Documenting Research Software in Engineering Science

Sibylle Hermann (sibylle.hermann@itm.uni-stuttgart.de)
University of Stuttgart
Jörg Fehr
University of Stuttgart

Research Article

Keywords:

Posted Date: January 13th, 2022

DOI: https://doi.org/10.21203/rs.3.rs-1239393/v1

Version of Record: A version of this preprint was published at Scientific Reports on April 21st, 2022. See the published version at https://doi.org/10.1038/s41598-022-10376-9.

Documenting Research Software in Engineering Science

Sibylle Hermann[1,2,*] **and Jörg Fehr**[1]

[1]University of Stuttgart, Institute of Engineering and Computational Mechanics, Stuttgart, 70569, Germany
[2]University of Stuttgart, Cluster of Excellence SimTech, Stuttgart, 70569, Germany
[*]sibylle.hermann@itm.uni-stuttgart.de

ABSTRACT

The reuse of research software needs good documentation, however, the documentation in particular is often criticized. Especially in non-IT specific disciplines, the lack of documentation is attributed to the lack of training, the lack of time or missing rewards. This article addresses the hypothesis that scientists do document but do not know exactly what they need to document, why, and for whom. In order to evaluate the actual documentation practice of research software, we examined existing recommendations, and we evaluated their implementation in everyday practice using a concrete example from the engineering sciences and compared the findings with best practice examples. In order to get a broad overview of what documentation of research software entailed, we defined categories and used them to conduct the research. Our results show that the big picture of what documentation of research software means is missing. Recommendations do not consider the important role of developers, whose documentation takes mainly place in their research articles. Moreover, we show that research software always has a history that influences the documentation.

Introduction

Documentation of research software[1] in engineering science is inadequate[2]. Nevertheless, researchers–particularly within the FAIR (Findable, Accessible, Interoperable, Reusable) movement–state that documentation of research software as a major prerequisite for reuse[3]. Although, research data and software play a central role in the Cluster of Excellence "Data-Integrated Simulation Science (SimTech)"[4], documentation is also lacking here.
But why is research software documented poorly? And what does good documentation actually imply? Previous approaches provide rather explanatory models why documentation is not done; they explain the missing documentation with lack of time[5] or insufficient training[6]. But is this really the case? Is it even clearly defined what documentation entails? Until now, incentives and rewards are missing for well documented research software. But, the scientific environment is changing, Gil et al.[7] observe a shift in the scientific environment in different areas: scientific publishing, scientists, public interest and funding. In recent years, these developments are gaining momentum with initiatives like EOSC (European Open Science Cloud)[8] and NFDI (National Research Data Infrastructure, Germany)[9]. Moreover, research funding agencies demand reusability; the guidelines for good scientific practices, for example, require documentation of research software explicitly[10]. Surprisingly, it remains rather unclear what good documentation of research software involves. We illustrate that there are recommendations on how to document (research) software. But are the recommendations actually applied?
The hypothesis of this paper is, that it is still unclear what good documentation actually involves. The approach intends to examine how documentation takes place in everyday work in a research environment in engineering science within the Cluster of Excellence, SimTech. We also examine if and how given recommendations are implemented. We defined categories to represent different documentation purposes. Based on these categories, we examined three different aspects:

- Given recommendations on how research software should be documented.
- An actual documentation workflow of a specific research software project from engineering science within SimTech.
- Given documentation of two best practice examples within SimTech.

Previous approaches have been concerned with reasons why research software is poorly documented, but not with what good documentation actually entails. Also, it has not been investigated what and how documentation must be implemented in order to be perceived as good. It's the intention of this approach to show what is missing and give an overview on who has to document for whom what, where, when, and how.

Methods

We want to investigate how research software is documented in a field where scientists usually don't have a computer science background. Due to the highly disciplinary nature of research software, we focused on our own discipline, Engineering Science. We followed a mixed quantitative and qualitative approach to compare the quantitative analysis of a literature review with the qualitative evaluation of specific research software. An evaluation of literature was conducted in order to assess what recommendations are given. To see how the documentation is actually implemented, one specific research software was chosen: Neweul-M^2[11], a research software that has been developed for over thirty years in an institute by engineers without formal software development training. Neweul-M^2 continues to be actively developed and is often used to address specific research questions. We assume that the documentation for other research software projects is not significantly different. To give a good insight, we solve a concrete task, based on the given documentation. Then the findings were compared to the documentation habits of other research software projects: two best practice examples were evaluated with semi-structured interviews and analysis of the research software documentation. These best practice examples were chosen from research areas targeted by SimTech, furthermore, they received funding from the DFG (German Research Foundation) sustainable software funding call to improve the documentation[12] Both best practice examples are open source. DuMu^x[13] is a research software from engineering sciences, which is also programmed by non-software developers. The other example preCICE[14] is a research software developed from more experienced software developers. Two main research questions structure the investigation.

Research Questions

RQ1: What are the recommendations for documenting research software? Which rules and best practices exist? Do the given recommendations cover the defined categories?

RQ2: What is the practice of documenting research software? How is research software documented in the daily life of researchers? Which workflows are implemented? What are the obstacles to document research software?

