Received: 31 May 2019 | Accepted: 5 July 2019

DOI: 10.1002/pamm.201900480

Requirements for Finding Research Data and Software

Sibylle Hermann[1,*], **Dorothea Iglezakis**[1], and **Anett Seeland**[1,2]

[1] University Library of Stuttgart

[2] Information and Communication Technology, University of Stuttgart

Research results in simulation engineering are primarily based on software - from small scripts written by a single researcher to big software projects developed at the researchers institute or by an international community. Usually the research results are published in a publication, whereas the underlying software and data resulting from the software is not available. Consequently, the results which are discussed in an article can be difficult to reproduce. Initiatives like the FAIR data principles try to give an idea how research data can be stored in order to reuse research results. This article describes how the FAIR data principles may be applied for research software in simulation engineering and give an idea how infrastructure services can help researchers handling their research software.

1 Dealing with research data and software

In the research process it can be an arduous task to reproduce results published in an article. In simulation technology, results are generated by software and codes. In this article we use the term software for more extensive programs, in comparison to the term code that refers to smaller scripts and programs. The results are visualized in a plot and the process of getting these data is described textually. Hours of programming are often needed in order to receive approximately the same results as the people who have already done this task. Although it would help to have access to the data to recreate the results, most important to reproduce the results are the underlying assumptions, parameters, and software codes [1]. But even available code can be difficult to understand. Software is usually shared and developed with others rather than published directly. Therefore, the software should be described in a way that everyone involved can understand it. Recommendations for handling of research software [2] therefore aim at a long-term availability and identification of specific software versions in addition to software development strategies and documentation of code. The Software Heritage Archive [3] tries to preserve the content of all openly available code repositories. Nevertheless, the software needs to be described and be accessible in a way that it can be found and used. The name of the software or the code file does often not describe the content accurately enough in order to find the results you are looking for. Hence, the FAIR-data-principles were developed to give rules how research data and software must be handled in order to be Findable, Accessible, Interoperable and Reusable [4] [5]. This article discusses, how the FAIR Data principles can be adopted for simulation engineering, and how data can be described according to the FAIR principles, so that research data and software can be reused not only after publication, but also within shared systems.

2 FAIR-Data-Principles for research software

Initially the FAIR-data principles were intended for published research data, whereby research software is seen as one aspect of research data. These principles apply primarily to metadata, enabling information about the research data or code to be understood by humans and machines. Wilkinson et al. [4] states that the principles are intended to be evolved into rules and standards for different disciplines.

For a software to be *findable*, there has to be a possibility to identify a specific version of software or code. Moreover, a search index is needed, where the software can be looked for. Globally unique and eternally persistent identifiers (PIDs) are assigned by a global registration organization. An example are Digital Object Identifiers (DOI), which are identifiers in the form 10.XXXXX/YYYYYYYYY, where XXXXX is a prefix registered by a publisher and YYYYYYYYY a suffix. DOIs have to be assigned by the responsible organization: Institutions apply for a prefix and can assign DOIs with this prefix and a self-designed suffix. Moreover, the organizations are responsible that the DOI points to the real location of the resource. But DOIs are intended only for published resources. To permanently identify code versions, that are not published, other PID systems like EPIC PID[1] can be used. DOIs are normally assigned by repositories, that manage not only the actual data but also metadata, e.g., a description of the data. For published resources, the metadata include at least the information necessary for a citation, such as author(s), title, year, and publisher. But metadata can also include other information for instance the implemented methods, the programming language or a description of the input and output of the code. A research data repository functions primarily as a metadata database and should not be confused with a version control system, such as Git, that manages code repositories. The purpose of a research data repository is the searchable and persistent storage of a publication, for example of software or code. It is not made for joint code development as it lacks typical features like branching and

* Corresponding author: e-mail sibylle.hermann@ub.uni-stuttgart.de, phone +0049 711 685 82502, https://www.izus.uni-stuttgart.de/fokus/

[1] see https://www.pidconsortium.eu/, last checked on May, 27th, 2019

merging. The integration of GitHub (as a code repository) into Zenodo (as a research data repository) [2] is an example of how specific versions (releases) of a software can be published and cited. Conveniently, a research data repository also functions as a search index. To make your code and software findable, you have to upload the specific version to a repository, describe the software with suitable metadata and receive a persistent identifier. The difficulties start when you try to identify suitable metadata. Thus, the first question to be answered is what to be searched for? Apart from properties of the software itself, such as the authors, the programming language and the version of the used software, research software should be findable by content based search criteria like the problem solved through this software, the method implemented and its parameter sets.

