Electronic Communications of the EASST
Volume 86 (1) Year 2026

deRSE26 - Selected Contributions of the 6th Conference for Research Software Engineering in Germany (Part 1)

Edited by Bernd Flemisch, René Caspart, Bernadette Fritzsch, Florian Goth, Jean-Noël Grad, Florian Thiery

Documenting multi-repository software ecosystems: The preCICE case

Gerasimos Chourdakis, Frédéric Simonis, Benjamin Uekermann

DOI: 10.14279/eceasst.v86.1.2812

License: This article is licensed under a CC-BY 4.0 License.

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

Documenting multi-repository software ecosystems: The preCICE case

Gerasimos Chourdakis [1], **Frédéric Simonis** [1], **Benjamin Uekermann** [1]

[1]Institute for Parallel and Distributed Systems, University of Stuttgart
gerasimos.chourdakis@ipvs.uni-stuttgart.de

Abstract: Documentation of software ecosystems is often scattered, and thus difficult to navigate and update. Moreover, separating the user documentation from the code repository often hides away the contribution and does not give appropriate credit in repository metrics. While documentation generation frameworks unify documentation and code, they pose certain limitations and primarily address a single repository. The preCICE coupling ecosystem includes components from multiple repositories with different technology stacks. This paper introduces the multi-repository solution implemented for the preCICE documentation, based on Git modules and a static site generator. This approach allows the documentation to be updated and reviewed together with the respective code contribution, ensuring up-to-date content and visibility of documentation contributions. This design also enables us to source content from third-party repositories or even the user forum, providing a comprehensive experience to the user. The website integrates with a search engine, provides analytics, and can be exported to a PDF file for archiving.

Keywords: Documentation, contributing, open-source, ecosystems

1 Introduction

Good documentation is essential to successful [BH13] and FAIR [BCK+22] research software, but it is difficult to coordinate in multi-repository ecosystems. Toolboxes following a bazaar structure (such as the DUNE toolbox for partial differential equations (PDE)[1], the computational framework Trilinos[2], or the astrophysics project Einstein Toolkit[3]) often maintain their documentation separately. Similarly, preCICE[4] –a coupling library and ecosystem for partitioned multi-physics and multi-scale simulations, with API in C++, C, Fortran, Python, Julia, MATLAB, Rust– is developed in multiple repositories (some extending third-party codes) and presented as one project [CDR+22]. The preCICE documentation has transitioned from scattered and disconnected resources to a modular documentation rendered in a central website.

This paper revisits common documentation approaches in Section 2, presents a modular documentation approach applicable to multi-repository software ecosystems in Section 3, demonstrates this approach for the preCICE documentation in Section 4, and discusses the current state and future work in Section 5 and Section 6.

[1] DUNE project website and documentation: https://www.dune-project.org/
[2] Trilinos website and documentation: https://trilinos.github.io/
[3] Einstein Toolkit website and documentation: https://www.einsteintoolkit.org/
[4] preCICE website and documentation: https://precice.org/ (web archive)

2 Common documentation approaches

Software projects provide documentation in different formats. In-code documentation (comments) might be rendered locally or online in Doxygen[5], Sphinx[6], or similar pages. Further developer and user documentation might be kept as stand-alone files and rendered in the same code documentation pages, or might be maintained in a separate website or wiki. In single-repository (monorepo) projects, documentation might be scattered for various reasons: separating the product presentation from the documentation, separating the user from the developer documentation, allowing users to make edits using more familiar interfaces, or for historical reasons. For example, the deal.II finite element library includes most of its documentation in Doxygen pages[7], with parts of it in a GitHub Wiki and a separate main website. Similarly, the SU2 fluid dynamics code[8] provides Doxygen pages (not available online) and a separate documentation website, while the OpenFOAM fluid dynamics code[9] scatters its documentation across user- and developer-oriented Doxygen pages, a GitLab wiki, a community-oriented MediaWiki, a website (legacy user and developer guides), and PDF versions of the latter. Other projects started with a GitHub/GitLab wiki and expanded to a minimal product website, as used to be the case with preCICE.

Multi-repository projects seem to fall into two categories: (a) projects that enforce the same technology stack over all components, and (b) projects that comprise uncoordinated ecosystem components. Two examples are the DUNE and Trilinos projects (Figure 1). DUNE (a multi-repository project with all subprojects built via a common build system) falls under the first category and offers most of their documentation via linked Doxygen and Sphinx pages (both used by different sections and DUNE modules). The Trilinos website falls under the second category, and is mainly a portal linking to the documentation of the various components of its ecosystem, each with its own history before joining Trilinos. In this case, minimal requirements are enforced.