Research Method

Our first idea was to evaluate the research software using given recommendations from the literature. As we soon noticed, the recommendations do not give a complete picture of what documentation should actually contain. Therefore, we have formed categories that we consider necessary from everyday work with research software, supplemented with categories from literature and internet resources like blogs and wikis. To answer the research questions, we defined four documentation categories for research software, intending to provide a picture of possible documentation forms. Based on the defined categories, we evaluated the recommendations given for, Neweul-M^2, DuMu^x, and preCICE. In the following, we introduce the categories, followed by the recommendations and conclude with the analysed research software examples.

Categories

Domain Research software can belong to different domains[15]: *private*, *shared* and *open*. Usually, research software is developed in the *private* domain with one main developer. The *shared* domain varies from a few users at an institute to many users all over the world, nevertheless the research software is unavailable to the broader public. Published research software in the *open* domain is accessible for everyone. Where *open* can have two different meanings: only the source code is available open source or the software is developed openly. The domains may change over time and require more documentation, as more people need to understand the research software.

Role As we noticed that it is important who documents for whom, we differentiate between three roles: *maintainer*, *developer* and *user*. One person can have multiple roles, the role can be shared by multiple people, and the role of a person can change. *Maintainers* are responsible for the infrastructure and maintenance of the software, they can give the rules how research software should be written, but are often not part of the active feature development – their problems are discussed within the research software engineers (RSE) movement[16]. *Users* want to use the software without writing code. *Developers* link between *maintainers* and *users*, they develop new features, mainly to answer–with the research software–specific research questions. *Developers* are mostly without education in software development and are often less experienced than maintainers. Nevertheless, they are an important part of the documentation process, as they only have knowledge about functions, they develop, but are usually not deeply involved in the maintenance and documentation process.

Purpose The purposes describe the content of the documentation: *problem*, *feature* and *implementation*. The documentation of the *problem* should describe "why" the research software or a feature is written–similar to describing the research question. The documentation of the *feature* should describe "what" is needed to be done in order to solve the research question. "How" the feature is *implemented* should be documented in a technical documentation.

Type The type describes the characteristics of the documentation[17]: *decision*, *product* and *technology*. The three above introduced categories can be expressed in different types of documentation: The documentation of the *decision* can involve how the problem is implemented and why a solution was preferred. The *product* documentation contains the list of all features provided by the software and how they work together. The *technical* documentation should help the developer and maintainer to understand the code, how the research software is engineered and how to build over the existing source code. It should contain different schemas about the used model, the logical, and physical architecture.

We assume that each of these categories requires a different type of documentation. Aspects of good research software, and its documentation, has also been addressed in various recommendations.

Recommendations

In order to find recommendations for documenting (research) software, we conducted a literature review in Web of Science using the terms "research software" and "documentation" or "reusability". Most articles refer to the whole process of developing research software, and not only to documentation. Often just one small paragraph is dedicated to documentation. The selection of the articles was limited to those that include rules or best practices for documenting research software in at least one paragraph. Ten articles with interdisciplinary and different disciplinary focus were found. As described above, the evaluation of the recommendations did not provide a complete picture of what the documentation in our opinion should contain. Therefore, we also analysed the recommendations according to the categories we defined.

Analysis of research software

Neweul-M^2 Neweul-M^2 is a software package that allows the dynamic analysis of mechanical systems in calculating multibody systems with symbolical equations[18]. The first version of Neweul was written in FORTRAN with an own symbolic formula manipulator engine in the mid 1970s and was rewritten in 2003 using MATLAB. The new version is called Neweul-M^2. In Kurz et al.[11], the history and changes are documented until the year 2010. Neweul-M^2 is used from:

- external people (user)
- PhD students (developer and user)
- students (user)

The source code is developed and administrated by PhD students within the developing institute, they aim for a degree in mechanical engineering and usually don't have a formal software development education. One experienced developer is the maintainer, a new colleague is briefed as maintainer from the previous one.

For the external people and students, a content-obscure (P-code) version of Neweul-M^2 is provided. One part of the documentation is in an integrated help within MATLAB. The help includes a product description, tutorials and examples and a function reference, automatically generated from the code. For PhD students from the developing institute, the full source code is accessible. The source code is managed via a Git repository hosted at an institutional GitLab instance. Bug fixes and support are the responsibility of the maintainer. Another part of the documentation is done in a local wiki with information on how to get started and how to document with coding guidelines, tests and checklists. Decisions and discussions are documented via GitLab. For the maintainer, an additional document gives information on how everything is organized. PhD students, who use Neweul-M^2 for their research, develop new features in Neweul-M^2 that they need for their research. They document these features mainly in publications.