The greatest benefit of FAIR data and software is noticeable when data and code can be *reused*. When data can be reused without difficulty, it can be a catalyst for the own research, making it straightforward to build upon existing data and code, to teach new group members or to improve and revise a colleague's work. However, reusability adds additional requirements to the structured description and deposition of the data and code. Further information are necessary to understand whether and how the data might be useful. Provenance information can help because they describe the process of creation and processing of the data. In the context of code, input and output information are relevant, how the software is started and parametrised, as well as details about used algorithms, numerical assumptions and known limitations. Licence information in the metadata help to reuse the code in an appropriate way. Licenses are to be considered particularly, since the use of the software is fixed by the license. So it is not only important to use the right license for your own software, but also to consider which license the code has, that you have used yourself. Copyleft licenses, [3] such as the GNU General Public License (GPL) [4] require that software with this license cannot be used in a closed source project, which can be a problem especially in research projects with industry involvement. Another requirement concerns the executability of the code. Code may depend on specific libraries, on the operating system (OS) or even on specific hardware. Thus a perfect description will not help, if the user does not have these dependencies and/or hardware. Additionally, complex installation routines with various libraries in specific versions may discourage potential users.

To be *interoperable*, not only the code itself should be in a language that is shared and broadly applicable, but also the metadata should be structured and meet recognized standards. Metadata has two implications: Format and content. Both need standardization that machines can interpret the key-value pairs and humans can search for specific terms. The format is not the problem, there are enough machine readable formats, for example JSON. The problem is to find an ontology which is generic enough in that it can be understand by many people and on the same time specific for the researcher community to really have a chance to search for relevant aspects of their work. For a descripton of (research) software by properties of the software itself, there is CodeMeta [6] as a recommondation for a standard metadata description. CodeMeta bases on Schema.org [5] and extends the properties of the schema.org data types SoftwareSourceCode and SoftwareApplication with further information, such as build instructions, maintainer of the software or embargo dates [6]. But while CodeMeta provides a standard for code properties, there is no such standard for content description of scientific problems to be solved with the code. Therefore, we developed the metadata scheme EngMeta [7] for the description of research data from computational engineering (see 3).

To be *accessible*, the location of the code or software should be accessible to anyone who is authorized to access it. The audience can be everyone for published resources or also a research or project group sharing a code base. Granting access does not mean, that the data should be openly available. In order to regulate the access, the code must be deposited in a suitable location. The location depends on who needs to have access on it. There exist different possibilities: The own PC, a shared device or a public repository. Even on the own PC the code must be described in a way that I can still find it in a year. On shared devices, different people can access the code. Therefore the data need an unique identifier and metadata which make it possible for others to find the software. The last possibility is to publish the software, either on a global or institutional repository or on a subject specific one. There the code normally gets an unique identifier. Actually, the metadata should be accessible even if the data is not published. Furthermore, the metadata should give information if and how the data is accessible. The metadata should be readable by humans and machines.