These examples are drawn from simulation software close to preCICE, while the patterns observed are common across the research software engineering community. Larger open-source projects supporting software engineers often provide extensive online documentation, sometimes maintained as separate websites to which everyone can contribute with a pull request. Examples include the GitLab[10] and Docker[11] documentation websites. In both cases, the documentation is hosted under a subdomain separate from the main product website, which might be of advantage when their maintenance falls under different responsibilities.

Maintaining documentation close to the code, and with the same tools as code, is important: it ensures versioning of the documentation together with the code it documents, it makes it easier to update code and documentation in the same patch (integrating it into the review process), and it provides equal visibility to code and documentation contributions. However, in-code documen-

[5] Doxygen: https://www.doxygen.nl/
[6] Sphinx: https://www.sphinx-doc.org/, see also Breathe (Doxygen-Sphinx bridge): https://breathe.readthedocs.io/
[7] deal.II documentation: https://dealii.org/current/doxygen/deal.II/
[8] SU2 website and documentation: https://su2code.github.io/
[9] OpenFOAM documentation: https://www.openfoam.com/documentation/overview
[10] GitLab website: https://about.gitlab.com/ – documentation: https://docs.gitlab.com/user/ (in the code repository)
[11] Docker website: https://www.docker.com/ – documentation: https://docs.docker.com/ (in a separate repository)

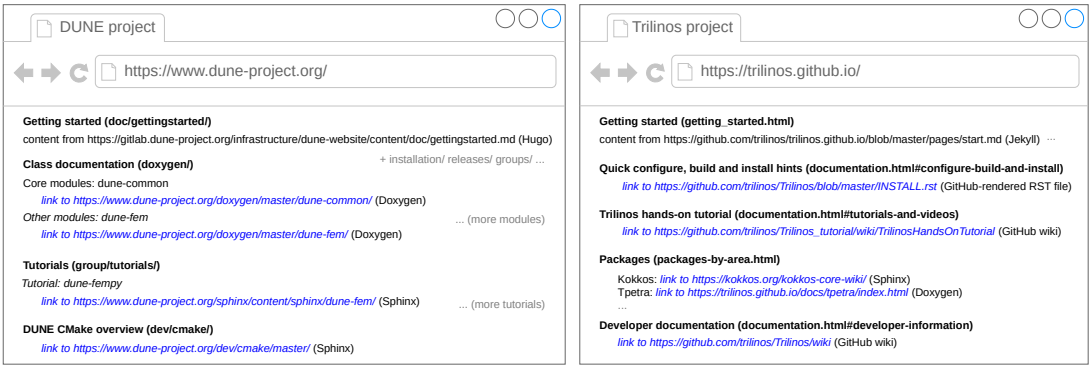

Figure 1: Schematic representation of the DUNE (left) and Trilinos (right) websites, both multi-repository projects. In the case of DUNE, where the modules are tightly integrated with each other, most of the documentation is maintained in the main Hugo website repository (internal), or in linked Doxygen/Sphinx pages in subdirectories. In the case of Trilinos, where the modules are loosely connected, the documentation is scattered across various resources, including a Jekyll website, GitHub wikis, Doxygen and Sphinx pages, and PDF files. In both cases, the website partially acts as a portal to other resources. Similarly scattered documentation can be found in many projects, and used to be the case for preCICE as well.

tation frameworks (Doxygen, Sphinx, and Antora[12] are common examples) typically impose a certain structure or make further restrictions in the underlying technology stack (e.g., programming language of the codebase, markup language, layout/style of the resulting web pages). With the notable exception of Antora, in-code documentation frameworks typically consider single-repository projects by default. Doxygen can link to external projects using tagfiles[13]. Sphinx provides the Intersphinx extension[14] and ReadTheDocs allows rendering subprojects[15] in the same Sphinx-based website (subdomain or subdirectory), with individual version menus.

Despite the many advantages of in-code documentation, several projects host a separate documentation repository. This solution is easy to implement and offered across the major forges (see, e.g., Codeberg Pages[16] and equivalent for GitHub and GitLab). Several static website generators are available and used for documentation, including the documentation-specific Sphinx, MkDocs[17], Docusaurus[18], or the general-purpose Jekyll[19] or Hugo[20] frameworks. This approach separates the web technologies stack from the project codebase and supports contributing and reviewing processes better fit for technical writers and external contributors.