DuMu^x The research code for the free and open-source simulator is written in C++ and is based on the DUNE (Distributed and Unified Numerics Environment) environment. DuMu^x stands for "DUNE for Multi-{Phase, Component, Scale, Physics, } flow and transport in porous media[19]. The main intention is to provide a sustainable and consistent framework for implementing and applying of porous media model concepts and constitutive relations. All documentation is linked on the Website. The documentation consists of a collection of documented code examples within the institute's publicly accessible GitLab instance, a manual in PDF format, code documentation within Doxygen, a reference to the most important publications and a wiki that is still under construction. The software is written by PhD candidates in civil engineering with a predominantly engineering background, who have taught themselves to program.

preCICE The research software preCICE[20] is an open-source coupling library for partitioned multi-physics simulations, including fluid-structure interaction and conjugate heat transfer simulations. The research is about methods how two systems can be coupled. In preCICE the research question is not solved with the software, rather the research results are provided to users of the software. For this reason, there are only maintainers and users. The software is written by PhD candidates with different backgrounds, who are aiming for a degree in computer science. The documentation is bundled in a website. By using GitHub pages, pull requests can be made to all the documentation. The website is divided in quick start, docs, tutorials,

community and blog. The section docs start with a user documentation with fundamentals, installation, configuration, tooling and provided adapters. The API is described in the category "couple your code". Followed by a developer documentation, with a link to the source code documentation in Doxygen, and a description of coding conventions, tooling, workflow and testing. Even a description, how the documentation is build, exists. For the users–in addition to the conventional documentation–a community page gives insights on workshops, other contributors and publications. Furthermore, there is a blog where there is also the possibility to ask questions.

Results

We present three main observations from the recommendations and the documentation of three research software examples, based on the categories that are presented in the method section.

Observation 1: A Big Picture is missing

A big picture is missing on what documentation of research software should contain and how it should be done. The examined recommendations focus only on specific aspects of research software documentation (see Figure 1).

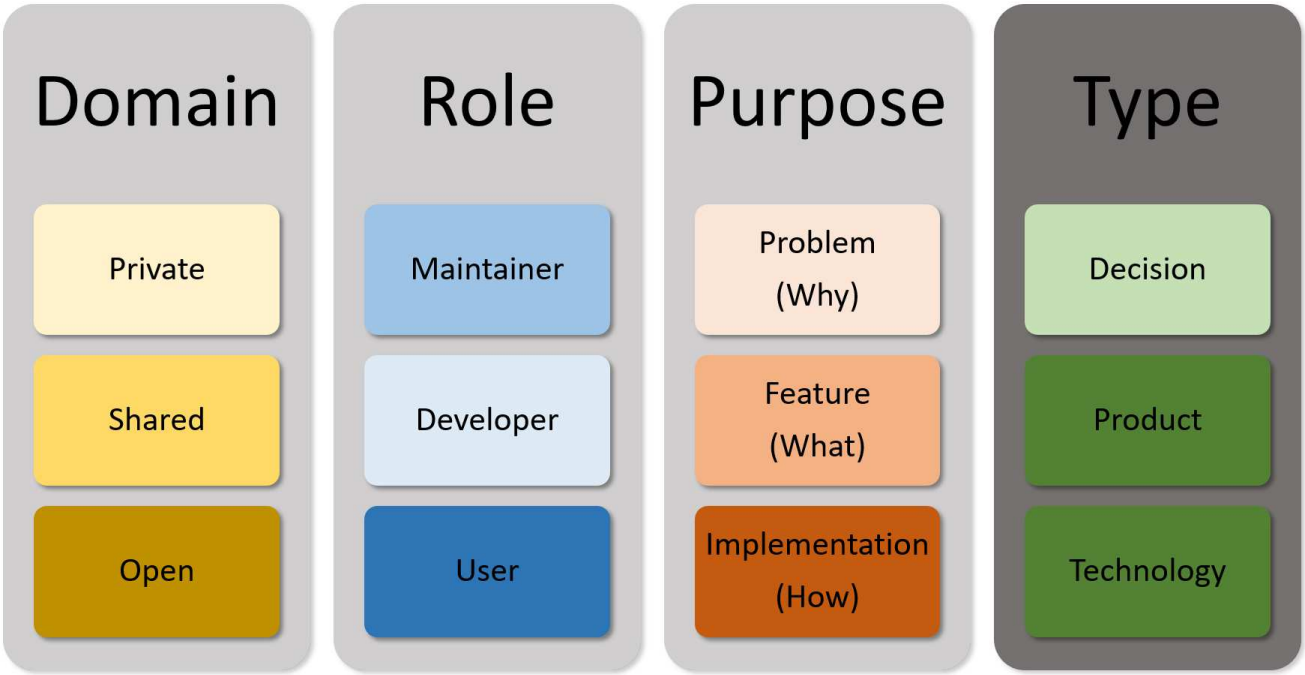

Figure 1. Different aspects of documentation in recommendations. The shading of the colors–darker means deeper focus–represents how much the content deals with our defined categories: They concentrate on the open domain, address mainly documentation for users and focus on the implementation of the research software.

Observation 1.1: Problem and decision are undocumented

The recommendations rarely mention that decisions should be documented and seldom take the underlying problem into account (see Table 1). They refer–according to our categories–mainly to the implementation of the research software. Their focus is on techniques (like version control systems and programs that generate a documentation out of comments in the code) not on content. Lee[6] state for example to "use automated documentation tools" and that "the best type of documentation is documentation that writes itself", but do not explain what have to be the content of the automated documentation. Looking into the practice, all three examples use these automated documentation tools.