3 Implementation at the University of Stuttgart

The metadata scheme EngMeta [7] can be used for the description of research data from computational engineering and is developed within the project Dipl-Ing[7]. EngMeta[8] is made for research data but is also applicable to describe research software

[2] see https://guides.github.com/activities/citable-code/, last checked on May, 27th, 2019

[3] for more information see https://opensource.com/resources/what-is-copyleft, last checked on May, 27th, 2019

[4] see https://www.gnu.org/licenses/gpl.html

[5] see https://schema.org, last checked on May, 27th, 2019

[6] see https://codemeta.github.io/terms/ for a complete list of terms

[7] funded by the Federal Ministry of Education and Research with grant no. FDM16-008, see https://www.ub.uni-stuttgart.de/dipling for more information, last checked May, 27th 2019

[8] see https://www.izus.uni-stuttgart.de/fokus/engmeta, last checked May 27th, 2019 for the schema, an example file and documentation of EngMeta

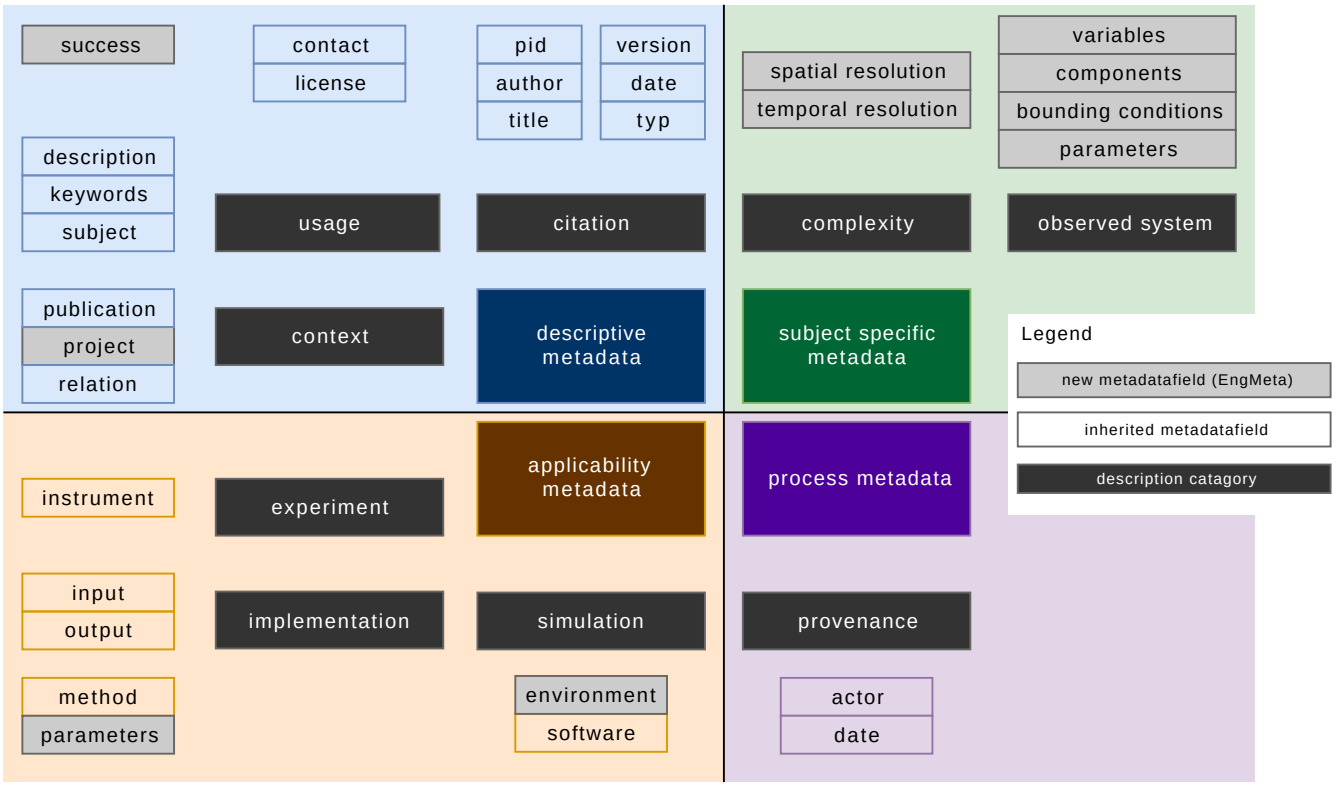

Fig. 1: Categories and terms of EngMeta applicable for research software in the fields descriptive

in terms of functionality and subject. A subset of the categories and terms of EngMeta applicable to software are shown in Figure 1. The fields highlighted in gray are newly defined fields that do not yet appear in any other metadata schema. The other transparent fields are adopted from existing metadata schemas.

