Electronic Communications of the EASST
Volume 83 Year 2025

deRSE24 - Selected Contributions of the 4th Conference for Research Software Engineering in Germany

Edited by: Jan Bernoth, Florian Goth, Anna-Lena Lamprecht and Jan Linxweiler

Software FAIRness, Documentation and Development Practices in Potsdam Researchers' GitHub Repositories

Akshay Devkate, Anna-Lena Lamprecht

DOI: 10.14279/eceasst.v83.2595

Electronic Communications of the EASST (https://eceasst.org).
Published by **Berlin Universities Publishing**
(https://www.berlin-universities-publishing.de/)

Software FAIRness, Documentation and Development Practices in Potsdam Researchers' GitHub Repositories

Akshay Devkate[1], **Anna-Lena Lamprecht**[2]

[1] akshay.devkate@uni-potsdam.de
[2] anna-lena.lamprecht@uni-potsdam.de
Institute of Computer Science, University of Potsdam

Abstract: This study examines GitHub repositories of researchers affiliated with organizations in Potsdam, aiming to analyze various aspects of software FAIRness, documentation quality, and software development practices. Our methodology builds upon the SWORDS pipeline that was initially developed to examine the GitHub repositories of researchers affiliated with Utrecht University for FAIRness-related parameters. Our extended version of the pipeline also collects information about the documentation available (project description, installation instructions, usage guides) and development practices followed (explicit requirements, use of continuous integration, use of linters, automated testing, comments at start of code files). Our results indicate a diverse range of adherence to FAIR principles and software development practices among the repositories. While some repositories exhibit exemplary practices with thorough documentation and robust community participation, others lack basic elements crucial for software reusability and interoperability. These findings highlight the need for enhanced training and resources to support researchers in adopting best practices of research software development.

Keywords: FAIR4RS, Software Development Practices, Repository Analysis

1 Introduction

Over the past years, numerous "best" or recommended practices for developing research software have been proposed. These guidelines aim to enhance the quality, reproducibility, and sustainability of scientific software. Among the most notable are the FAIR Principles for Research Software [BCK+22, HKB+22, LGK+19], which emphasize Findability, Accessibility, Interoperability, and Reusability, and corresponding recommendations (https://fair-software.eu/). Additionally, several frameworks and guidelines, such as the Best Practices for Scientific Computing [SGM21, WAB+14], Good Enough Practices for Scientific Computing [WBC+17], the DLR's Software Development Guidelines [SMH18] and various "10 Simple Rules" papers on programming and software management (https://collections.plos.org/collection/ten-simple-rules/), offer pragmatic advice for researchers and software engineers. Interestingly, classic software engineering frameworks like the capability maturity model (CMM) [PCCW93] for helping developers select process-improvement strategies have so far not played a significant role in research software development, although some works have explored its use for research data management [QCK14] and research software projects [DBM+24].

Despite the plethora of guidelines, there remains a significant gap in understanding how well these practices are actually followed by research software developers. Research software developers are a large and heterogeneous group that includes both professional Research Software Engineers (RSEs) and the diverse types of researchers who code as part of their scientific work. A thorough understanding of adherence to recommended research software development practices is crucial for identifying where RSEs and researchers who code need additional training or support, and what organisations or communities can do to help them. Moreover, it helps to identify areas where further research is needed to develop new, suitable methods and tools for research software development.

In this paper, we present our approach to assessing the adoption of recommended practices and the level of software FAIRness, documentation, and development practices among Potsdam researchers' GitHub repositories. Our analysis of software repositories revealed that half do not specify dependency requirements, and none document the use of software quality checklists. Basic project information is usually provided, but installation and usage guides are often missing, and the adoption of continuous integration along with automated testing and linting rules in continuous integration is low, indicating these practices are not a priority for scientific software developers.

The paper is structured as follows: Section 2 surveys related work, including studies that aim to understand adherence to best practices. Section 3 describes the methods of data collection and analysis we applied. Section 4 presents our results, and Section 5 discusses threats to validity. Section 6 concludes the paper with a summary of our findings and directions for future work. The dataset and analysis code that was used for this study is provided in as supplemental data to this article.

2 Related Work

Empirical studies specifically targeting research software and research software engineering are still relatively scarce. Especially empirical analyses as in this paper, focusing on artifacts in research software repositories, are a quite recent development, while most existing work is based on surveying researchers and reviewing literature. Table 1 surveys some significant studies of the past years. These studies touch on a variety of topics, including scientific documentation, software engineering concepts, software quality, verification, testing, test-driven development, and the use of version control tools like Git by research software developers.

For example, in 2010 Nguyen-Hoan Luke et. al. [NFS10] surveyed 47 developers of scientific software, aiming to find out where the scientific software development can be improved aligning them with the previous studies. They found that adoption of Integrated Development Environments (IDEs) and version control tools among the surveyed developers has risen, and documentation seems to have become more widespread compared to earlier studies. However, they pointed out a persisting need for improvement in the areas of scientific software development regarding tool usage, documentation standards, testing protocols, and verification activities.

A survey study conducted by Hannay et al. [HMS+09] in 2009 aimed to gain insight into the practices of scientific software developers. Hannay et al. found significant differences in understanding of software engineering concepts and recommended that understanding of software

Study	Year	Method	Outcomes
How do scientists develop and use scientific software? [NFS10]	2009	Survey	Scientific software development relies on peer learning and self-study, testing is valued but not well understood.
A survey of scientific software development [HMS+09]	2010	Survey	Highlights the need for better tool adoption, documentation, testing, and verification.
How do scientists develop software? An external replication [PWD18]	2017	Survey	Encompasses R programming, scientific software developers are self taught, works alone, lack of collaboration, and are insufficiently rewarded.
Test-driven development in scientific software [NC17]	2015	Survey	Discusses the challenges and benefits of testing and refactoring, offering TDD advice.
Towards computational reproducibility: Researcher perspectives on the use and sharing of software [AB18]	2018	Survey	Some practices ensure reproducibility, but there is a lack of long-term software maintenance.
A survey of the state of the practice for research software in the United States [CWR+22]	2022	Survey	Focuses on software engineering practices, testing, coding standards, and documentation.
Software engineering practices for scientific software development: A systematic mapping study [AACC]	2021	Literature review	Scientific software developers prioritize implementation efficiency through code reuse, third-party libraries, and strong programming techniques.

Table 1: Literature on development of research/scientific software quality

engineering concepts may increase if the development teams are larger. However, in a replication study conducted by Pinto et al. [PWD18] in 2017, the original hypothesis was contradicted suggesting that there is no correlation between understanding of software engineering practices amongst research software developers if the teams are larger.