[12] Antora: https://antora.org/
[13] Doxygen – linking to external documentation: https://www.doxygen.nl/manual/external.html
[14] Intersphinx: https://docs.readthedocs.com/platform/stable/guides/intersphinx.html
[15] ReadTheDocs subprojects: https://docs.readthedocs.com/platform/stable/subprojects.html
[16] Codeberg Pages: https://docs.codeberg.org/codeberg-pages/
[17] MkDocs: https://www.mkdocs.org/
[18] Docusaurus: https://docusaurus.io/
[19] Jekyll: https://jekyllrb.com/
[20] Hugo: https://gohugo.io/

3 Central rendering of distributed content

Combining documentation files maintained close to the code with a pre-processing step of collecting the content from different sources, it is possible to maintain the documentation as code while separating the content from its rendering infrastructure (Figure 2). We already apply a similar distributed idea for testing the various ecosystem components in the system regression tests of preCICE [CSU25], where the testing framework collects ecosystem components from different repositories. Having components available as Git repositories, we can declare them as Git submodules[21] in a website project maintained as a Git repository. Sources are specified in the `.gitmodules` file of the consuming website repository[22]. These can be updated automatically, in a pull or push fashion: for example, update the submodules every night[23], or notify the CI system of the project to update the submodules[24]. For the latter, a GitHub token needs to be shared with the repositories triggering the workflow, as an "Actions secret".

The respective static site generator needs to then integrate the content located in the submodules. A common static site generator bundled with GitHub Pages is Jekyll. This needs to be made aware of the additional files, and some files need to have unique names at deployment[25]. For this reason, we have developed a Jekyll plugin that collects files from the submodule paths[26], listed in the main Jekyll configuration file[27]. Each subproject (e.g., a `docs/` directory in a submodule) is expected to have Markdown files with Jekyll frontmatter, images with unique names under `images/`, and needs to be linked from the sidebar located in the website repository[28]. To apply this idea on any Jekyll documentation website, move the documentation files of each subproject to a `docs/` directory in the respective code repository[29], declare the individual repositories as Git submodules, and use our custom Jekyll `subprojects.rb` plugin. In the `_config.yaml`, declare each repository documentation path as a subproject. See Section 4 for more details and additional features. Other static site generators offer compatible concepts, such as Hugo mounts[30], which is natively supported by Hugo, without the need for Git modules or custom plugins.

Additional components (e.g., Doxygen pages) can be aggregated, built, and deployed to subdirectories or subdomains at build time. This still allows to have linked documentation websites under the same domain name, and, in some cases, integrate them into the same search index.

[21] Git submodules: https://git-scm.com/book/en/v2/Git-Tools-Submodules
[22] List of submodules: https://github.com/precice/precice.github.io/blob/v202609.0.0/.gitmodules
[23] CI: https://github.com/precice/precice.github.io/blob/v202609.0.0/.github/workflows/update-submodules.yml
[24] CI: https://github.com/precice/tutorials/blob/develop/.github/workflows/update-website.yml
[25] GitHub Pages deployment branch: https://github.com/precice/precice.github.io/tree/gh-pages
[26] Custom Jekyll plugin: https://github.com/precice/precice.github.io/blob/v202609.0.0/_plugins/subprojects.rb
[27] Main Jekyll configuration file: https://github.com/precice/precice.github.io/blob/v202609.0.0/_config.yml
[28] Sidebars: https://github.com/precice/precice.github.io/tree/v202609.0.0/_data/sidebars
[29] One can use git-filter-repo to retain the history: https://github.com/newren/git-filter-repo
[30] Hugo mounts: https://gohugo.io/configuration/module/#mounts