In Neweul-M^2 (see Table 2) the function reference within the help is created with a given template, including: short description, syntax, long description, parameters, examples and references. In DuMu^x (see Tab. 3) the modules are documented with Doxygen[21]. In comparison to Neweul-M^2 the description involves the underlying concept, mostly explaining the formula behind the code. In preCICE (see Table 4) as well, Doxygen is used with a generic documentation template: the parameters and a one line description is needed and an optional elaborate description. Here, the description do not contain the underlying concept. These tools are intended to document the code not the decisions and problems: Developers document the problem

and decision in research articles or thesis, which are not linked to the documentation; they implement new features to solve a specific task for a thesis; and they describe the problem only in the thesis referring to a specific version of the software. Eventually, the description of the problem and the feature differ from the software solution. Once the feature is included in the main branch, the dependencies are further maintained.

Article	Problem	Feature	Implementation
Stodden V, Miguez S.[22]			x
Fehr J, and Heiland J.[23]			x
Wilson G, Aruliah DA, et al.[24]	x	x	x
Wilson G, Bryan J, et al.[25]			x
Hastings J, Haug K, Steinbeck C.[26]		x	x
Lee BD.[6]		x	x
Taschuk M, Wilson G.[2]		x	x
Sandve GK, Nekrutenko A, et al.[5]	x	x	x
Karimzadeh M, Hoffman MM.[27]			x
Gil Y, David CH, et al.[7]		x	x

Table 1. Purpose mentioned in the recommendations

Documentation	Problem	Feature	Implementation
Wiki		x	x
Help		x	x
Local GitLab	x	x	x
Maintainer			x
Publications	x	x	

Table 2. Purpose of the different Neweul-M^2 documentations

Documentation	Problem	Feature	Implementation
Examples		x	
Handbook		x	x
Code	x		x
Publications	x	x	
Wiki			x

Table 3. Purpose of the different DuMu^x documentations

Documentation	Problem	Feature	Implementation
User Docs		x	x
Dev docs		x	x
API			x
GitHub	x	x	x
Community		x	x
Blog	x	x	x
Publications	x		x

Table 4. Purpose of the different preCICE documentations

Observation 1.2: Shared and private domain are neglected

The recommendations focus on the open domain (see Table 5). Especially, the documentation for the shared domain is rarely mentioned. Neweul-M^2 is a research software from the shared domain, the different documentation types live in different domains (see Table 6). Developers document mainly for their successors at the own institute. But the knowledge is not only transported by documentation: students of the institute learn about the software in their lectures and later on from their supervisor and fellow students or colleagues. In the best practice examples, the documentation is openly available.

Article	Private	Shared	Open
Stodden V, Miguez S.[22]		x	x
Fehr J, and Heiland J.[23]			x
Wilson G, Aruliah DA, et al.[24]	x	x	
Wilson G, Bryan J, et al.[25]	x		
Hastings J, Haug K, Steinbeck C.[26]		x	
Lee BD.[6]			x
Taschuk M, Wilson G.[2]	x		x
Sandve GK, Nekrutenko A, et al.[5]	x		x
Karimzadeh M, Hoffman MM.[27]			x
Gil Y, David CH, et al.[7]			x

Table 5. Domains mentioned in the recommendations

Documentation	Private	Shared	Open
Wiki		x	
Help		x	
Local GitLab		x	
Maintainer	x		
Publications			x

Table 6. Domains of the different Neweul-M^2 documentations

Observation 2: Research software has a history

In engineering science, PhD candidates usually stay round about six years, become experts in a very specific field and then leave. Successors–interested in the same topic–often do not have the chance to talk to them. So the experts omit feedback on their documentation and are ignorant of which questions they have to address in their documentation.

Figure 2 describes the observed effect in Neweul-M^2, showing the quality of the research software documentation over the different phases of research software development. The quality of the documentation behaves similarly to Kondratiev waves[28]: prosperity, recession, depression, and improvement. In the case of Neweul-M^2 one researcher developed the research software to solve a specific problem in the initiation phase. In the maturation phase, other researchers adopt the research software and more people get involved in the project. At the beginning, the documentation was good (enough) for the people who use the software (prosperity). Eventually, the initial developer left, and new researchers added new features and modified the code. The quality of the documentation decreased (recession) in the saturation phase because modifications were not documented (depression); until a point was reached where the research software needed a refactoring. A new documentation is needed, usually written with a new tool (improvement). But the old documentation is still used because some aspects are important in there: different documentations exist for different roles in different places with different up-to-dateness. The descriptions of the implemented features in the articles referring to the research software before the refactoring are now more difficult or even impossible to reproduce. A new researcher inherits this history. Depending on the dynamic of the research software, the cycles are more or less distinct and happen more or less fast. The best practice examples have a similar history. They benefit of being open source, receive more feedback from users outside the institute, and spend more effort and money to improve the documentation. So the effect of unconscious knowledge can be minimized.