EngMeta allows not only to describe resources by general *descriptive metadata* but also by *subject specific metadata* like information about the simulated system (by components, variables and parameters) and the complexity of the simulation (by spatial and temporal resolution). *Applicability metadata* specify the scope of application of a software in more detail: Implemented methods, supported instruments, possible input and output, the targeted computing environment and software dependencies. Provenance information like author and date of changes is specified in the *process metadata*.

We integrated EngMeta into our Data Repository DaRUS[9], a repository basing on Dataverse[10]. Dataverse is an open source software for research data repositories. Although Dataverse is mainly intended for the publication of data, it also offers the possibility to describe, manage and share unpublished resources through a detailed user and role management.

A solution for reusability of research results could be to deposit the research data with the research code and the environment including all dependencies. Various container solutions can accomplish this task, such as, Docker[11], Singularity[12], Firecracker[13], or Kata containers[14]. Every solution has its specific pros and cons. We are currently working in the DFG-funded project SusI (Sustainable infrastructure for the improved usability and archivability of research software on the example of the porous-media-simulator DuMux) on publishing research software with Docker and making the software available in an executable form. For single code files, such as scripts, without specific OS dependencies, hubs can be useful to run code with a few clicks from the data repository. Examples are JupyterHub[15], BinderHub[16], or ShinyApps[17]. An essential criteria for reusability is code comprehensibility, simple rules as clear interfaces, meaningful variables names and rather short functions or methods helps others to understand the results.

To support coding quality, especially for those who did not learn programming as part of their studies, the infrastructure services offer key qualification courses and software carpentry workshops [18] . Because a standard only makes sense if everyone

[9] see https://darus.uni-stuttgart.de,last checked May 27th, 2019
[10] see https://dataverse.org/, last checked May 27th, 2019
[11] see https://www.docker.com/, last checked May 27th, 2019
[12] see https://www.sylabs.io/singularity/,last checked May 27th, 2019
[13] see https://firecracker-microvm.github.io/,last checked May 27th, 2019
[14] see https://katacontainers.io/,last checked May 27th, 2019
[15] see https://jupyter.org/hub,last checked May 27th, 2019
[16] see https://binderhub.readthedocs.io/en/latest/,last checked May 27th, 2019
[17] see https://www.shinyapps.io/,last checked May 27th, 2019
[18] see https://software-carpentry.org/,last checked May 27th, 2019

uses it, we get involved in various initiatives. We are involved in the application of the National Infrastructure for Engineers (NFDI4ING) [19], which aims, among other things, to develop a metadata standard for engineers. The proposed interest group for Engineers at the Research Data Alliance (RDA) [20] will also deal with metadata standards and research software for engineers.

4 Conclusion

Research software and code underlying a publication published in a FAIR way has the potential to significantly facilitate and accelerate research in simulation engineering. Not only is the reproducibility of research results noticeably improved by software and code that is findable, identifiable and understandable but also the reusability of this code for other research questions. It is important to concentrate not only on published code, but also to start with the documentation during the research process. The focus should be on describing the software in a way that other people can understand it. Metadata schemes helps to structure and standardize this information to automate many steps via interfaces, which previously had to be done manually. Although a sustainable description of the research software and data is associated with additional effort at the beginning, it can simplify a lot of documentation work and tedious searching later on. Infrastructure services can provide support for a sustainable research data and software management. Standards for publishing research software and code can be developed in cooperation with infrastructure services and researchers but the sharing of code needs a broader discussion in the researcher community. On the infrastructure side, there is cooperation in research data management between the technical universities in Germany and Europe. The aim is to establish a networked infrastructure with shared standards. Since there is no point in conducting this discussion only in the infrastructure facilities, the discussion must be continued together with and within the disciplines. Only with shared standards added value can be created, which then leads to automated and sustainable management of research results.

