.3233/SSW240014

Assessing the FAIRness of Software Repositories Using RDF and SHACL

Tobias HUMMEL [a] Leon MARTIN [a,1] Andreas HENRICH [a]
[a] *Media Informatics, University of Bamberg, An der Weberei 5, 96047 Bamberg.*
ORCiD ID: Tobias Hummel https://orcid.org/0009-0000-5561-8471, Leon Martin https://orcid.org/0000-0002-6747-5524, Andreas Henrich https://orcid.org/0000-0002-5074-3254

Abstract. *Purpose:* A previous paper proposed the usage of SHACL to assess the FAIRness of software repositories. Following this call to action, this paper introduces and discusses the changes made to QuaRE, a SHACL-based tool for validating GitHub repositories against sets of quality criteria, to facilitate this task.
Methodology: An operationalization of the abstract FAIR best practices from previous work is devised to enable a FAIRness assessment based on concrete quality criteria. Afterwards, a SHACL shapes graph implementing these constraints is introduced, followed by a discussion of the efficient generation of suitable RDF representations for GitHub repositories. Improvements regarding the usability of QuaRE are examined, as well. An evaluation on the FAIRness of 223 GitHub repositories and on the runtime performance of the assessment is conducted.
Findings: On average, trending repositories comply with fewer FAIR best practices than repositories expected to be FAIR. However, the latter still exhibit deficiencies, for example, regarding the correct application of semantic versioning. The low average runtime of the FAIRness assessment of respectively 3.50 and 5.73 seconds per repository permits the integration of QuaRE in, e.g., CI/CD pipelines.
Value: The FAIR principles are often mentioned as a measure to tackle the reproducibility crisis, which continues to have a significant impact on science. To implement these principles in practice, it is crucial to provide tools that facilitate the automated assessment of the FAIRness of software repositories. The enhanced version of QuaRE introduced in this paper represents our proposal for this demand.

Keywords. FAIR software, GitHub repositories, SHACL

1. Introduction

In computer science and various other disciplines, large amounts of research software are produced [1]. Research software encompasses “source code files, algorithms, scripts, computational workflows and executable files that were created during the research process or for a research purpose” [2, p. 16]. To develop and publish research software, GitHub[2] and other Git[3]-based platforms are a common choice [1]. In 2016, a survey [3]

[1]Mail: `leon.martin@uni-bamberg.de`.
[2]`https://github.com` (visited 2024-03-28)
[3]`https://git-scm.com` (visited 2024-03-28)

found that 52 percent of the respondents saw a "significant crisis" [3, p. 452] and 38 percent a "slight crisis" [3, p. 452] regarding reproducibility in science. According to the Association for Computing Machinery, an experiment is reproducible if another team that uses the same setup observes the same or very similar results as stated by the original team[4]. A literature review on reproducibility studies [4] in computer science from 2023 yields results similar to the 2016 survey. In four of the nine considered studies, the reproduction was successful, in three only in parts, and in two it was unsuccessful. This indicates that reproducibility in research is still an issue today. To mitigate this issue, the DFG (German Research Foundation) and others recommend that researchers should publish results following the FAIR principles [5]. These principles require the data to be Findable, Accessible, Interoperable, and Reusable [1,6].

In [7], the authors present ten FAIR best practices and a pipeline approach leveraging RDF-star [8] to assess the *FAIRness* of repositories in GitHub organizations, i.e., shared accounts with multiple associated repositories. They also state that a future approach could use SHACL [9] to validate repositories against the FAIR principles. Following this call to action, we adapt and expand the already existing SHACL-based research prototype QuaRe[5], which supports the validation of GitHub repositories against predefined project types. In this context, a project type corresponds to a set of quality criteria describing desired requirements for a repository [10]. The present paper introduces a new project type to validate repositories against the FAIR best practices from [7]. Moreover, we improve the usability of the application and evaluate how FAIR popular repositories on GitHub are, also considering the runtime performance of the new QuaRe version.

The remainder of this paper is structured as follows: Section 2 gives an overview of theoretical foundations and related work. Based on this, Section 3 explains our concept for the FAIRness assessment, followed be a presentation of the implementation in Section 4. Subsequently, Section 5 discusses the implementation and the evaluation results. Finally, a summary and an outlook in Section 6 conclude the paper.

2. Foundations & Related Work

The FAIR principles have been designed as "domain-independent, high-level principles" [6, p. 4] for data developed by the community. They are not only designed for human data users but also for machines [6]. Since the word *FAIR* suggests a judgment, some authors state to "explicitly describe FAIR as a spectrum, and a continuum; that there is no such thing as 'unfair' being associated with the FAIR principles, except maybe the specific case of data that are not even findable" [11, p. 52]. To support research software developers, the Netherlands eScience Center and Data Archiving and Networked Services (DANS) launched a website[6] with five research software specific FAIR recommendations in 2019[7]. Mainly members of the Netherlands eScience Center also developed *howfairis*,

[4] https://www.acm.org/publications/policies/artifact-review-and-badging-current (visited 2024-03-28)

[5] The implementation and all resources required to reproduce the evaluation results presented within this paper are available in the repository at https://github.com/uniba-mi/quare, which is also indexed in the Software Heritage Project's archive (https://archive.softwareheritage.org; visited 2024-03-28).

[6] https://fair-software.eu (visited 2024-03-28)

[7] https://www.esciencecenter.nl/news/netherlands-escience-center-and-dans-launch-new-fair-software-website-2 (visited 2024-03-28)

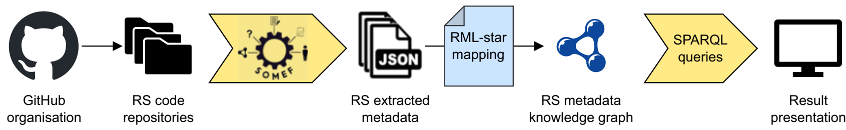
Table 1. Ten FAIR best practices for research software, adopted from [7, p. 2]. The original column labeled *Source* has been omitted.