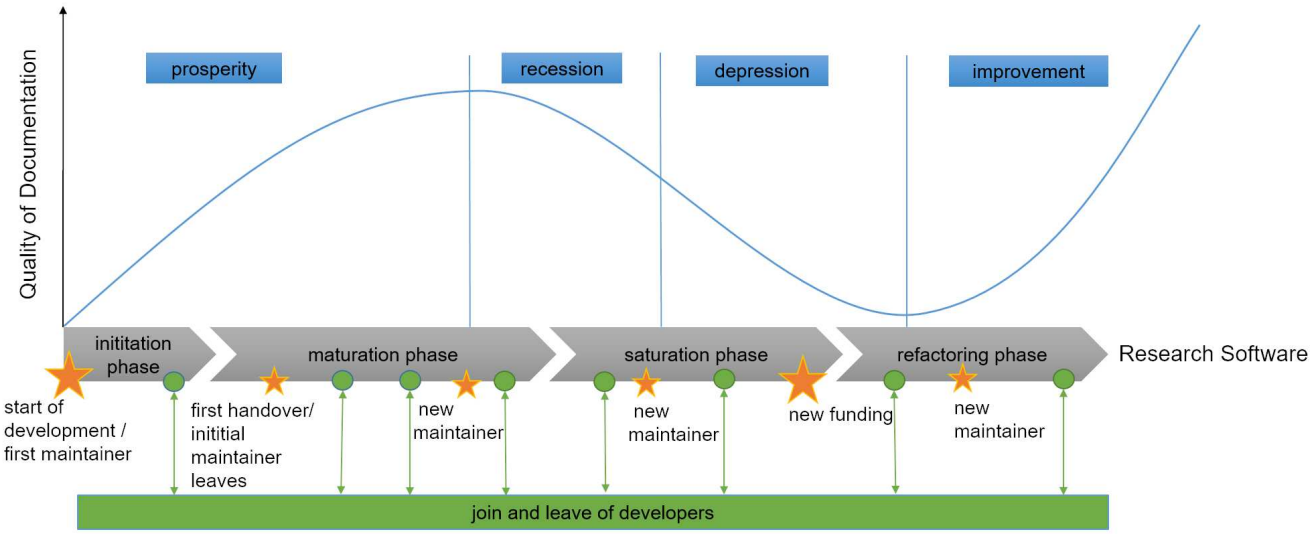

Figure 2. Quality of research software documentation over time. The research software undergoes different phases, from first implementation to continuous application. During these phases, the quality of the documentation varies accordingly. In particular, the change of maintainer involves risks, but can also lead to improvements.

Observation 2.1: Unconscious Knowledge

In Neweul-M^2 new users and developers are inducted to the software with the help of an experienced developer or the maintainer. The knowledge of experienced developers is often unconscious, that means that they are not aware of their own knowledge and therefore do not explain important steps to the new developer[29]. The effect shows up for example in the description of the workflow. It is described in the help theoretically but not concretely how and where information is stored and called. Relevant information about what is written in which files are presupposed. Usually, the maintainer or experienced developer compensates this divergence. For example, where and how storing files is explained in a lecture about Neweul-M^2, but this information is totally missing in the documentation. In the best practice examples the problem is less pronounced, because users from outside ask questions and draw attention to the problem, and especially in preCICE there is a workflow to update the help, according to the asked questions. Both projects have improved user-friendliness of the documentation through third-party funding. The recommendations do not address this problem.

Observation 2.2: Missing Consistency

In Neweul-M^2 the documentation lives in different places: An integrated MATLAB help, mainly intended for users; an internal Wiki with more information, mainly intended for developers; and an internal document for the maintainer with storage locations, workflows and pieces of the history. Not all the documentation is updated with changes in the code. Developers usually archive their documentation with their thesis in a zip-file. This kind of documentation often refers to an obsolete version, inherited from the history. Moreover, outdated dependencies, which are not documented, invalidate the function or a lot of effort must be spent to fix the dependencies. Looking at the best practice examples: one of the first issues was to have all in one place. Through feedback from external users, the best practice examples are more consistent. Moreover, they spend money and effort to meet these challenges. In DuMu^x all the documentation is linked on the homepage, an overview where to find which information is missing. In preCICE the documentation is directly on the homepage, with an overview where to find what. Additionally, there is even meta information about the documentation itself. The recommendations give hints about documentation tools, which can be used–how to structure this information is mentioned in[27].

Observation 3: Research software has different purposes

Researchers write research software for different purposes. Therefore, the focus of the documentation can differ as well. The purpose of Neweul-M^2 is to implement a physical model to evaluate the same effects that occur in or beyond experiments. Engineers use already developed algorithms to solve their research question using research software. Here, in addition to documentation for the users, documentation for the developers is also necessary, because the scientists at the institute need to understand the results of their predecessors and want to be able to adapt them to their own needs.

The purpose of DuMu^x is similar. However, there is a larger community outside the own institute, which uses the software and develops it accordingly.

The purpose of preCICE is to implement new algorithms and to show that these algorithms work. In preCICE the role of developer is not specifically taken into account. Users do couple their software with the help of preCICE but do not contribute to the code with features. So the documentation is mainly intended and improved for scientific users (see Table 7).