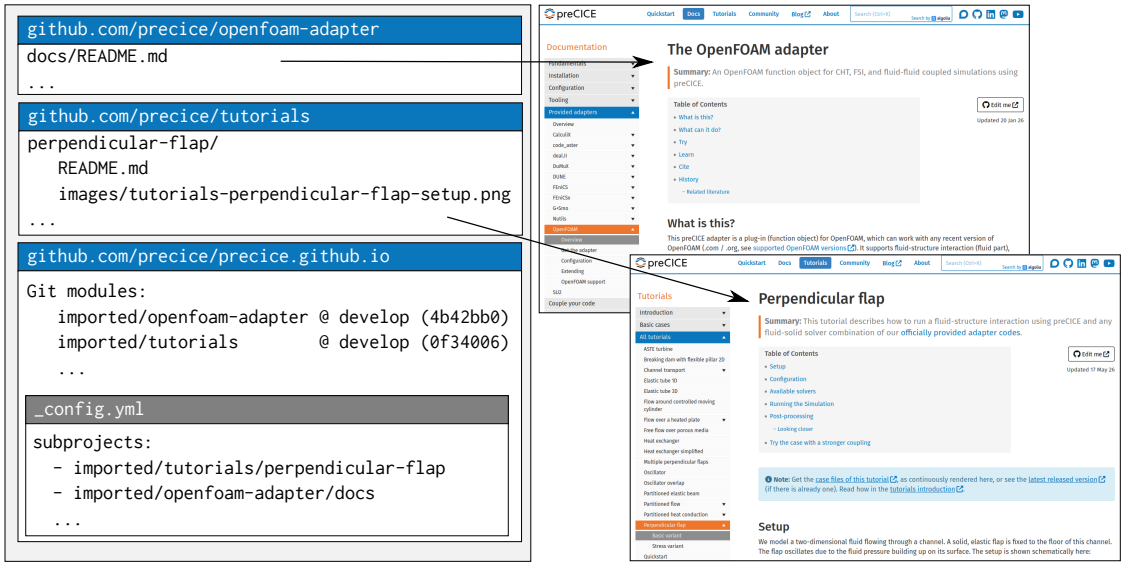

Figure 2: The content of the preCICE documentation is sourced from multiple code development repositories, defined as Git submodules, and integrated into the source tree of the Jekyll build with a plugin. Git modules are updated using CI workflows, documentation is expected in Markdown files with Jekyll frontmatter, and images need to have unique filenames. The user experiences a single website with a coherent structure and appearance. Doxygen (C++, C, Fortran API) and Sphinx (Python API) pages are also available in a subdomain, but only meant for navigating the API and the internals of the library.

4 The preCICE documentation

The preCICE ecosystem includes a core library, ready-to-use adapters for established simulation codes (typically modifying and extending these third-party codes), examples coupling such simulation codes, auxiliary tools, and further components [CDR+22]. All of these components are developed openly on GitHub[31]. Previously, an entry point website was pointing to the wiki of each GitHub project: a disconnected Git repository with a web-based editor. Some documentation was hosted mainly in these wikis, some other in the entry-point website, and some in Doxygen pages[32]. Updating the documentation for an unreleased feature required post-release updates to the documentation, which could not be discussed beforehand in a pull request.

The new preCICE documentation, available on precice.org, follows the approach described in Section 3. The same entry point website renders all the available documentation. Content of new components is being sourced from their code repositories, existing documentation is gradually being migrated close to the respective code it documents, and overarching content (such as a guide to the available literature[33]) is maintained in the website repository. The generated code documentation is rendered separately from the curated documentation using the in-code

[31] preCICE on GitHub: https://github.com/precice/
[32] Older preCICE docs: https://precice.org/fundamentals-previous-versions.html (web archive, 2020, 2026)
[33] Literature guide: https://precice.org/fundamentals-literature-guide.html (web archive)

documentation framework native to each component[34], only meant as a tool for code navigation.

The website offers a few features similar to specialized documentation systems, with meta-documentation pages[35] describing these in detail. While alternatives exist and continuously appear new for implementing various features, we refer here to those we currently use. All content is rendered with the same style, which is based on a widely-used theme[36]. The search engine Algolia indexes all content of the website, using its open-source plan[37]. A curated PDF export of the website is generated periodically, using the non-commercial license of Prince[38], and published together with the preCICE Distribution on a persistent repository[39]. Mathematical expressions are rendered with KaTeX[40]. Further visual features include annotation boxes[41] and Mermaid graphs[42]. The website invites everyone to contribute, with a prominent "Edit me" button on every page, which directs to the respective Markdown file on GitHub[43] and shows when the page was last modified, querying the corresponding Git repository using a custom Jekyll plugin[44]. Additional content is sourced from GitHub (such as unreleased changelog entries[45]) using the GitHub API on the client side[46] and from the community forum on Discourse[47] using the Discourse API on regular CI updates[48]. Finally, cookie-free analytics are publicly available, provided by Plausible[49].