ID	Best practice	FAIR Principle
BP1	A description (long or short) is available	F
BP2	A persistent identifier (e.g., DOI) is available	F
BP3	A download URL is available	A
BP4	A semantic versioning scheme is followed	A
BP5	Usage documentation (including I/O) is available	I,R
BP6	A license is declared	R
BP7	An explicit citation is provided	R
BP8	Software metadata (programming language, keywords, etc.) is available	F,R
BP9	Installation instructions are available	R
BP10	Software requirements are available	R

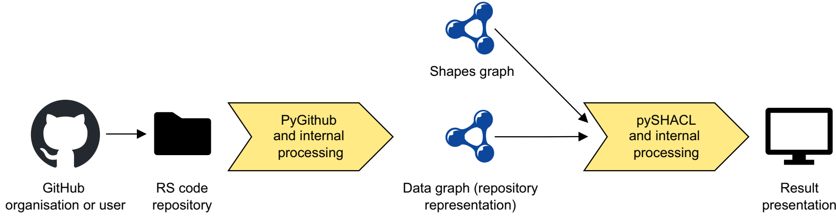
a "command line tool to analyze a GitHub or GitLab repository's compliance" [12] based on the five recommendations. The analysis result is listed in the terminal and provided as a badge that can be included in the README file of the respective repository. The GitHub organization[8] of the *howfairis* repository has several similar repositories, for example one[9] that checks the FAIR compliance as a GitHub Action[10]. There are also other publications regarding FAIR principles for research software. Motivated by the differences between pure data and research software, BARKER ET AL. [13] tailored the FAIR principles from WILKINSON ET AL. [6] to the research software domain in a community project. The result is a collection of guiding principles that specify the four foundational FAIR principles more concretely. Repositories can be checked against these principles with an online checklist[11]. After completing the checklist, a badge with the result can be included in the repository's README file. The tool was released by the Netherlands eScience Center and the Australian Research Data Commons[12]. Another possibility to assess the compliance of research software with the FAIR principles and an important basis for the present paper is the work by IGLESIAS-MOLINA and GARIJO [7,14]. They mention the *howfairis* tool and state that it is limited to only a subset of the FAIR principles. Additionally, they identify a general lack of tools for checking compliance with all FAIR principles on a GitHub organization level. As a consequence, they developed a pipeline approach for this task [7]. Their approach is based on the ten FAIR best practices shown in Table 1, which they have sourced from two other publications [15,16].

Before we summarize the implementation details of IGLESIAS-MOLINA and GARIJO [7,14] and present the QUARE tool, we briefly introduce relevant semantic web standards as context. The Resource Description Framework (RDF) [17] describes resources using triples where each triple comprises a subject, a predicate, and an object. Sets of such triples constitute RDF graphs. Subjects and objects correspond to nodes in these graphs, which can be Internationalized Resource Identifiers (IRIs), literals, or

[8] https://github.com/fair-software (visited 2024-03-28)
[9] https://github.com/fair-software/howfairis-github-action (visited 2024-03-28)
[10] https://docs.github.com/en/actions (visited 2024-03-28)
[11] https://fairsoftwarechecklist.net (visited 2024-03-28)
[12] https://www.esciencecenter.nl/news/new-self-assessment-tool-to-promote-fair-research-software (visited 2024-03-28)



(a) Pipeline-based validation process using RDF-star. The figure is adapted from [7].



(b) SHACL-based validation process with QUARE [10]

Figure 1. Simplified diagrams of the validation processes by IGLESIAS-MOLINA and GARIJO (1a) and by MARTIN and HENRICH (1b). RS is short for research software.

blank nodes. The predicates, which represent the edges, are IRIs that express properties, i.e., relations between subjects and objects. Since IRIs are a generalization of Uniform Resource Identifiers (URIs), absolute URIs also qualify as valid IRIs. Literals hold values of a specified datatype, for instance strings. RDF-star, an extension of RDF, additionally allows making statements about triples using other triples [8]. There is also a query language for RDF graphs, the SPARQL Protocol And RDF Query Language (SPARQL) [18]. The Shapes Constraint Language (SHACL) [9] allows the validation of an RDF graph (the *data graph*) against conditions formulated in another RDF graph (the *shapes graph*). The validation is done by a processor which outputs the results in a validation report [9]. The shapes in the shapes graph are either *node* or *property* shapes. Node shapes specify conditions on the level of nodes, whereas property shapes specify conditions about property or path values of a node. A node shape generally has one or more property shapes [19].

Figure 1 compares the validation process by IGLESIAS-MOLINA and GARIJO with the validation process underlying QUARE. The approach shown in Figure 1a leverages an existing application, the Software Metadata Extraction Framework (SOMEF) [20], to extract metadata from all repositories of a GitHub organization. Based on this data, an RDF-star representation of the repositories is created using a mapping language [7]. Afterwards, SPARQL queries are issued to test the compliance of the repository representation with the FAIR best practices. The validation results are then presented as plots. This approach does not use SHACL but its authors state that it opens “up the way towards designing validation mechanisms (e.g., SHACL shapes)” [7, p. 4]. QUARE, in comparison, is a single-page application that allows to check whether a given GitHub repository conforms with a selected project type. The user interface features two pages: On the specification page, available project types and associated quality criteria can be viewed. On the validation page, GitHub repositories of interest and desired project types can be selected. Afterwards, one can issue the validation. Originally, two approaches have been implemented for the specification and validation of quality criteria, one based on OWL [21], the other on SHACL. However, the study in [10] revealed that SHACL is better suited as it is significantly faster, the resulting report contains all and not only one of the viola-

tions, and the project types are easier to maintain. Consequently, the OWL approach has been deprecated and is not considered in the present paper. Figure 1b shows a simplified version of the validation process. Initially, QuaRe fetches the required repository data from the GitHub API and creates a repository representation in RDF, the data graph, based on this data. A SHACL processor validates the created data graph against a shapes graph that reflects the project types and their associated quality criteria. The resulting validation report and a basic verbalized version are then presented on the validation page.