	Maintainer		Developer		User
Documentation	Author	Audience	Author	Audience	Audience
User Docs	x				x
Dev Docs	x	x			
API	x	x			x
GitHub	x	x			x
Community	x				x
Blog	x				x
Publications	x	x			x

Table 7. Authors and audience of the different preCICE documentations

Observation 3.1: Developers are neglected

Developers are often neglected as authors and as audience. As authors of the documentation, the recommendations consider mainly maintainers; the audience are maintainers and users (see Table 8). The documentation requirements for developers are unclear. While in practice some requirements are given, the documentation reality often differs. In Neweul-M^2 the maintainer has formalized the documentation of and for the developers through a given template. Developers document directly in the code, which is automatically transferred to the help. The description often remains very short and are sometimes insufficient to be understood by others. No feedback from the successors is given about the quality of the documentation, because the developers leave the institute and no longer notice possible problems (see Table 9).

In DuMu^x developers have as well a guideline how and what they have to document. Experience shows that instead of following the guidelines, developers tend to keep the effort as small as possible and do not describe as expected, especially if the requirement is considered unnecessary (see Table 10).

	Maintainer		Developer		User
Article	Author	Audience	Author	Audience	Audience
Stodden V, Miguez S.[22]	x	x			x
Fehr J, and Heiland J.[23]	x	x			x
Wilson G, Aruliah DA, et al.[24]			x	x	
Wilson G, Bryan J, et al.[25]			x	x	
Hastings J, Haug K, Steinbeck C.[26]	x		x	x	x
Lee BD.[6]	x	x			x
Taschuk M, Wilson G.[2]	x	x			x
Sandve GK, Nekrutenko A, et al.[5]	x	x			x
Karimzadeh M, Hoffman MM.[27]	x	x			x
Gil Y, David CH, et al.[7]	x	x			x

Table 8. Authors and audience of documentation mentioned in the recommendations

	Maintainer		Developer		User
Documentation	Author	Audience	Author	Audience	Audience
Wiki	x			x	
Help	x		x	x	x
Local GitLab	x	x	x	x	
Maintainer	x	x			
Publications	x				

Table 9. Authors and audience of the different Neweul-M^2 documentations

	Maintainer		Developer		User
Documentation	Author	Audience	Author	Audience	Audience
Examples	x			x	x
Handbook	x	x			x
Code	x			x	
Publications					
Wiki	x			x	

Table 10. Authors and audience of the different DuMu^x documentations

Discussion

The results undoubtedly show that research software is documented. Our study shows that not necessarily the motivation or missing skills lead to the opinion that software is not documented; rather, research software is not documented as expected.

Often the main problem is that documentation is seen as an event and not a process. The path dependency described in the results as well as the missing consistency and missing framework can be mitigated by setting **uniform standards**. This will always be a balancing act between freedom of research and predefined framework. However, the movement in given structures allows a more efficient work. Also, writing and documenting are not in itself the actual research work, but only the framework in which the research takes place. Researchers have other goals in writing and documenting code than professional software developers. Software developers aim to achieve the objectives defined in the product requirements: Specifying these requirements can be seen as a part of the documentation. Research software, on the other hand, is a means to an end and is not documented in product requirements. Researchers aim to answer research questions with the help of software[30]. Consequently, one part of the documentation happens in research articles, doctoral, master and bachelor theses. Those can be seen as **delayed product requirements**. Nevertheless, this kind of documentation is focused on the research question and not on the research software. Moreover, research papers discuss scientific results based on research software; but the research software behind the results is quickly outdated and developed further. Above all, the **precise implementation of physic** into code in the research software is not specified[31], but particularly this point is essential for reusing the research software. Besides, articles document research results, not software. Sometimes the material is also **not accessible or difficult to find**. Certainly the lack of time to document is critical, but at a later stage much more time needs to be invested to support the users[27] and to understand the research software as a developer. Some papers argue with the lack of training of researchers in software engineering[2,6]. But even in professional software development, documentation is neglected. Ludewig and Lichter[32] see two reasons for this neglection: Firstly, documentation is not necessarily learned even in software developer training and secondly, although it is said that documentation is important, other aspects usually have priority. Websites like "write the docs"[33] and The blog "I'd Rather be Writing"[34] gives advice for technical writers how to document code. Some approaches can certainly be adopted, though not everything can be transferred one-to-one to research software. Initiatives like "Better Scientific Software"[35] and the "Software Sustainability Institute"[36] draw attention to the problem and provide assistance. Although these sites are certainly helpful, you need to know them. They give only possible assistance and are not in themselves a standard. Nevertheless, a generally applicable standardization of documenting research software is difficult to find. Existing standards from software engineering[37] are complex, in parts not relevant for research software and difficult to access. Even if a standardization will be helpful to share results openly, it needs a **clear guideline** to document results for oneself and in a group. Therefore, all three examined research software have some forms of standardized templates, testing strategies and checklists provided by the maintainer. However, compliance must also be checked, which in turn costs time. Especially, in engineering science several points add to the described difficulties:

- use of other funding possibilities
- existence of confidentiality reasons
- fear of sabotaging the business model
- modification of existing software, which is unclear how to document

But working together with industry demands good documented results–independent from publishing the software. Moreover, other developers need the documentation in order to understand the work from their predecessors. In Neweul-M^2 developers contribute to code and documentation. They have to understand the code, and they have to develop new features, which have to be documented. This part of the documentation can not be written but be controlled by a maintainer. Developers depend on the documentation of their predecessors and the existing structure. This experience happens as well in DuMu^x. It can also be

discussed whether some problems described could be avoided by making the software open source. There are good reasons, such as confidentiality obligations and also often a business model, that make these steps undesirable. Nevertheless, from our point of view, some investigated methods can be transferred to the shared domain.