References

[1] J. Fehr, J. Heiland, C. Himpe, and J. Saak, AIMS Mathematics **1**(3), 261–281 (2016).

[2] F. Bach, W. zu Castell, M. Denker, A. Finke, B. Fritzsch, M. Hammitzsch, U. Konrad, Y. Leifels, C. Möhl, M. Nolden, M. Scheinert, T. Schlach, T. Schnicke, and D. Steglich, Empfehlungen zur Implementierung von Leit- und Richtlinien zum Umgang mit wissenschaftlicher Software an den Helmholtz-Zentren, Tech. rep., Helmholtz Gemeinschaft, November 2017.

[3] R. D. Cosmo and S. Zacchiroli, Software heritage: Why and how to preserve software source code, in: iPRES 2017 - 14th International Conference on Digital Preservation, (Kyoto, Japan, September 2017), pp. 1–10.

[4] M. D. Wilkinson, M. Dumontier, I. J. Aalbersberg, G. Appleton, M. Axton, A. Baak, N. Blomberg, J. W. Boiten, L. B. da Silva Santos, P. E. Bourne, J. Bouwman, A. J. Brookes, T. Clark, M. Crosas, I. Dillo, O. Dumon, S. Edmunds, C. T. Evelo, R. Finkers, A. Gonzalez-Beltran, A. J. Gray, P. Groth, C. Goble, J. S. Grethe, J. Heringa, P. A. t. Hoen, R. Hooft, T. Kuhn, R. Kok, J. Kok, S. J. Lusher, M. E. Martone, A. Mons, A. L. Packer, B. Persson, P. Rocca-Serra, M. Roos, R. van Schaik, S. A. Sansone, E. Schultes, T. Sengstag, T. Slater, G. Strawn, M. A. Swertz, M. Thompson, J. van der Lei, E. van Mulligen, J. Velterop, A. Waagmeester, P. Wittenburg, K. Wolstencroft, J. Zhao, and B. Mons, Scientific Data **3**(March), 160018 (2016).

[5] C. Erdmann, N. Simons, R. Otsuji, S. Labou, R. Johnson, G. Castelao, B. V. Boas, A. L. Lamprecht, C. M. Ortiz, L. Garcia, M. Kuzak, P. A. Martinez, L. Stokes, T. Honeyman, S. Wise, J. Quan, S. Peterson, A. Neeser, L. Karvovskaya, O. Lange, I. Witkowska, J. Flores, F. Bradley, K. Hettne, P. Verhaar, B. Companjen, L. Sesink, F. Schoots, E. Schulte, R. Kaliyaperumal, E. Toth-Czifra, R. de Miranda Azevedo, S. Muurling, J. Brown, J. Chan, N. Quigley, L. Federer, D. Joubert, A. Dillman, K. Wilkins, I. Chandramouliswaran, V. Navale, S. Wright, S. D. Giorgio, A. M. Fasemore, K. Förstner, T. Sauerwein, E. Seidlmayer, I. Zeitlin, S. Bacon, K. Hannan, R. Ferrers, K. Russell, D. Whitmore, and T. Dennis, Zenodo (2019).

[6] M. B. Jones, C. Boettiger, A. C. Mayes, A. Smith, P. Slaughter, K. Niemeyer, Y. G. Gil, M. Fenner, K. Nowak, M. Hahnel, L. Coy, A. Allen, M. Crosas, A. Sands, N. C. Hong, P. Cruse, D. Katz, and C. Goble, Codemeta: an exchange schema for software metadata. version 2.0., 2017.

[7] B. Schembera and D. Iglezakis, The Genesis of EngMeta - A Metadata Model for Research Data in Computational Engineering, in: Metadata and Semantic Research, edited by E. Garoufallou, F. Sartori, R. Siatri, and M. ZervasNo. 846 in Communications in Computer and Information Science (Springer International Publishing, Limassol, 2019), pp. 127–132.

[19] see https://nfdi4ing.de/,last checked May 27th, 2019

[20] see https://rd-alliance.org/groups/research-data-management-engineering-ig,last checked May 27th, 2019