In "A Survey of the State of Practice for Research Software in the United States" [CWR+22] Carver et al. assess various aspects of software engineering practices, including testing, licensing, continuous integration best practices, architecture design, requirements, peer code reviews, and tools. This evaluation was conducted through a survey aimed at understanding the landscape of research software engineering. Additionally, the survey investigated the availability of training, funding, and career pathways in research software engineering to provide a comprehensive overview of the field. A variety of earlier surveys have offered insightful information about the development and use of research software, as briefly explained above. With the available surveys we can draw certain conclusions about software development techniques employed researchers.

Arvanitou et al. have conducted a systematic literature review of 39 papers [AACC], and identified which of the software engineering practices which scientific software developer tends to focus and found that practices such as code reuse, use of third-party libraries, and the application of "good" programming techniques are given high importance. On the other hand, there is a noticeable lack of empirical information concerning the possible trade-offs related to these software approaches, namely their unintentional effects on other quality criteria. Arvanitou et al. also note that further empirical research is needed to understand the kinds of quality methods that research developers apply.

3 Method

For our study we adapted the SWORDS[1] pipeline [BQSL22], originally developed for analyzing GitHub repositories of Utrecht University researchers. The aim of this SWORDS@UU instance was to find out more about the knowledge gaps and training needs of research software developers at Utrecht University, in the context of a strategic realignment of its central IT services. Also being concerned with RSE-related training of students and research staff in Potsdam, we were interested in corresponding insights for our university and other local research organizations. Accordingly, we decided to adapt the SWORDS pipeline to collect repositories of Potsdam-based researchers, and also extended it to collect further variables for analysis.

Figure 1 summarizes the process, with dashed borders indicating the changes that we made to the original SWORDS pipeline: In the first phase, GitHub profiles (users and organizations) are identified by employing various methods, including GitHub search via the ghapi library (https://ghapi.fast.ai), accessing data from Papers with Code (https://paperswithcode.com), a customized collection method of GitHub organization commits to get the users who contributed to one of the repositories of research organization, and additional manual collection to supplement the above. In the second phase, repositories associated with the collected profiles are gathered using the ghapi library. This step is repeated once with the additional conbtributores found in organization repositories. In the third phase, a diverse set of variables is collected from the retrieved repositories, including FAIRness scores, software documentation, and indicators for the

[1] Scan and revieW of Open Research Data and Software

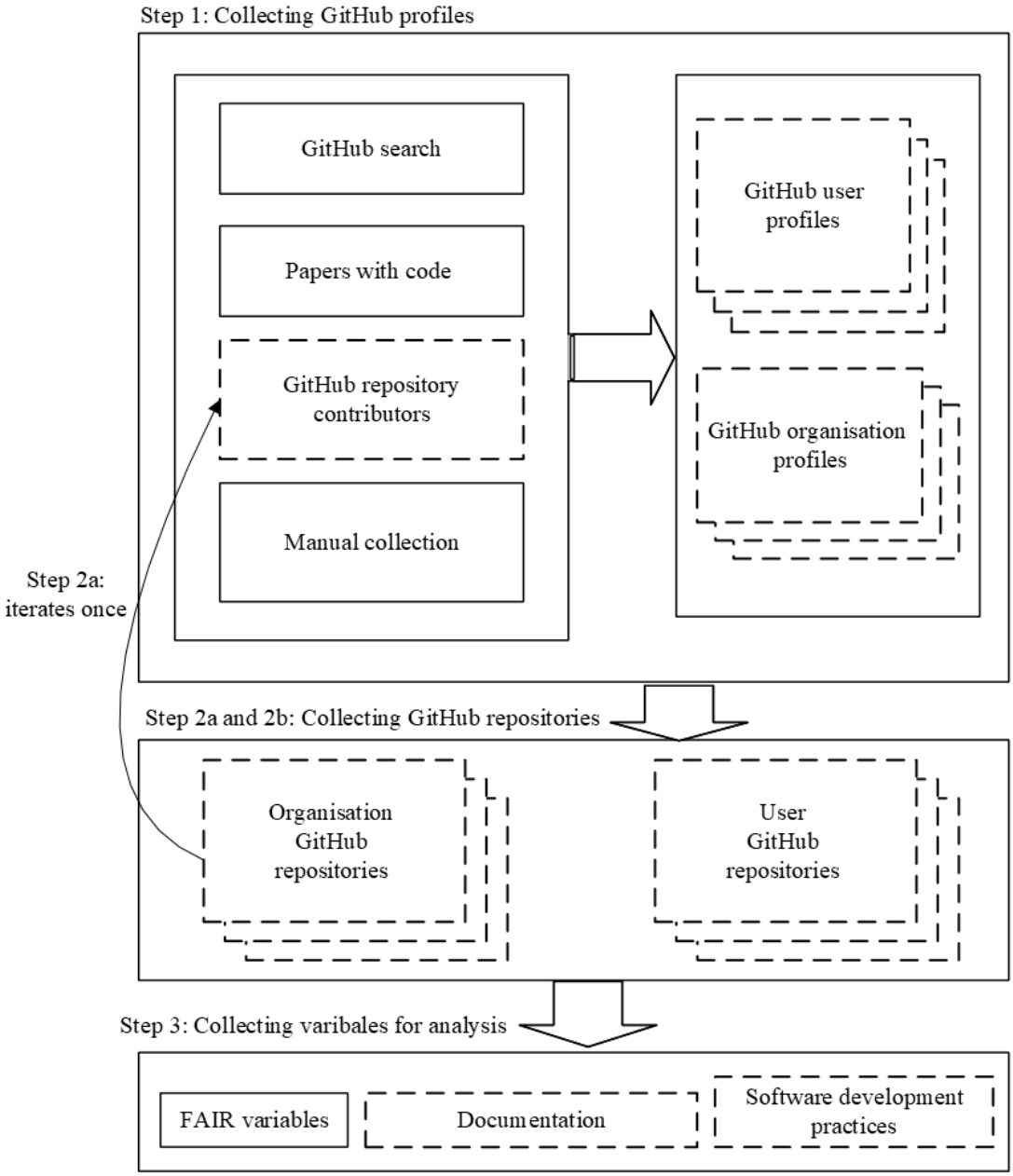


Figure 1: Methods for collecting GitHub profiles and repositories

software development practices followed. These variables are later analyzed and visualized to facilitate interpretation. We describe these three phases in more detail below.