3. Concept

To assess the FAIRness of repositories, their compliance with the best practices from Table 1 needs to be measurable. For this, an operationalization of the rather abstract best practices is necessary. However, Iglesias-Molina and Garijo provide details in this regard only for a selection of best practices in their paper [7]. Their repository documentation is more detailed in this respect but often still vague [14]. For instance, for BP10, they write in a README file: "Are there any requirements specified anywhere?" [14, l. 14]. This is not a problem for their approach since they delegate the operationalization to a metadata extraction tool [7,20]. In contrast, QuaRe is standalone. Thus, concrete quality criteria are required. To tackle this problem, the two original publications [15,16] have been consulted. The candidate quality criteria developed in the process have been iteratively refined through a manual inspection of relevant repositories regarding the alignment between our operationalization and the actual characteristics of these repositories. The following list presents the final quality criteria that constitute the new project type *FAIRSoftware* with respect to the original best practices:

BP1. **"A description (long or short) is available"**: The GitHub repository must have a README file or[13] a short description. This interpretation is in line with the one of Iglesias-Molina and Garijo [7].

BP2. **"A persistent identifier (e.g., DOI) is available"**: Such an identifier points "to the same version and location for long, specified amount of time" [16, p. 18], for example, for 20 years [16]. For this best practice, at least one of the following has to be fulfilled:

- There is at least one release, and for each release, the version tag matches the semantic versioning regular expression[14]. Note that tags such as *latest* contradict the idea of persistence.
- The homepage attribute on GitHub is filled with a DOI.
- The README file contains a DOI.

Compared to Iglesias-Molina and Garijo [14], we additionally include releases since they are expected to be available for a long time and identify particular repository versions, as well. This rule could be further extended based on Software Heritage Identifiers[15][15,16]. Note that some DOIs may also be false positives in cases where they are pointing to papers instead of the associated code.

[13] In this list, the term *or* means that at least one option must be fulfilled (not exclusive).
[14] `https://semver.org` (visited 2024-03-28)
[15] `https://www.softwareheritage.org` (visited 2024-03-28)

BP3. **"A download URL is available"**: IGLESIAS-MOLINA and GARIJO interpret this as a call for releases [14]. With a focus on GitHub, we interpret this rule differently, though. By default, a release contains the code from a particular point in time for download[16]. However, repositories can also be cloned or downloaded without explicit releases. The only prerequisite is that the repository can be accessed. Therefore, we require repositories to be public to conform with this best practice.

BP4. **"A semantic versioning scheme is followed"**: This criterion cannot be validated without a release. Therefore, at least one release is required, and for each release, the version tag must match the semantic versioning regular expression (as in BP2). Additionally, the increment between two consecutive version numbers must be valid. Semantic versions have three positions, called *major*, *minor*, and *patch* with an optional suffix[14]. Hence, a valid increment is as follows: If *major* is increased, *minor* and *patch* have to be set to zero. If *minor* is increased, *patch* has to be set to zero. If *patch* is increased, *major* and *minor* have to be unchanged. Finally, the new version must have a (different) suffix if all three positions are unchanged. These criteria are stricter than those from IGLESIAS-MOLINA and GARIJO who check compliance of the latest release with the semantic version naming scheme [14] but without an increment check. This way, a newer release could have a lower version number, which is not semantic.

BP5. **"Usage documentation (including I/O) is available"**: There must be at least one section in the README file where the lower-cased title contains *usage*, *how to use*, or *user manual*.

BP6. **"A license is declared"**: The repository must include a LICENSE file. This enables the GitHub API to return a license[17] for the repository.

BP7. **"An explicit citation is provided"**: IGLESIAS-MOLINA and GARIJO check whether there is a README file with citation information or a .cff file [14]. We adopt and detail the examples from the original publication [15] as they are more extensive. Thus, at least one of the following has to be fulfilled:

- A CITATION.cff file is present in the root directory of the default branch.
- Exactly one .bib file is present in the root directory of the default branch.
- A README file located in the root directory of the default branch contains at least one section where the lower-cased title contains *citation*, *cite*, or *citing*.

BP8. **"Software metadata (programming language, keywords, etc.) is available"**: For this best practice to be fulfilled, IGLESIAS-MOLINA and GARIJO require that the programming language, the creation date, a description (as in BP1), *and* at least one topic are specified explicitly and that one release is present [14]. Conversely, we only require at least one topic *or* the *about* attribute to be specified. An explicit mention of the programming language is not necessary since it is automatically determined by GitHub based on the files in the repository[18]. Similarly, the creation date can be estimated via the date of the first commit. BP4 already checks for releases such that this aspect can be ignored here.

[16] https://docs.github.com/en/repositories/releasing-projects-on-github/about-releases (visited 2024-03-28)

[17] https://docs.github.com/en/rest/licenses/licenses (visited 2024-03-28)

[18] https://docs.github.com/en/repositories/managing-your-repositorys-settings-and-features/customizing-your-repository/about-repository-languages (visited 2024-03-28)

BP9. **"Installation instructions are available"**: There must be at least one section in the README file where the lower-cased title contains *install*, *setup*, *set up*, or *setting up*.