The landing page presents the project as a product. A video-like example, based on Asciinema[50] presents the basic functionality and links to the Quickstart guide[51] for a concrete first example to run. This example makes assumptions about the user's system and provides concrete suggestions on how to install which components, reducing the options that might otherwise confuse the user[52] and postpone their attempt to try the software. The rest of the documentation includes installation guides, feature descriptions, configuration references, developer documentation, and the aforementioned meta-documentation. Installation guides are provided for different operating systems, package managers, and specific HPC systems, and the list of dependencies includes hints for their installation, which has previously attracted visitors that do not seem to visit any further pages. Next to the documentation, community-related information

[34] See the API pages: https://precice.org/couple-your-code-api.html (web archive)
[35] Documentation of the documentation: https://precice.org/docs-meta-overview.html (web archive)
[36] Google Jekyll Documentation Theme: https://github.com/tomjoht/documentation-theme-jekyll
[37] Algolia for open source: https://www.algolia.com/for-open-source
[38] Prince XML non-commercial license: https://www.princexml.com/purchase/license_faq/#non-commercial
[39] preCICE Distribution: https://precice.org/installation-distribution.html (web archive)
[40] KaTeX: https://katex.org/
[41] Alerts: https://precice.org/docs-meta-cheatsheet.html#alerts (web archive)
[42] Mermaid: https://mermaid.ai/ – example: https://precice.org/configuration-coupling-multi.html
[43] See the page layout file: https://github.com/precice/precice.github.io/blob/v202609.0.0/_layouts/page.html
[44] Plugin: https://github.com/precice/precice.github.io/blob/v202609.0.0/_plugins/last_modified_at.rb
[45] Roadmap: https://precice.org/fundamentals-roadmap.html#unreleased-changes-in-precice (web archive)
[46] GitHub REST API: https://docs.github.com/en/rest
[47] Frequently Asked Questions (FAQ): https://precice.org/fundamentals-faq.html (web archive)
[48] CI: https://github.com/precice/precice.github.io/blob/v202609.0.0/.github/workflows/update-discourse-data.yml
[49] Plausible analytics for preCICE: https://plausible.io/precice.org
[50] Asciinema: https://asciinema.org/
[51] Quickstart: https://precice.org/quickstart.html
[52] Reminder: At least two simulations are needed for preCICE to couple, typically using third-party codes. Given that users are often not familiar with all the involved codes, dependencies, or the computing platform itself, reducing the friction at the first encounter is essential to user success.

(such as events, contributors, and contributing guidelines) is rendered on the same website, under the "Community" section. The latter meets in a dedicated Discourse[53]-based forum: these structured conversations are indexed by search engines and contribute to the available documentation, with some content being rendered on the website (News, FAQ).

5 Discussion

The preCICE documentation includes all the components of the diátaxis framework [Pro25], and the content has evolved mainly out of need: when a user asks a question, maintainers often extend the documentation to be able to answer with a reference to that. Literature [HF22] highlights the preCICE documentation as a good example, next to DuMux[54]. Many components and the documentation itself follow many best practices suggested in literature [Lee18, Wri26]: the code includes comments, there are several examples, there is a quickstart guide and README files, all the documentation is version-controlled, the API is documented in detail, error messages provide suggestions, and an extensive literature guide explains when to read and cite what.

Overall, the distributed solution described in Section 3 has demonstrated automation for user documentation across a broader ecosystem of components. Documentation is being updated and discussed in the same GitHub pull requests as code patches, friction to update the documentation is reduced, a copy of a codebase is accompanied by the respective documentation, and contributions to the documentation are automatically listed on GitHub similarly to code contributions.

A shortcoming of the current solution is the lack of rendering for different versions of the ecosystem or the individual components: to keep the documentation up-to-date and avoid documentation patches to the main branch, the latest development branch of each component is being rendered. Annotations inform the user when a feature is expected in a future version or changed in some release. Only the latest version of every component is supported; as a workaround, published archives of the exported PDF and direct links to the Wayback Machine[55] provide access to older information. Versioning the ecosystem is itself a challenge with effects in the documentation and testing. Furthermore, specific assumptions are required across the rendered repositories (file structure, permalinks, Markdown frontmatter, documentation license), which need to be standardized and enforced, and some duplication of effort remains[56]. Finally, while these components are maintained primarily by close collaborators, legal safeguards need to be in place when applying this approach to a community-driven bazaar-style ecosystem: documentation licenses need to be compatible, and the legal responsibility for each documentation page should be clear. Clear project governance would increase confidence when applying such a solution.

In-code documentation frameworks such as Sphinx or Antora have made significant advances in supporting multi-repository projects: For projects that can enforce such a framework across all repositories, that might be a technically superior solution, even though the non-technical issues remain. For community-driven ecosystems, the presented solution is readily applicable to a wide range of existing documentation websites, without changing the underlying framework.