3.1 Collecting GitHub profiles

The GitHub profiles we are interested in encompass user profiles of individuals (researchers who are employed to work on a research project, academic staff, PhD candidates, postdoctoral researchers, open-source contributors to research software, etc.) who have actively participated in repositories affiliated with research institutes/organizations in Potsdam, as well as GitHub organization profiles (research institutes, research organizations, smaller research projects and research groups, repositories that have a paper publication) belonging to research institutes/organizations in Potsdam. For finding GitHub profiles of researchers and research organizations, we employ multiple strategies that are available in the original SWORDS framework, and introduce some new strategies:

- **GitHub search:** The GitHub search method uses the ghapi library to systematically retrieve all matching GitHub profiles with the query provided in the search argument. To utilize this method effectively, we must establish search criteria that can include keywords such as research groups, institute names, and complement them with other relevant identifiers, such as the city name.

- **Papers with Code:** This approach gathers GitHub user profiles associated with repositories featured on https://paperswithcode.com. Papers with Code functions as a collaborative platform committed to offering unrestricted access to machine-learning papers, along with their corresponding GitHub open-source repositories. Using a topic search string input via a command-line argument, the method leverages the PapersWithCodeClient module to retrieve pertinent papers. Subsequently, it identifies the GitHub usernames linked to the owners of these repositories.

- **GitHub organization commits:** The method to retrieve GitHub user profiles that have contributed code or made at least one commit to repositories associated with GitHub organization profiles is iterated once. Initially, GitHub organization profiles are filtered, followed by parsing the repositories associated with these profiles. Although this process appears sequential, it includes a single feedback step in which the parsed repositories refine the initial profile filtering. This approach ensures that contributors who do not explicitly list their affiliations in their profiles or README files are still identified. By focusing on organization profiles, the method also addresses the limitations of GitHub search method, which may overlook contributors due to the absence of clear affiliation data.

- **Adding users manually:** Despite employing these methods for collecting GitHub user profiles, numerous GitHub accounts were overlooked by the search. This happened primarily because many profiles lack sufficient public information visible on their GitHub accounts, rendering them unidentifiable through the aforementioned collection methods. As a result, several accounts were manually added to the dataset by identifying known missing researchers who appeared on research institution and university websites, but were absent

from our dataset. We searched for these researchers using standard search engines with keywords related to their research and then merged their profiles with the other GitHub profiles obtained through our automated collection methods.

We identified several relevant GitHub organizations during this process, connected to the University of Potsdam, various local research institutes (Hasso Plattner Institute, Potsdam Institute for Climate Impact Research, GFZ Helmholtz Centre for Geosciences, Berlin-Brandenburgische Akademie der Wissenschaften, Leibniz Institute, Alfred Wegener Institute) and even some private companies and associations (Open System Pharmacology, Potsdamer Bürgerstiftung).

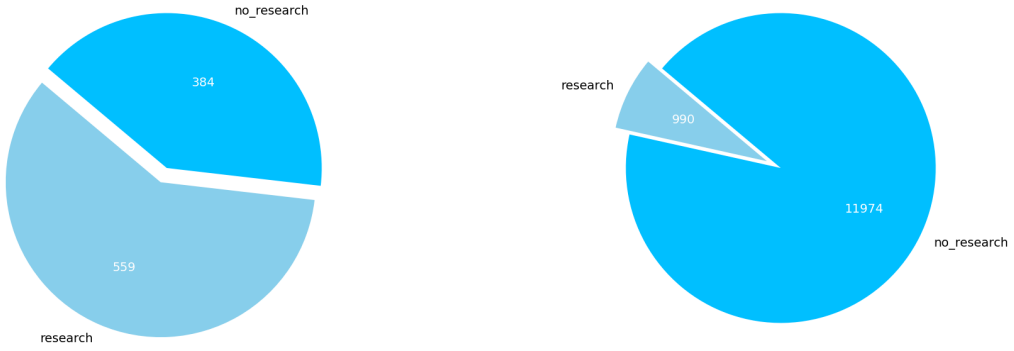
Finally, we consolidated all user profiles from the various sources into a unified dataset. This dataset was then meticulously examined to ensure accuracy and relevance, as we observed, for example, instances where profiles not directly related to Potsdam, Germany, were included due to keyword matches, such as researchers from Sunny Potsdam, USA. Following the consolidation, we enriched the GitHub profiles with additional data about users' affiliations. We annotated the profiles to denote the method of collection, eliminated duplicate entries, and manually filtered out irrelevant user profiles.

3.2 Collecting GitHub repositories

In the second step, the repositories associated with the GitHub profiles from the previous step are collected and filtered. We used ghapi to retrieve all the repositories of GitHub profiles, processing users and organizations separately. To better capture the iterative process involved, we logically divided this step into two sub-steps. In Step 2a, we collected repositories from organizations and then used this data to feed back into the GitHub Organization Commit method to collect additional GitHub user profiles. In Step 2b, after getting the additional user profiles, we then collected all repositories associated with those individual users. During this process, we filtered and removed duplicated (forked) repositories. We then classified the software projects into DLR application classes [SMH18], enabling a more detailed analysis of the research software by examining different metrics across all projects as well as within specific application classes.

3.2.1 Research Repositories

In the initial collection of GitHub repositories associated with Potsdam researchers and organizations, we filtered for those repositories that were used to analyze, interpret, and produce research paper results, repositories used to reproduce or replicate research papers with other datasets, or repositories that reference other research papers. While looking at repositories, there were some repositories which do not have specific information about their publication or how it is used for publication. However, they were funded by some research organization, so we have annotated them as research repository, too. The probability of finding research repositories was higher among the organization profiles than the among user profiles (see Figure 2). Still in our collection user research repositories (990) outnumbered organization research repositories (559). Given the limited number of repositories in both categories, which would not be sufficient for a meaningful comparative analysis, we decided to merge them into a single dataset of then 1548 repositories in total.



(a) organisation repositories

(b) user repositories

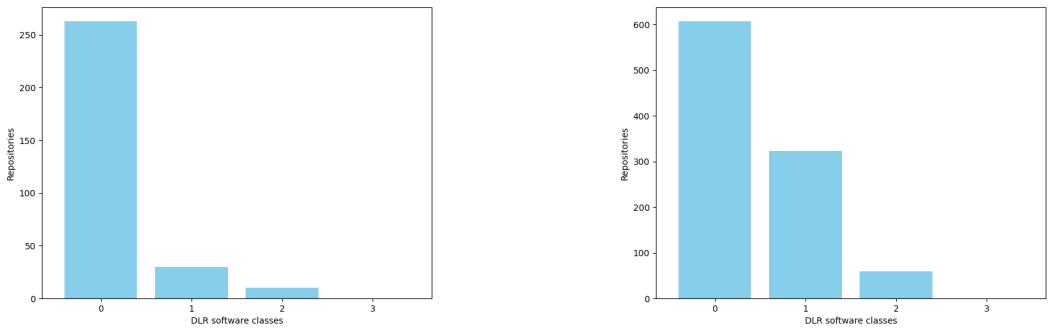
Figure 2: Research repositories found in the dataset