BP10. **"Software requirements are available"**: There must be at least one section in the README file where the lower-cased title contains *dependencies*, *requirements*, or *prerequisite*. Alternatively, depending on the main programming language, precisely one file specifying the requirements must be present in the root directory of the default branch. Currently, this is limited to only four popular languages and a selection of admissible files:

- For JavaScript and TypeScript: a *package.json* file
- For Python: a *requirements.txt*, *environment.yaml* or *environment.yml* file
- For Java: a *pom.xml* or *build.gradle* file

Note that this approach does not work for repositories with software components located in separate folders as there is no single requirements file in the root directory. The QuaRe repository[5] is an example of this setup. In these cases and for languages other than the supported ones, the approach using the README file must be employed.

Based on this list, the SHACL shapes graph shown in Figure 2, which represents the *FAIRSoftware* project type, has been compiled. The naming of the shapes differs from that of the best practices for two reasons. First, other project types unrelated to FAIR data might also employ them in the future. Second, descriptive shape names reflect our interpretation of the best practices better. As depicted in the figure, the shapes graph has a modular structure and comprises property, node, and project type shapes. Project type shapes are nodes that combine all relevant node and property shapes for a project type. In standard SHACL terminology, these are regular node shapes, though.

3.1. Revision of the Repository Representation

To be able to validate software repositories against the FAIRSoftware shapes graph, data graphs that encompass their relevant characteristics have to be created, first. For this purpose, the Software Description Ontology is employed. It extends the ontologies schema.org and CodeMeta for organizing information about software and its metadata[19] [22]. Iglesias-Molina and Garijo use this ontology [7], as well, with D. Garijo being one of its authors [22]. Figure 3 visualizes our repository representation ontology. In cases where the Software Description Ontology lacks appropriate properties, custom properties in the *props* namespace are introduced.

As shown in [10], creating the repository representation by retrieving the necessary data via the GitHub API has taken around 90 percent of the total validation time. The reason for this is that QuaRe used to fetch all data required by the supported project types, even if the respective project type is not selected in the user interface. To mitigate this, the shapes graph now also provides information about which repository characteristics are necessary for the selected project type through the optional *sh:description* property attached to each project type node. Hence, the repository representations are now created as a composite of one or more of the following building blocks:

[19] https://w3id.org/okn/o/sd (visited 2024-03-28)

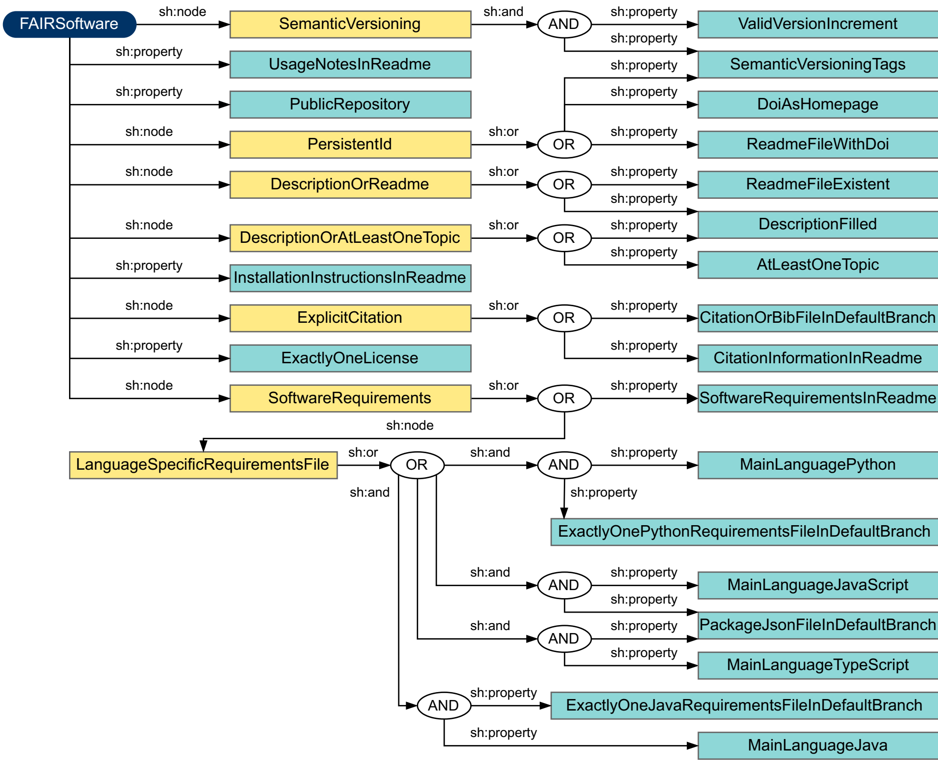


Figure 2. The fragment of the SHACL shapes graph representing the FAIRSoftware project type; other project types are omitted here. The project type shape for the FAIRSoftware project type is depicted in blue, the corresponding node and property shapes are light yellow and turquoise. *sh* refers to the SHACL namespace.

- Branches (optionally with files in the root directory of the default branch)
- Description (*about* attribute)
- Homepage
- Issues (not used in the *FAIRSoftware* project type, but potentially in others)
- License
- Main programming language
- README file (optionally with sections or the check for a DOI, or both)
- Releases (optionally with a check for the correct increment)
- Topics
- Visibility

An example value for the *sh:description* is: “The following repository properties are required to validate this project type: Visibility, Topics, Description, [. . .].” The sentence before the enumeration ensures readability and explains the enumeration to new developers. Once the validation process is triggered, the shapes graph is queried for the *sh:description* property of the selected project type, and the enumeration is extracted as a list. For each code word in the list, a function is called that retrieves the associated data and attaches it to the data graph.