Conclusion

All in all, it can be said that it should be clear who documents what and where. Hence, adopting best practices and principles from technical documentation and professional software development can help to improve the documentation of research software. Future research should explore how principles from the best practices examples can be transferred into the shared domain. A possible standardization of content would certainly be helpful here, but this cannot be solved by the individual scientist. The national and international initiatives certainly contribute to improving the situation here. One limitation of the current research is that the findings are not evaluated with more examples. This obstacle can be overcome in evaluating more software documentations. It can be expected that other research software in engineering science has similar problems. Moreover, there is a personal bias when trying to solve the problem with a given documentation. The experience showed that it was totally clear for the maintainer of the help where they can find the information and how the documentation is structured. But for inexperienced users, it is not obvious, they have to ask the maintainer. The effort to write documentation should be taken into account. Will the benefit exceed the effort that must be used for documentation? This can be an area for future research. As long as people are there who can help, it is just inefficient but not impossible to solve the given task without a sufficient documentation. But in the current discussion about FAIR, research software documentation plays an important role. The pay-off for developers is may be marginal at the moment, but the importance is increasing. Good documentation pays off in the long run.

We discovered that researchers are often not aware for whom and why they document. A big picture what documentation for research software means is missing. The new approach in this paper is to define for what purpose and what appearance the documentation is intended and who has to document what for whom depending on the domain. The paper shows that even in recommendations, the objective of the documentation of research software is unclear. Until now the focus lies on maintainer and user, the researcher who develops features to an existing research software is here brought into focus. While essentially only the open domain has been considered so far, a substantial part of research software does not take place publicly in the first place; here, too, documentation is needed in order to ensure sustainable research.

References

1. FAIR for Research Software (FAIR4RS) WG. Research Data Alliance Workinggroup. Online (Accessed 7 Jan 2022). https://www.rd-alliance.org/group/fair-research-software-fair4rs-wg/outcomes/fair-principles-research-software-fair4rs.
2. Taschuk, M. & Wilson, G. Ten simple rules for making research software more robust. *PLOS Comput. Biol.* **13**, e1005412, DOI: https://doi.org/10.1371/journal.pcbi.1005412 (2017).
3. Chue Hong, N. P. *et al.* Fair principles for research software (fair4rs principles), DOI: https://doi.org/10.15497/RDA00065 (2021).
4. SimTech. The Cluster of Excellence SimTech. Online (Accessed 7 Jan 2022). https://www.simtech.uni-stuttgart.de.
5. Sandve, G. K., Nekrutenko, A., Taylor, J. & Hovig, E. Ten simple rules for reproducible computational research. *PLoS Comput. Biol.* **9**, DOI: https://doi.org/10.1371/journal.pcbi.1003285 (2013).
6. Lee, B. D. Ten simple rules for documenting scientific software. *PLOS Comput. Biol.* **14**, e1006561, DOI: https://doi.org/10.1371/journal.pcbi.1006561 (2018).
7. Gil, Y. *et al.* Toward the geoscience paper of the future: Best practices for documenting and sharing research from data to software to provenance. *Earth Space Sci.* **3**, 388–415, DOI: https://doi.org/10.1002/2015ea000136 (2016).
8. EOSC. European Open Science Cloud. Online (Accessed 7 Jan 2022). https://eosc.eu/.
9. NFDI. National Research Data Infrastructure Germany. Online (Accessed 7 Jan 2022). https://nfdi.de.
10. Deutsche Forschungsgemeinschaft. Guidelines for Safeguarding Good Research Practice. Code of Conduct, DOI: https://doi.org/10.5281/zenodo.3923602 (2019). Available in German and in English.
11. Kurz, T., Eberhard, P., Henninger, C. & Schiehlen, W. From neweul to neweul-m2: Symbolical equations of motion for multibody system analysis and synthesis. *Multibody Syst. Dyn.* **24**, 25–41, DOI: https://doi.org/10.1007/s11044-010-9187-x (2010).
12. DFG. Sustainability of research software. Online (Accessed 7 Jan 2022). https://www.dfg.de/foerderung/info_wissenschaft/2016/info_wissenschaft_16_71/.

13. Koch, T. *et al.* DuMux 3 – an open-source simulator for solving flow and transport problems in porous media with a focus on model coupling. *Comput. & Math. with Appl.* **81**, 423–443, DOI: https://doi.org/10.1016/j.camwa.2020.02.012 (2021).