Application class	Description	Indicator
0	simple scripts and projects with no distribution	single contributor, no forks, no stars, no downloads
1	small projects, some distribution	1-3 contributors, some forks, some stars, some downloads
2	Projects with distribution	more than 4 contributors, more downloads, more stars, more forks with a community etc
3	Mission critical	Not included in the study

Table 2: DLR Application Classes

3.2.2 DLR Application Classes

The DLR Software Engineering Guidelines [SMH18] provide a framework for good software development practices at the German Aerospace Center (Deutsches Zentrum für Luft- und Raumfahrt, DLR). They define research software application classes ranging from 0-3 (see Table 2) as an indicator for the software quality assurance measures required for a research software project. The guidelines outline criteria for classifying research software based on its usage and distribution. To use the application classes in our research, we identified concrete metrics such as contributors, forks and stars count as suitable indicators (right column). Categorizing research software into application classes was essential for comparing software development across different levels of distribution and usage. In our datasets, DLR application class 0 (primarily consisting of small scripts or projects with no distribution) outnumbered the other DLR application classes 1 and 2 (see Figure 3).



(a) organisation repositories

(b) user repositories

Figure 3: Application classes of organization and user repositories.

3.3 Collecting Variables for Analysis

The software quality variables we collected in this study were carefully chosen through a comprehensive analysis of survey studies [PWD18, NC17, AB18, CWR+22] in line with recommendations outlined in ”Best” [SGM21, WAB+14] and ”Good enough” [WBC+17] practices for scientific computing. We prioritized those practices that were most feasible to assess in an approach that involved gathering variables using semi-automated scripts for parsing, followed by manual assessment. Given the extensive number of projects involved, we opted for quantitative evaluation over qualitative analysis.

3.3.1 FAIR Score

The Python package howfairis [SVT+22] can be used to evaluate a repository's adherence of research software to the Five Recommendations for FAIR Software https://fair-software.eu). Notably, these recommendations already cover several of the ”Best” and ”Good enough” practices in scientific computing. Concretely, they recommend to:

1. Use a publicly accessible repository with version control
2. Include a license
3. Register the software in a community registry
4. Enable software citation
5. Use a software quality checklist

Depending on how many of these recommendations the tool finds to be followed, it returns a FAIR score between 0 and 5. Some repositories indicate their FAIR score with a corresponding badge on the landing page.

3.3.2 Software Documentation

Next, we searched for documentation that would make it easier for users to install and use the software, or to enable them to reuse it with their datasets:

- **Project descriptions:** Recognizing the diverse range of software types in the repository, we conducted a quantitative analysis to ensure the presence of project descriptions or introductions. This approach allowed us to efficiently assess the availability of repository's introductory information across all projects.
- **Installation instructions:** To provide a comprehensive overview, we focused on quantitatively evaluating the installation guides. We ensured that repositories included sufficient information on where to find installation instructions, whether within the README.md file or linked to other resources such as Wiki pages or external websites.
- **Usage guides:** Our quantitative assessment of usage guides included checking for the availability of help commands for command-line applications and ensuring that usage information was present in the README.md file or linked to other relevant resources. This way we could gauge the accessibility of usage instructions across all repositories.

3.3.3 Software Development Practices

Following the "Best" [SGM21, WAB+14] and "Good enough" [WBC+17] scientific software development practices suggested, we build scripts that automate the collection of the following variables for analysis.

- **Testing:** We simply checked for the existence of a folder named *test* or *tests* within the root directory of each repository.
- **Making dependencies requirements explicit:** The "good enough practices" [WBC+17] suggests to make dependency requirements explicit by adding a `requirements.txt` file to the root directory of the project, or by adding a "Getting started" section to the README.md file. However, the paper only describes how to make dependency requirements explicit for Python. We extended that to check if dependency requirements are made explicit for R by having `DESCRIPTION` in the root directory, and for C++ by checking presence of `CMakeList.txt` in the root directory[2].
- **Continuous integration :** Continuous integration is a crucial practice in software development, encompassing various processes aimed at maintaining code quality and ensuring its reliability. This includes the incorporation of linter rules (such as automated code review, feedback on violation, consistent code base adhering linting rules of programming language) within continuous integration to assess code quality with each commit, along with the automation of testing procedures. To determine if continuous integration has been implemented within a repository, we examined the presence of specific files and folders located in the root directory. For example, we checked the exis-

[2] As our collection has R, C++ being used most next to Python.

tence of the folder `./github/workflow` to ascertain the utilization of GitHub Actions, `.circleci/config.yml` for CircleCI, and `.travis.yml` to confirm the integration with Travis Continuous Integration.

- **Linters in workflows:** After confirming the integration of continuous integration, our next step involved inspecting whether further linter rules were specified within .yml or .yaml files. We scrutinized these files for the inclusion of linter names. While certain projects aggregate all supplementary rules into a singular .yaml or .yml file, others may distribute them across multiple files. Consequently, we carefully examined each file within the respective continuous integration tool directory to ascertain the presence of linters. For instance, in Python projects, we searched for linters like *pylint, pycodestyle* and *flake8*. Additionally, for R projects, we checked for *lintr* and *styler*, whereas for C++ projects, we looked for *cpplint*, *cppcheck*, and *clang-tidy*.

- **Automated testing:** We reviewed each file within the respective continuous integration tool directory to ensure the presence of automated testing rules in .yml or .yaml file. For example, in Python projects, we searched for testing frameworks such as *pytest, unittest* and *nose*. Similarly, for R projects, we looked for *testthat*, while for C++ projects, we sought out libraries like *Google test* and *catch*.

- **Comment at the start of program/script.:** The "good enough practices" [WBC+17] recommend having a brief comment at the start of a program that should include at least one example of how the program is used. Additionally, they recommend developers to indicate reasonable values for parameters. We only accessed the presence of comments at start of every script by identifying the keywords that starts single or multiline comments for Python, R, C++. For example:

```
# Single line comment for Python

''' Multiline
comment for Python
'''
```

We checked for the symbols `#`, `'''`, `"`, `\\` which start the comments in the program files. We then checked them across the repository in each code file and annotated *less* if comments are present in less then 25% of program files, *some* if they are present in between 25% - 50%, *more* if they are present in between 50% - 75% and *most* if they are present in more then 75% of program files.

4 Results

In this section, we present the findings of our analysis on the FAIRness, documentation quality, and software development practices of the GitHub repositories maintained by researchers affil-

iated with organizations in Potsdam. We detail the results of our assessments, highlighting key trends and areas for improvement.