Figure 3. An abstract visualization of the ontology underlying the repository representations (data graphs). IRIs are depicted in blue, literals in green. The cardinality is given in brackets. *sd* refers to the Software Description Ontology namespace, and *props* to the namespace for additional custom properties.

3.2. *Improving the Usability of* QuaRe

Apart from the introduction of the *FAIRSoftware* project type, QuaRe's usability has been improved for the present paper, as well. Previously, the verbalized explanation of the validation report on the validation page has been generated using violation-type specific templates. However, explanations generated this way were vague and included no information about the changes required to avoid the violations [10]. To enhance the verbalization, we first conducted tests with the large language models ChatGPT[20] and Google Bard (now Google Gemini[21]), focusing on ChatGPT because of more convincing results. When the prompt provided to ChatGPT included the raw validation report, the answer was not concrete enough and also contained hallucinations, though. For example, when the node shape *ExplicitCitation* was violated, ChatGPT incorrectly stated that this might be caused by missing information about software sources. When the shapes graph is added to the prompt, the responses improved but were still error-prone. For instance, ChatGPT regularly failed to mention that there are different ways to fulfill quality criteria. Moreover, the content was frequently paraphrased incorrectly: For example, it stated that it is sufficient that some .cff file is present to comply with BP7, which is, however, incorrect as explained above. Nevertheless, the results are promising but a follow-up paper

[20]https://chat.openai.com (visited 2024-03-28)
[21]https://gemini.google.com (visited 2024-03-28)

focusing on prompt engineering and user feedback is required before the verbalization of SHACL validation reports can be implemented using large language models.

As an interim solution, the *sh:message*, an optional shape property "for example to communicate additional textual details to humans" [9] is employed. When this property is added to shapes in the shapes graph and the shapes in question are violated, its value is automatically included in the validation report through the property *sh:resultMessage*. This way, hand-crafted yet informative static messages can be added to validation reports. Consequently, this option was leveraged to add information about the causes of a violation and recommended actions to all relevant shapes. For instance, the message "No citation information was found. Make sure they are included in the README file or there is a file CITATION.cff or a .bib file in the root directory of the default branch." was attached to the node shape *ExplicitCitation*. During validation, the backend extracts the custom messages from the raw validation report and displays them in a accessible list format. For a quick overview, an interface element revealing the share of fulfilled quality criteria was implemented, as well. Further details and screenshots follow in Section 4.

On the specification page, users can see available project types and their quality criteria. Previously, the quality criteria were presented in the SHACL notation. We introduce verbalization using *sh:description* and set similar values as for *sh:message*. However, here, the focus is on the target state and how it can be reached in detail. We also include the shape's name for developers and traceability. As with *sh:message*, we set *sh:description* at the node and property shape level. Although shapes can be referenced in different project types (see Figure 2), they only need to be described once. We use Markdown to provide formatting options such as lists because the employed operationalization of best practices often includes alternatives. Here, the formatting contributes to a quick understanding of the quality criteria. The backend fetches the project type nodes from the shapes graph and, for each of these nodes, looks up nodes connected via the properties *sh:node* and *sh:property*, i.e., node and property shapes. Their values for *sh:description* are extracted, and one list per project type is created and then displayed.

4. Implementation

Based on the insights from Section 3, we adapted and expanded QuaRe with its Svelte frontend and Python backend. The frontend interacts with an HTTP API provided by the backend to retrieve project types and submit repositories for validation. The response to the latter contains the validation results, including the verbalized explanation. Apart from the frontend, other tools could also access the API, for instance, a GitHub Action in a CI/CD scenario. QuaRe's backend uses the same libraries as before, with the exception that OWL-related libraries have been removed. The libraries include Flask[22] as the basis for the HTTP API, RDFLib[23] and pySHACL[24]. Moreover, we use PyGithub[25] as a wrapper for the GitHub API, and Beautiful Soup[26] to process sections of the README

[22]`https://github.com/pallets/flask` (visited 2024-03-28)
[23]`https://github.com/RDFLib/rdflib` (visited 2024-03-28)
[24]`https://github.com/RDFLib/pySHACL` (visited 2024-03-28)
[25]`https://github.com/PyGithub/PyGithub` (visited 2024-03-28)
[26]`https://www.crummy.com/software/BeautifulSoup` (visited 2024-03-28)

Figure 4. A screenshot of the validation page: In response to a click on the submit button, a repository has been validated against the new project type FAIRSoftware. Afterwards, the button labeled *View* was clicked, revealing the raw and verbalized explanations.

file, for example. The frontend now also depends on marked[27], which we use to convert the Markdown strings from the *sh:description* properties to HTML.

Figure 4 shows a screenshot of the revised validation page. It displays the results of validating a repository against the new project type *FAIRSoftware*. On the right, the additional interface element displaying the share of fulfilled criteria is located. As depicted, the GitHub repository *RDFLib/rdflib* complies with seven of the ten quality criteria of the FAIRSoftware project type as of 2024-03-28. The shown verbalized explanation is generated based on the values of the *sh:message* properties of the violated node and property shapes. In this case, the screenshot shows the messages for the violations of the node shapes *SemanticVersioning* and *SoftwareRequirements*.