14. Bungartz, H.-J. *et al.* preCICE – a fully parallel library for multi-physics surface coupling. *Comput. & Fluids* **141**, 250–258, DOI: https://doi.org/10.1016/j.compfluid.2016.04.003 (2016).

15. Treloar, A., Groenewegen, D. & Harboe-Ree, C. The data curation continuum. *D-Lib Mag.* **13**, DOI: https://doi.org/10.1045/september2007-treloar (2007).

16. Anzt, H. *et al.* An environment for sustainable research software in germany and beyond: Current state, open challenges, and call for action. *F1000Research* **9**, 295, DOI: https://doi.org/10.12688/f1000research.23224.1 (2020).

17. Oliveira, V. How to write good software technical documentation.*Medium*. Online (Accessed 7 Jan 2022). https://medium.com/@VincentOliveira/how-to-write-good-software-technical-documentation-41880a0e7814.

18. Neweul-M^2. Software package for the dynamic analysis of mechanical systems in MATLAB. Online (Accessed 7 Jan 2022). https://www.itm.uni-stuttgart.de/en/software/neweul-m/.

19. DuMu^x. DUNE for Multi-{Phase, Component, Scale, Physics, }. Online (Accessed 7 Jan 2022). https://dumux.org/.

20. preCICE. The coupling library for partitioned multi-physics simulations. Online (Accessed 7 Jan 2022). https://precice.org.

21. Doxygen. Generate documentation from source code. Online (Accessed 7 Jan 2022). https://www.doxygen.nl.

22. Stodden, V. & Miguez, S. Best practices for computational science: Software infrastructure and environments for reproducible and extensible research. *J. Open Res. Softw.* **2**, DOI: https://doi.org/10.5334/jors.ay (2014).

23. Fehr, J., Heiland, J., Himpe, C. & Saak, J. Best practices for replicability, reproducibility and reusability of computer-based experiments exemplified by model reduction software. *AIMS Math.* **1**, 261–281, DOI: https://doi.org/10.3934/math.2016.3.261 (2016).

24. Wilson, G. *et al.* Best practices for scientific computing. *PLoS Biol.* **12**, e1001745, DOI: https://doi.org/10.1371/journal.pbio.1001745 (2014).

25. Wilson, G. *et al.* Good enough practices in scientific computing. *PLOS Comput. Biol.* **13**, e1005510, DOI: https://doi.org/10.1371/journal.pcbi.1005510 (2017).

26. Hastings, J., Haug, K. & Steinbeck, C. Ten recommendations for software engineering in research. *GigaScience* **3**, DOI: https://doi.org/10.1186/2047-217x-3-31 (2014).

27. Karimzadeh, M. & Hoffman, M. M. Top considerations for creating bioinformatics software documentation. *Briefings Bioinforma.* **19**, 693–699, DOI: https://doi.org/10.1093/bib/bbw134 (2017).

28. Barnett, V. *Kondratiev and the dynamics of economic development : long cycles and industrial growth in historical context* (St. Martin's Press, in association with Centre for Russian and East European Studies, University of Birmingham, New York, 1998).

29. Ambrose, S. A., Bridges, M. W., DiPietro, M., Lovett, M. C. & Norman, M. K. *How Learning Works: seven research-based principles for smart teaching* (Jossey-Bass, a Wiley Imprint, 2010).

30. Segal, J. & Morris, C. Developing scientific software. *IEEE Softw.* **25**, 18–20, DOI: https://doi.org/10.1109/ms.2008.85 (2008).

31. Hinsen, K. Verifiability in computer-aided research: The role of digital scientific notations at the human-computer interface. *PeerJ Comput. Sci.* **4**, e158, DOI: https://doi.org/10.7717/peerj-cs.158 (2018).

32. Ludewig, J. & Lichter, H. *Software Engineering* (Dpunkt.Verlag GmbH, 2013).

33. Docs as Code. Write the Docs. Online (Accessed 7 Jan 2022). https://www.writethedocs.org/guide/docs-as-code/.

34. Johnson, T. I'd Rather Be Writing. Blog (Accessed 7 Jan 2022). https://idratherbewriting.com/.

35. BSSw. Better Scientific Software. Online (Accessed 7 Jan 2022). https://bssw.io/.

36. Chue ong, N. e. a. Software Sustainability Institute. Online (Accessed 7 Jan 2022). https://www.software.ac.uk/about/.

37. ISO/IEC JTC 1/SC 7 Software and Systems Engineering, ISO/IEC JTC 1/SC 7, ISO/CEI JTC 1/SC 7. Systems and software engineering - systems and software quality requirements and evaluation (SQuaRE) - guide to SQuaRE. Standard (2014).

Acknowledgements

Funded by Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2075 390740016. We acknowledge the support by the Stuttgart Center for Simulation Science (SimTech). We also want to thank the maintainers of the three examined research software examples Bernd Flemisch, Georg Schneider and Benjamin Uekermann for their support and helpful inputs.

Author contributions statement

SH studied the research software documentation, developed the methodology, and wrote the article. JF contributed to the writing process through valuable discussion and feedback, as well as his own experience in documenting research software. All authors read and approved the final manuscript.

Additional information

Competing interests

The authors declare that they have no competing interests.