4.1 FAIRness

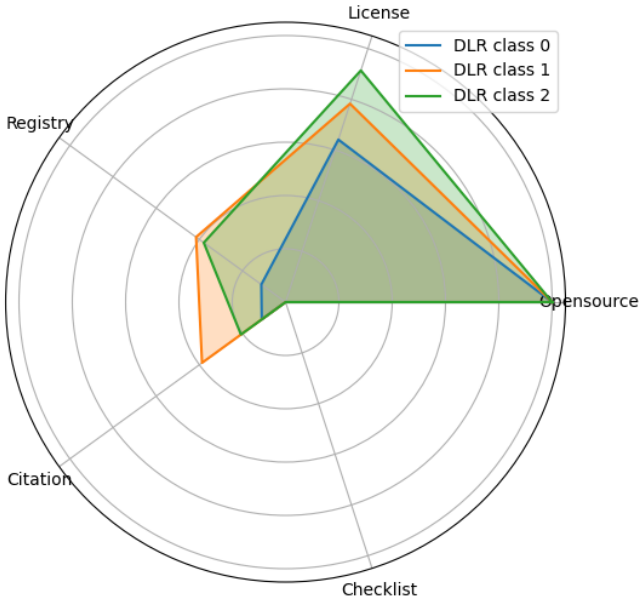


Figure 4: FAIRness

The spider diagram in Figure 4 summarizes the results of our FAIRness analysis of the repositories. For each of the five recommendations, it shows in different colors representing the application classes the percentage of repositories that follow these recommendations. Generally, we observe a trend that higher application classes follow more of the recommendations, which is in line with the expectation that software with greater distribution and usage adopt more best practices. More concretely, almost 90% of the application class 2 projects include a license, while this is the case for about 75% of the class 1 projects and for about 60% of the class 0 projects. For software citation, class 1 is the top scorer with about 40% of the projects enabling software citation, followed by roughly 20% of class 2 projects and about 10% of class 0 projects. The adoption of publishing software in a registry and including a corresponding badge on the README.md follows the same pattern. It should be noted here, however, that howfairis for this criterion only checks for the presence of a registration badge in the README.md, so it might both miss registrations that are not reflected by a badge in the README.md, as well as erroneously count pseudo-registrations that have a badge but are not known in the respective registry. Interestingly, none of the repositories in our dataset have a checklist badge to demonstrate adherence to the

Open Source Security Foundation (OpenSSF) best practices (https://www.bestpractices.dev/en). Trivially, all repositories in our dataset are from publicly accessible GitHub repositories and therefore score 100% on the open source criterion. Clearly, there is still room for improving the FAIRness score of research software projects across all application classes, although it is debatable, for example, whether publishing an application class 0 project to a registry is desirable.

4.2 Software Documentation

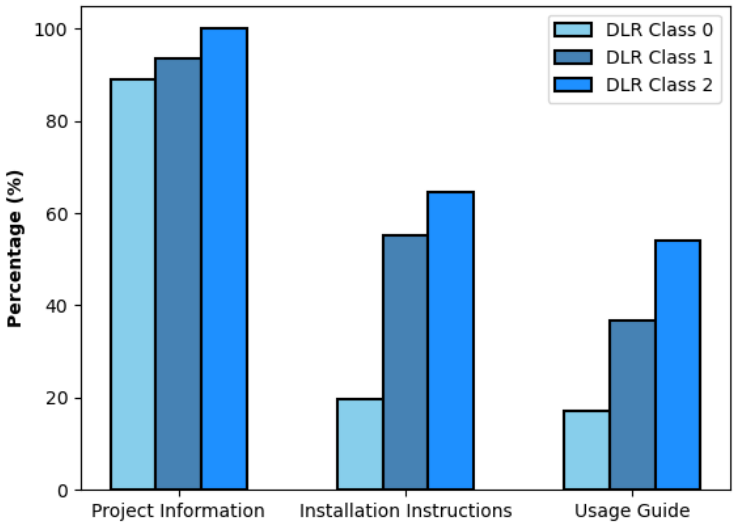


Figure 5: Software documentation

Figure 5 summarizes our findings regarding documentation in the research software projects examined. It shows that basic information about the project or repository was present in most of the projects across DLR application classes. However, installation instructions and usage guides were absent in more than half of the repositories, even for application class 2 projects, which have greater distribution. Nonetheless, there is a noticeable trend where the presence of installation and usage guides increases with higher distribution levels, progressing from DLR application class 0 to class 2.

4.3 Software Development Practices

In our evaluation of software development practices (see Section 3.3.3), we first assessed the presence of testing across all repositories, irrespective of programming language. Subsequently, we focused our analysis on specific practices for Python, R, and C++, as these were the most prevalent languages in our dataset (see Figure 6). As detailed earlier, these practices included checking for comments at the start of program scripts, making dependency requirements ex-

plicit, implementing continuous integration, and using additional linter rules within continuous integration and automated testing.

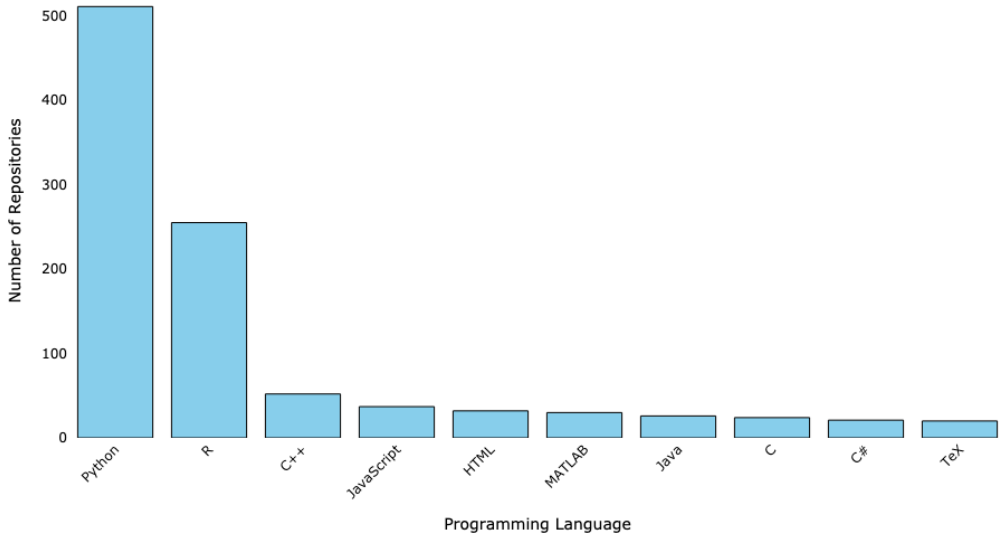


Figure 6: Most used programming languages in our dataset.