Figure 5 shows a snippet of the specification page's current state, depicting information about the last two quality criteria of *FAIRSoftware*. The quality criteria are presented as an unordered list with tables as items. Due to the complexity of shapes like the node shape *SoftwareRequirements* from the screenshot, tables are used as they provide a better overview. The first row of each table is a one-sentence summary of the respective quality criterion. Further details can be looked up on demand in the second row. The last row provides the shape name for reference. Apart from tables and lists, we also use a hyperlink to the semantic versioning regular expression[28] in corresponding shapes (not in the

[27] https://github.com/markedjs/marked (visited 2024-03-28)
[28] https://semver.org/#is-there-a-suggested-regular-expression-regex-to-check-a-semver-string which links to https://regex101.com/r/Ly7O1x/3 (visited 2024-03-28)

Summary	**Sufficient software metadata has to be available.**
Details	At least one of the following has to be present: ◦ a description ◦ a minimum of one topic.
Shape name	node-shapes/DescriptionOrAtLeastOneTopic

Summary	**Information on the requirements of the software has to be present.**
Details	At least one of the following has to be fulfilled: ◦ The README file contains a corresponding section. The title of this section contains: ▪ "dependencies" or ▪ "requirements" or ▪ "prerequisite". ◦ The root directory of the default branch contains a requirements file common to the main programming language: ▪ for Java, exactly one of the following: ▪ build.gradle ▪ pom.xml ▪ for JavaScript and TypeScript: package.json ▪ for Python, exactly one of the following: ▪ requirements.txt ▪ environment.yaml ▪ environment.yml ▪ For other languages, the README approach has to be used currently.
Shape name	node-shapes/SoftwareRequirements

Figure 5. A screenshot of QuaRe's specification page showing the specification of two quality criteria.

figure) such that users can look up the original source, see valid and invalid examples, and test their tags.

5. Evaluation & Discussion

To evaluate the FAIRness assessment capabilities of the updated QuaRe, we ran the tool on six repositories expected to be compliant with the FAIR best practices and 217 trending GitHub repositories. The former include:

- The repository of Iglesias-Molina and Garijo [14].
- One repository[29] that lists *fair-research-software* as one of its topics.
- One repository[30] that lists *fair-software* as one of its topics.
- Three repositories[31,32,33] with a green *howfairis* badge.

[29] https://github.com/comses-education/wolf-sheep (visited 2024-03-28)
[30] https://github.com/zenodraft/zenodraft (visited 2024-03-28)
[31] https://github.com/fair-software/howfairis (visited 2024-03-28)
[32] https://github.com/GrainLearning/grainLearning (visited 2024-03-28)
[33] https://github.com/online-behaviour/machine-learning (visited 2024-03-28)

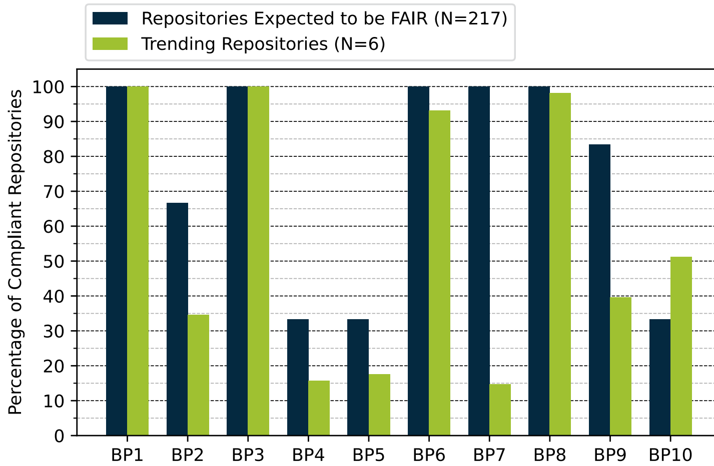


Figure 6. Compliance of the examined repositories with the FAIR best practices from Table 1

The 217 comprise the 20 most trending GitHub repositories as of the 15th of each month, from March 2023 to February 2024, without duplicates and without two repositories that are not available anymore. Notably, many of the examined repositories are closely related to artificial intelligence, which is reflected by the employed programming languages. Addressing the FAIRness first, Figure 6 shows the percentage of compliant repositories with respect to each best practice.

As expected, the results show that the repositories expected to be FAIR are generally at least as compliant with the FAIR best practices as the trending repositories with the exception of BP10. The reason for this is that three of the repositories expected to be FAIR happen to use ways of documenting the software requirements that are not yet covered by our operationalization: One uses a "Preparation"[29] section, another a setup.py file[33] and the last one mentions the software requirements in the installation section[32]. The biggest discrepancy between the two repository groups can be observed for BP7, indicating the maintainers of the examined trending repositories do not prioritize explicit citations. For BP1, BP3, BP6, and BP8, both groups perform very well. For BP3, this is no surprise since all examined repositories had to be public such that we can access them for the evaluation, which, at the same time, also fulfills this best practice. For BP4 and BP5 both groups perform poorly. In some cases, there are no releases, which directly results in a violation of BP4, but often the semantic versioning scheme is not applied correctly for all available releases. Largely due to the different formulations for usage documentation or its implicit inclusion in the installation instructions, BP5 is frequently violated. In many cases, this is not a flaw of the repositories but rather of our operationalization of the FAIR principles, though. The definition of FAIR principles generally leaves room for interpretation. However, tools like QuaRE need a concrete operationalization for the validation. Our operationalization is intended as a next step for the community efforts on the FAIRness assessment of software, as it provides one compiled list of concrete quality criteria for all best practices established in previous work. That being said, this operationalization is neither final nor complete. Its maturation, which, for example, includes the introduction of additional alternatives that occur in practice, strongly relies