[53] Discourse: https://discourse.org/

[54] DuMux website: https://dumux.org/ – documentation: https://dumux.org/docs/, a DUNE extension module

[55] Wayback Machine – The Internet Archive: https://web.archive.org/

[56] The sidebar remains in the website repository, and the shared test suite [CSU25] is hosted in the tutorials repository.

6 Future work

Looking ahead, we are migrating more content to the respective code repositories, and we are integrating additional content from external sources, such as the community forum, and the API documentation in various programming languages. We are also in the late stages[57] of migrating the technical infrastructure from Jekyll to Hugo, a more actively developed static site generator currently used by, for example, the GitLab and Docker projects for their documentation. This allows us to scale to more pages, while reducing build times and introducing a sections-based structure friendlier to search engines. Looking at the larger community of Hugo, we aim to reduce our maintenance cost, and we have invested significant effort in porting the technical infrastructure[58], without any visual difference. Immediate next steps are sourcing the sidebar entries from the same source as the documentation files, integrating selected Doxygen pages in the same search index, and categorizing tutorials, adapters, and other content to allow filtering. In parallel to these technical improvements, and having accumulated extensive documentation, forum discussions, and research literature, we are researching ways to improve the integration of the documentation and the community forum with large language models. Moderated user studies could also help us identify further navigation issues and potential for such integrations.

Acknowledgements: The authors acknowledge funding from the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under project numbers 391150578 and 528693298, funding from Germany's Excellence Strategy – EXC 2075 under project number 390740016, and support from the Stuttgart Center for Simulation Science (SimTech). We would like to thank Christopher Lorenz for the initial technical implementation of the Jekyll website, as well as the preCICE team and its community for providing content and technical improvements. In particular, we would like to thank Muhammad Aashir Aslam for the migration to Hugo, as part of a Google Summer of Code 2026 project, developed while this manuscript was in review. We would also like to thank the attendees of the corresponding session at the deRSE26 conference for the constructive and encouraging discussion and recommendations.

Data availability: The source code is available on https://github.com/precice/precice.github.io, with this manuscript corresponging to the v202609.0.0 tag. The described components of the preCICE ecosystem are regularly archived on DaRUS, as part of the preCICE Distribution. Latest available version: v2404, available on https://doi.org/10.18419/DARUS-4167.

Declaration on the use of AI: No generative AI was used in the preparation of this text or the underlying software. For language checks, we acknowledge https://languagetool.org/. Community contributions to the code might have incorporated GenAI in fixing issues and improving features.

[57] See https://github.com/precice/precice.github.io/pull/979 (result of a Google Summer of Code 2026 project)
[58] This is not effort that every project can invest, or the benefits might not be relevant. The main idea of distributing the content remains the same, and is readily applicable to any multi-repository project currently serving its documentation using Jekyll or potentially other static website generators.

Bibliography

[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

[BH13] W. Bangerth, T. Heister. What makes computational open source software libraries successful? *Computational Science & Discovery* 6(1):015010, Nov. 2013.
doi:10.1088/1749-4699/6/1/015010

[CDR+22] G. Chourdakis, K. Davis, B. Rodenberg, M. Schulte, F. Simonis, B. Uekermann, G. Abrams, H.-J. Bungartz, L. Cheung Yau, I. Desai, K. Eder, R. Hertrich, F. Lindner, A. Rusch, D. Sashko, D. Schneider, A. Totounferoush, D. Volland, P. Vollmer, O. Z. Koseomur. preCICE v2: A sustainable and user-friendly coupling library [version 2; peer review: 2 approved]. *Open Research Europe* 2(51), 2022.
doi:10.12688/openreseurope.14445.2

[CSU25] G. Chourdakis, V. Seitz, B. Uekermann. System Regression Tests for the preCICE Coupling Ecosystem. *Electronic Communications of the EASST* 83, Feb. 2025.
doi:10.14279/eceasst.v83.2614

[HF22] S. Hermann, J. Fehr. Documenting research software in engineering science. *Scientific Reports* 12(1):6567, Apr. 2022.
doi:10.1038/s41598-022-10376-9

[Lee18] B. D. Lee. Ten simple rules for documenting scientific software. *PLOS Computational Biology* 14(12):1–6, Dec. 2018.
doi:10.1371/journal.pcbi.1006561

[Pro25] D. Procida. Diátaxis documentation framework. May 2025.
https://diataxis.fr/

[Wri26] Write the Docs community. Software documentation guide. Accessed: May 2026.
https://www.writethedocs.org/guide/