Testing. Figure 7 shows that the adoption of testing increases with higher application classes (0, 1, and 2). The highest frequency of testing is observed in application class 2; however, it is noteworthy that testing is still frequently neglected by developers within this class. It is arguable that testing may be less critical in DLR application classes 0 and 1, as many of these repositories consist merely of scripts for plotting or data cleaning. Moreover, over half of the repositories in DLR application class 2 have indeed implemented testing.

Comment at the start of program/script. As Figure 8 shows, the practice of including comments at the beginning of programming files is often overlooked by research software developers, irrespective of the software's classification into DLR application classes 0, 1, or 2. Only about 50% of the projects across application classes use such comments regularly.

Making dependency requirements explicit. We found that about half of the repositories, irrespective of their DLR application class, do not clearly specify their dependency requirements (see Figure 9).

Continuous integration, linters in workflows, automated testing. Finally, as Figure 10 shows, we found that the adoption of continuous integration is below 30% for DLR application class 2,

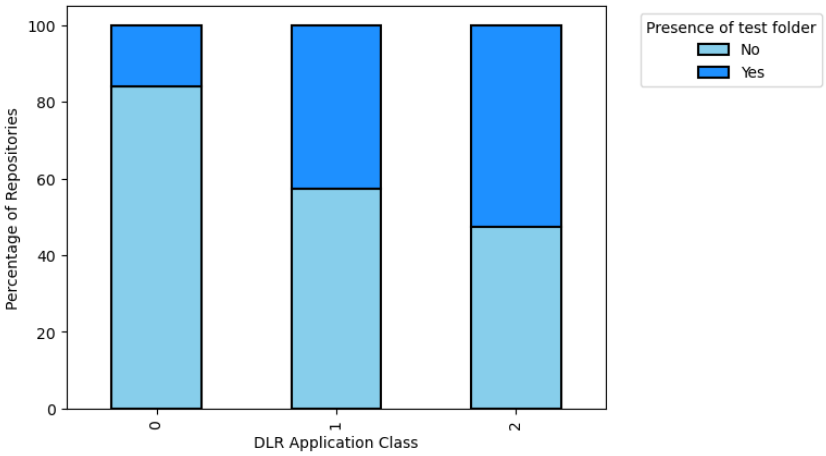


Figure 7: Testing (presence of test folder)