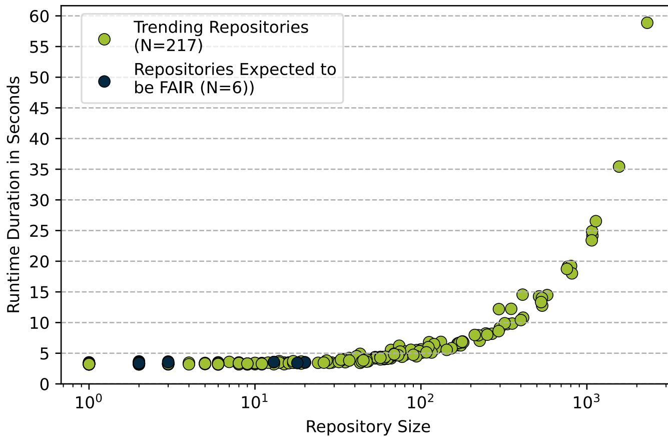


Figure 7. The runtime of the FAIRness assessment with respect to the size of the examined repositories. The size equals the sum of the branches and releases within a repository.

on discussion with the community. Potentially, there will always be room for error since it appears impossible to cover *all* alternatives to, for example, provide usage documentation in software repositories. At the same time, a good approximation should already be enough in practice. Currently, the validation page does not indicate what alternatives were found in the repository and with what confidence. Improving the explainability, a future version could provide additional information on provenance, confidence, and technique, like the approach of IGLESIAS-MOLINA and GARIJO does. The provenance of identified data includes, for example, whether something was identified in a separate file or in a section in the README file [7].

Compared to QUARE which uses ten best practices, *howfairis* (see Section 2) uses five recommendations for the *FAIRness* assessment. For the recommendations regarding registry and checklist, *howfairis* checks if a corresponding badge is present in the README file. QUARE does not need this preparation of badges. Furthermore, the interfaces differ as *howfairis* is a tool for the command line or a GitHub Action and QUARE a single-page application. Furthermore, *howfairis* prompts its results as text in the terminal and generates a badge [12], while QUARE provides a user-friendly visualization of the share of fulfilled criteria, a validation report and a verbalized explanation.

Addressing the runtime of QUARE's FAIRness assessment next, Figure 7 shows the duration of the validation process with respect to the size of the examined repositories. Here, the repository size refers to the sum of the releases and branches a repository comprises since these artifacts are the major factors determining the size of the data graph due to their cardinality (cf. Figure 3). As shown, the repositories expected to be FAIR are small in size, demonstrating that the size of repositories does not correlate with their FAIRness. The large majority of the trending repositories also has a size of less than 100. Unsurprisingly, for repositories of this dimension the FAIRness assessment takes the shortest time, i.e., well below ten seconds[34]. The results contain two outliers: The

[34]The runtime benchmarks have been executed on a PC with an AMD Ryzen 7 7700X processor and 32 GB of RAM. Clearly, internet connection and other factors also affect the measurements. Nevertheless, the benchmarks give a realistic impression of the relative runtime for the repositories.

assessment of one repository[35] with a size of 2311 took 58.87 seconds. Less drastically, the assessment of another repository[36] with a size of 1563 took 35.4 seconds. That being said, these repositories are not representative. In fact, the average duration of the FAIRness assessment is 3.50 seconds for the repositories expected to be FAIR and 5.73 seconds for the trending repositories. At first glance, these numbers appear weak considering their similarity to the results presented in [10], though. However, the new *FAIRSoftware* shapes graph is significantly more complex than the shapes graphs of the previously examined project types, which also leads to more interactions with the GitHub API. Hence, the results are actually commendable as they indicate that the time saved through the optimization of the repository representation generation and the time increase resulting from the more complex project type effectively cancel each other out. Overall, the results confirm the feasibility of efficiently assessing the FAIRness of GitHub repositories using SHACL. The duration of the validation process for typical repositories is low enough to integrate the tool in software development processes. This includes its addition to CI/CD pipelines using GitHub Actions, for example.

6. Conclusion

In this paper, we extended the SHACL-based research prototype QuaRe with a new project type to assess the *FAIRness* of GitHub repositories. The quality criteria are based on a work by Iglesias-Molina and Garijo [7], with a more detailed and partly different operationalization. Moreover, we presented the usability improvements applied to the tool. The evaluation gives an overview of how FAIR (trending) GitHub repositories are and how long the FAIRness assessment using QuaRe takes.

In addition to the leads on future work mentioned above, we want to point out the following ideas, as well. To mature the application of FAIR best practices, it might be worthwhile to introduce a prioritization to the quality criteria. Currently, all quality criteria of *FAIRSoftware* (and also of the other project types) are considered equally important in QuaRe. However, depending on the concrete repository, some FAIR best practices might be more relevant than others [7]. Using SHACL's *sh:severity* property, violations could be categorized on a quality criteria level into *sh:Info*, *sh:Warning* or *sh:Violation* [9]. This additional information could then be displayed on the validation page by ranking the violated quality requirements. Regarding QuaRe itself, the next step is to extend the capabilities of the specification page such that users can create new quality criteria and compose new project types directly in the user interface. As an alternative to GitHub, GitLab[37] and other platforms are employed to develop and distribute (research) software, as well. As long as appropriate APIs and, in the best case, Python wrappers[38] are available, QuaRe can be adapted to interact with other platforms.

[35] `https://github.com/ggerganov/llama.cpp` (visited 2024-07-03)
[36] `https://github.com/nextui-org/nextui` (visited 2024-07-03)
[37] `https://gitlab.com` (visited 2024-03-28)
[38] In the case of GitLab, there is `https://github.com/python-gitlab/python-gitlab` (visited 2024-03-28), for example.