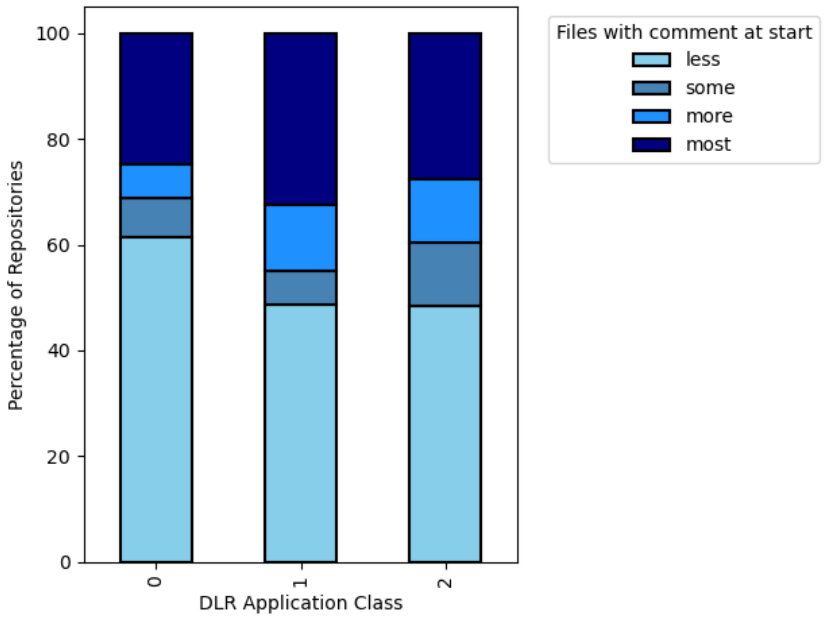


Figure 8: Comment at start

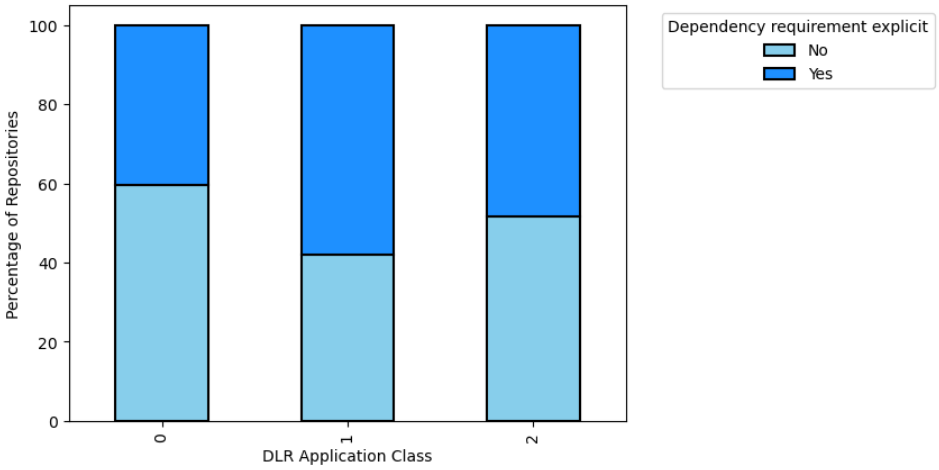


Figure 9: Dependencies explicit

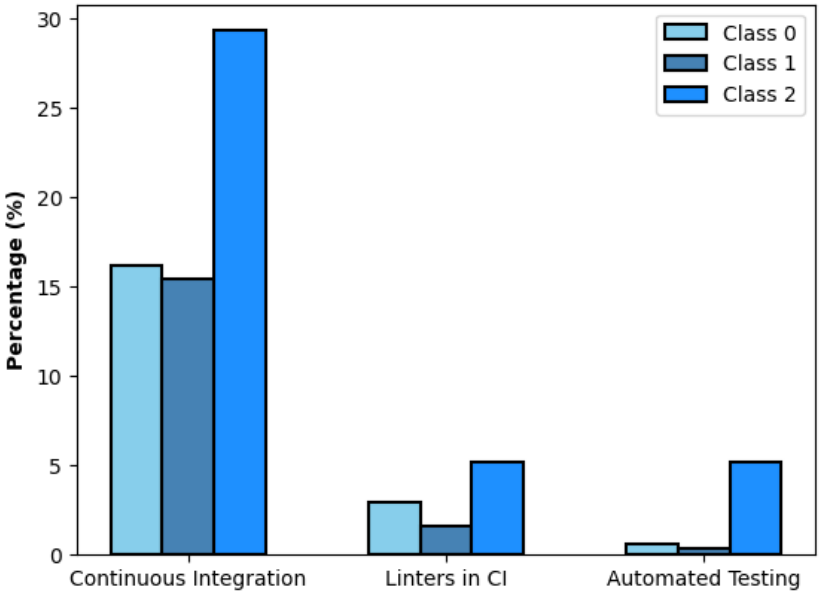


Figure 10: Continuous integration

and even lower, below 15%, for application classes 0 and 1. The adoption of linter rules in CI and automated testing is below 5%, indicating that these practices are not a priority for scientific software developers. Note that while most of the repositories with continuous integration have implemented it using *GitHub actions*, and small number repositories have implemented it using *TravisCI* and *CircleCI*.

5 Threats to Validity

Several threats to the validity of our study must be acknowledged. First, the repository collection is incomplete by design. Without a well-defined list of researchers affiliated with organizations in Potsdam, we relied on a comprehensive but inherently incomplete strategy to identify them. This has very likely led to an underrepresentation of certain researchers or projects. Generally, the identification of GitHub repositories of Potsdam-based researchers turned out to be more difficult than in Utrecht, where the personal web pages of university staff often point to their GitHub profiles, and where the university actively promotes the use of its own GitHub organization (https://github.com/UtrechtUniversity/getting-started).

Second, our study is subject to selection bias, as we only included projects that are already publicly available on GitHub. This excludes any private or non-GitHub repositories, potentially skewing the results. Also, the method does not take into account contributions from researchers who commit to repositories that are not associated with their own organization. Thus, we are likely missing external repositories that Potsdam-based researched contributed to. Similarly, we did not filter the repositories by time of last activity. It could be argued, though, that repositories that have not been active for, e.g., five or more years, are not representative of the current state of the field and should be left out. We note that most repositories in our dataset have indeed shown recent activity, but nevertheless we plan to explore the effects of such filtering in future work.

Third, the automated assessment of variables, while thorough, is not perfectly accurate. For example, assessing testing through the presence of folders named 'test' or 'tests' appears not to be the most effective approach. Manual inspection revealed that some repositories had implemented testing but did not organize the test files in the root directory folders named 'test' or 'tests'. We are working now on assessing the presence of testing using a more diverse approach, for example also examining imported libraries. Also, the analysis for comments at the beginning of programming files was often inaccurate. While evaluating the presence of comments at the start of programs, we did not assess the content of these comments. This approach may have led to some false positives, as some Python files included file paths as comments at the start of the program and copyright headers.

To mitigate some of these effects, several ongoing and future initiatives are in place. One such initiative is the analysis of GitLab repositories from Potsdam University, currently being undertaken as part of a Bachelor's thesis. Additionally, an ongoing Master's thesis aims to develop an AI-based classification of research repositories and their application classes, which could significantly enhance the accuracy and comprehensiveness of our analyses. We also plan to conduct similar analyses on other repository collections to further validate and refine our findings. For example, the SciCat project has set off to provide a curated dataset of scientific software repositories [MMP+23], and will hopefully enable us to obtain more representative and

generalizable results.

Furthermore, on a larger and more representative collection of repositories, it would be interesting to assess how robust the analyses are against diversity within the projects such as, for example, the application domain or choice of the programming language. This would also help to understand how easy it is to apply/reuse the analysis framework to other project repositories.

6 Conclusion

This study examines the GitHub repositories of researchers affiliated with organizations in Potsdam, aiming to analyze various aspects of software FAIRness, documentation quality, and software development practices. Using a comprehensive repository dataset, we assessed the degree to which these researchers adhere to FAIR principles and best practices in software documentation and development. Our methodology builds upon the SWORDS pipeline, originally developed to evaluate the GitHub repositories of Utrecht University researchers for FAIRness-related parameters. Our extended pipeline also collected information on the documentation available (project description, installation instructions, usage guides) and development practices followed (explicit requirements, use of continuous integration, use of linters, automated testing, comments at the start of code files).

Our results indicate a diverse range of adherence to FAIR principles and software development practices among the repositories. While some repositories exhibit exemplary practices with thorough documentation and robust community participation, others lack basic elements crucial for software reusability and interoperability. These findings underscore the need for enhanced training and resources to support researchers in adopting best practices in software development.

Several open questions remain to be investigated in future work. For example, do the results differ for types of developers, such as PhD candidates versus professional RSEs? As there is not "the one" leading guideline as a reference for software research engineers, some of the projects could have decided to adhere to a certain guideline and therefore have obtained better results. If that is the case, is there possibly a (unified) guideline that could be recommended? Could quality metrics for research software projects be defined, like the quality indicators discussed at the Helmholtz Association [CDJ+24]? And, related to this, how could it be avoided that researchers start "gaming" the metrics that they are evaluated with, without really improving the overall quality of the software?

Bibliography

[AACC] E.-M. Arvanitou, A. Ampatzoglou, A. Chatzigeorgiou, J. C. Carver. Software engineering practices for scientific software development: A systematic mapping study. 172:110848.
doi:10.1016/j.jss.2020.110848
https://www.sciencedirect.com/science/article/pii/S0164121220302387

[AB18] Y. AlNoamany, J. A. Borghi. Towards computational reproducibility: researcher perspectives on the use and sharing of software. *PeerJ Computer Science* 4:e163,

Sept. 2018.
doi:10.7717/peerj-cs.163
https://doi.org/10.7717/peerj-cs.163

[BCK+22] M. Barker, N. P. Chue Hong, D. S. Katz, A.-L. Lamprecht, C. Martinez-Ortiz, F. Psomopoulos, J. Harrow, L. J. Castro, M. Gruenpeter, P. A. Martinez, T. Honeyman. Introducing the FAIR Principles for research software. *Scientific Data* 9(1):622, Oct. 2022.
doi:10.1038/s41597-022-01710-x

[BQSL22] J. de Bruin, K. Quach, C. Slewe, A.-L. Lamprecht. Template of Scan and revieW of Open Research Data and Software. 2 2022.
https://github.com/UtrechtUniversity/SWORDS-template

[CDJ+24] W. zu Castell, D. Dransch, G. Juckeland, M. Meistring, B. Fritzsch, R. Gey, B. Höpfner, M. Köhler, C. Meeßen, H. Mehrtens, F. Mühlbauer, S. Schindler, T. Schnicke, R. Bertelmann. Towards a Quality Indicator for Research Data publications and Research Software publications – A vision from the Helmholtz Association. 2024.
https://arxiv.org/abs/2401.08804

[CWR+22] J. Carver, N. Weber, K. Ram, S. Gesing, D. Katz. A survey of the state of the practice for research software in the United States. *PeerJ Computer Science* 8, 2022. Publisher Copyright: © Copyright 2022 Carver et al.
doi:10.7717/peerj-cs.963

[DBM+24] Deekshitha, R. Bakhshi, J. Maassen, C. M. Ortiz, R. van Nieuwpoort, S. Jansen. RSMM: A Framework to Assess Maturity of Research Software Project. 2024.
https://arxiv.org/abs/2406.01788

[HKB+22] N. P. C. Hong, D. S. Katz, M. Barker, A.-L. Lamprecht, C. Martinez, F. E. Psomopoulos, J. Harrow, L. J. Castro, M. Gruenpeter, P. A. Martinez, T. Honeyman, A. Struck, A. Lee, A. Loewe, B. v. Werkhoven, D. Garijo, E. Plomp, F. Genova, H. Shanahan, M. Hellström, M. Sandström, M. Sinha, M. Kuzak, P. Herterich, S. Islam, S.-A. Sansone, T. Pollard, U. D. Atmojo, A. Williams, A. Czerniak, A. Niehues, A. C. Fouilloux, B. Desinghu, C. Goble, C. Richard, C. Gray, C. Erdmann, D. Nüst, D. Tartarini, E. Ranguelova, H. Anzt, I. Todorov, J. McNally, J. Burnett, J. Garrido-Sánchez, K. Belhajjame, L. Sesink, L. Hwang, M. R. Tovani-Palone, M. D. Wilkinson, M. Servillat, M. Liffers, M. Fox, N. Miljković, N. Lynch, P. M. Lavanchy, S. Gesing, S. Stevens, S. M. Cuesta, S. Peroni, S. Soiland-Reyes, T. Bakker, T. Rabemanantsoa, V. Sochat, Y. Yehudi, F. Wg. FAIR Principles for Research Software (FAIR4RS Principles). Mar. 2022.
doi:10.15497/RDA00065
https://www.research.manchester.ac.uk/portal/en/publications/fair-principles-for-research-software-fair4rs-principles(751dfce3-56e5-441f-8e3f-8c1401e0a1e0).html

[HMS+09] J. E. Hannay, C. MacLeod, J. Singer, H. P. Langtangen, D. Pfahl, G. Wilson. How do scientists develop and use scientific software? In *2009 ICSE Workshop on Software Engineering for Computational Science and Engineering*. Pp. 1–8. 2009.
doi:10.1109/SECSE.2009.5069155

[LGK+19] A.-L. Lamprecht, L. Garcia, M. Kuzak, C. Martinez, R. Arcila, E. Martin Del Pico, V. Dominguez Del Angel, S. van de Sandt, J. Ison, P. A. Martinez, P. McQuilton, A. Valencia, J. Harrow, F. Psomopoulos, J. L. Gelpi, N. Chue Hong, C. Goble, S. Capella-Gutierrez. Towards FAIR principles for research software. *Data Science* Preprint(Preprint):1–23, Nov. 2019.
doi:10.3233/DS-190026
https://content.iospress.com/articles/data-science/ds190026

[MMP+23] A. Malviya-Thakur, R. Milewicz, L. Paganini, A. S. I. Mahmoud, A. Mockus. SciCat: A Curated Dataset of Scientific Software Repositories. 2023.
https://arxiv.org/abs/2312.06382

[NC17] A. Nanthaamornphong, J. C. Carver. Test-Driven Development in scientific software: a survey. *Software Quality Journal* 25(2):343–372, 2017.
doi:10.1007/s11219-015-9292-4
https://doi.org/10.1007/s11219-015-9292-4

[NFS10] L. Nguyen-Hoan, S. Flint, R. Sankaranarayana. A survey of scientific software development. In *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*. ESEM '10. Association for Computing Machinery, New York, NY, USA, 2010.
doi:10.1145/1852786.1852802
https://doi.org/10.1145/1852786.1852802

[PCCW93] M. Paulk, B. Curtis, M. Chrissis, C. Weber. Capability maturity model, version 1.1. *IEEE Software* 10(4):18–27, July 1993.
doi:10.1109/52.219617
https://ieeexplore.ieee.org/document/219617

[PWD18] G. Pinto, I. Wiese, L. F. Dias. How do scientists develop scientific software? An external replication. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. Pp. 582–591. 2018.
doi:10.1109/SANER.2018.8330263

[QCK14] J. Qin, K. Crowston, A. Kirkland. A Capability Maturity Model for Research Data Management. *School of Information Studies - Faculty Scholarship*, Jan. 2014.
https://surface.syr.edu/istpub/184

[SGM21] R. Sanchez, B. A. Griffin, D. McCaffrey. Best Practices in Scientific Computing. *arXiv:2101.11857 [stat]*, Jan. 2021. arXiv: 2101.11857.
http://arxiv.org/abs/2101.11857

[SMH18] T. Schlauch, M. Meinel, C. Haupt. DLR Software Engineering Guidelines. Aug. 2018.
doi:10.5281/zenodo.1344612
https://doi.org/10.5281/zenodo.1344612

[SVT+22] J. H. Spaaks, S. Verhoeven, E. Tjong Kim Sang, F. Diblen, C. Martinez-Ortiz, E. Etuk, M. Kuzak, B. van Werkhoven, A. Soares Siqueira, S. Saladi, A. Holding. howfairis. Sept. 2022. original-date: 2020-09-04T13:42:15Z.
https://github.com/fair-software/howfairis

[WAB+14] G. Wilson, D. A. Aruliah, C. T. Brown, N. P. C. Hong, M. Davis, R. T. Guy, S. H. D. Haddock, K. D. Huff, I. M. Mitchell, M. D. Plumbley, B. Waugh, E. P. White, P. Wilson. Best Practices for Scientific Computing. *PLOS Biology* 12(1):e1001745, July 2014. Publisher: Public Library of Science.
doi:10.1371/journal.pbio.1001745
https://journals.plos.org/plosbiology/article?id=10.1371/journal.pbio.1001745

[WBC+17] G. Wilson, J. Bryan, K. Cranston, J. Kitzes, L. Nederbragt, T. K. Teal. Good enough practices in scientific computing. *PLOS Computational Biology* 13(6):e1005510, June 2017. Publisher: Public Library of Science.
doi:10.1371/journal.pcbi.1005510
https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1005510