References

[1] Hasselbring W, Carr L, Hettrick S, Packer H, Tiropanis T. From FAIR research data toward FAIR and open research software. it - Information Technology. 2020 Feb;62(1):39-47.

[2] Gruenpeter M, Katz DS, Lamprecht AL, Honeyman T, Garijo D, Struck A, et al.. Defining Research Software: a controversial discussion. Zenodo; 2021.

[3] Baker M. 1,500 scientists lift the lid on reproducibility. Nature. 2016 May;533(7604):452-4.

[4] Hummel T, Manner J. A Literature Review on Reproducibility Studies in Computer Science (short paper). In: Böhm S, Lübke D, editors. Proceedings of the 16th ZEUS Workshop, Ulm, Germany, February 29-March 1, 2024. vol. 3673 of CEUR Workshop Proceedings. CEUR-WS.org; 2024. p. 54-62. Available from: https://ceur-ws.org/Vol-3673/paper9.pdf.

[5] Deutsche Forschungsgemeinschaft. Guidelines for Safeguarding Good Research Practice. Code of Conduct. Zenodo; 2022.

[6] Wilkinson MD, Dumontier M, Aalbersberg IJ, Appleton G, Axton M, Baak A, et al. The FAIR Guiding Principles for scientific data management and stewardship. Scientific Data. 2016 Mar;3(1).

[7] Iglesias-Molina A, Garijo D. Towards Assessing FAIR Research Software Best Practices in an Organization Using RDF-star. In: Keshan N, Neumaier S, Gentile AL, Vahdati S, editors. Proceedings of the Posters and Demo Track of the 19th International Conference on Semantic Systems co-located with 19th International Conference on Semantic Systems (SEMANTiCS 2023), Leipzig, Germany, September 20 to 22, 2023. vol. 3526 of CEUR Workshop Proceedings. CEUR-WS.org; 2023. Available from: https://ceur-ws.org/Vol-3526/paper-09.pdf.

[8] Arndt D, Broekstra J, DuCharme B, Lassila O, Patel-Schneider PF, Prud'hommeaux E, et al.. RDF-star and SPARQL-star. Final Community Group Report; 2021. Visited 2024/03/28. Available from: https://w3c.github.io/rdf-star/cg-spec/.

[9] Knublauch H, Kontokostas D. Shapes Constraint Language (SHACL). W3C Recommendation; 2017. Visited 2024/03/28. Available from: https://www.w3.org/TR/shacl/.

[10] Martin L, Henrich A. Specification and Validation of Quality Criteria for Git Repositories using RDF and SHACL. In: Reuss P, Eisenstadt V, Schönborn JM, Schäfer J, editors. Proceedings of the LWDA 2022 Workshops: FGWM, FGKD, and FGDB, Hildesheim (Germany), Oktober 5-7th, 2022. vol. 3341 of CEUR Workshop Proceedings. CEUR-WS.org; 2022. p. 124-35. Available from: https://ceur-ws.org/Vol-3341/WM-LWDA_2022_CRC_1149.pdf.

[11] Mons B, Neylon C, Velterop J, Dumontier M, da Silva Santos LOB, Wilkinson MD. Cloudy, increasingly FAIR; revisiting the FAIR Data guiding principles for the European Open Science Cloud. Inf Serv Use. 2017;37(1):49-56.

[12] Spaaks JH, Verhoeven S, Tjong Kim Sang E, Diblen F, Martinez-Ortiz C, Etuk E, et al.. howfairis; 2022. Visited 2024/03/28. Available from: https://github.com/fair-software/howfairis.

[13] Barker M, Chue Hong NP, Katz DS, Lamprecht AL, Martinez-Ortiz C, Psomopoulos F, et al. Introducing the FAIR Principles for research software. Scientific Data. 2022 Oct;9(1).

[14] Iglesias-Molina A, Garijo D. oeg-upm/oeg-software-graph: v1.0.0. Zenodo; 2023.

[15] Gruenpeter M, Granger S, Monteil A, Chue Hong N, Breitmoser E, Antonioletti M, et al.. D4.4 - Guidelines for recommended metadata standard for research software within EOSC. Zenodo; 2023.

[16] Martinez PA, Erdmann C, Simons N, Otsuji R, Labou S, Johnson R, et al.. Top 10 FAIR Data & Software Things. Zenodo; 2019.

[17] Cyganiak R, Wood D, Lanthaler M. RDF 1.1 Concepts and Abstract Syntax. W3C Recommendation; 2014. Visited 2024/03/28. Available from: https://www.w3.org/TR/rdf11-concepts/.

[18] Harris S, Seaborne A. SPARQL 1.1 Query Language. W3C Recommendation; 2013. Visited 2024/03/28. Available from: https://www.w3.org/TR/sparql11-query/.

[19] Gayo JEL, Prud'hommeaux E, Boneva I, Kontokostas D. Validating RDF Data. Springer International Publishing; 2018.

[20] Kelley A, Garijo D. A framework for creating knowledge graphs of scientific software metadata. Quant Sci Stud. 2021;2(4):1423-46. Available from: https://doi.org/10.1162/qss_a_00167.

[21] Hitzler P, Krötzsch M, Parsia B, Patel-Schneider PF, Rudolph S. OWL 2 Web Ontology Language Primer (Second Edition). W3C Recommendation; 2012. Visited 2024/03/28. Available from: https://www.w3.org/TR/owl-primer/.

[22] Garijo D, Ratnakar V, Gil Y, Khider D. The Software Description Ontology; 2020. Revision: 1.8.0, visited 2024/03/28. Available from: https://w3id.org/okn/o/sd/1.8.0.