

Software Management Plans as a Path to Sustainable and Reproducible Research Software – Potentials, Discussions and Obstacles in Dealing with Code in Science



Michael Franke¹, franke@mpdl.mpg.de, <https://orcid.org/0000-0002-2661-8242>
Yves Vincent Grossmann¹, grossmann@mpdl.mpg.de, <https://orcid.org/0000-0002-2880-8947>

¹ Max Planck Digital Library, Collections, Landsberger Straße 346, 80687 Munich, Germany

1. Research Software

- Software is often required for the reproducibility of scientific results or the result itself
- There are different reasons to make software available, i.e. as an autonomous publication
- DFG Guideline 13: “Software programmed by researchers themselves is made publicly available along with the source code.”
- Internal guidelines and/or general regulations might require/recommend a software publication
- Increasing focus on software by third-party funders as an important project outcome

2. Objectives for SMPs

- Low-threshold offer for the software project organisation with a focus on science
- Management tool to promote explicit use of research software
- SMP-Definition by DINI/nestor: “A software management plan (SMP) contains general and technical information about the soft-w are project, information on quality assurance, release and public availability as well as legal and ethical aspects that affect the software.” (<https://forschungsdaten.info/praxis-kompakt/english-pages/software-management-plans/>)

3. Advantages Through SMP Use

- Transforming implicit knowledge into explicit knowledge
- Tool for project management with focus on research software
- SMP as a service for scientists and research software engineers
- Use for consulting services, e.g. by local IT, scientific computing unit, third-party funding office
- Quality management and assurance
- Third-party funding applications
- Better overview of software project in an organisation

4. SMP User Groups

Scientists, who:

- have not yet dealt much with software management
- would like to achieve quality in research software with little time investment

Information specialists from:

- IT, Scientific Core Unit
- Third-party funding applications
- Project and quality management
- Research coordination

5. Discussions about SMPs

- Chue Hong et al. (2014): “Writing and using a software management plan”, <https://www.software.ac.uk/resources/guides/software-management-plans>.
- Martinez-Ortiz et al. (2022): Practical guide to Software Management Plans, <https://doi.org/10.5281/zenodo.7248877>.
- Giraldo et al. (2023): Workshop machine-actionable Software Management Plans. <https://doi.org/10.5281/zenodo.8087357>.
- Grossmann and Franke: “Software ist kein Beiprodukt! Nachhaltige Forschungssoftware durch Softw are-Management-Pläne”, in: b.i.t. online 26/5 (2023), pp. 457–463.

6. Some Available SMPs Tools

- ELIXIR
 - [In https://smw.ds-wizard.org](https://smw.ds-wizard.org)
 - <https://doi.org/10.37044/osf.io/k8znb>
- PRESOFT
 - <https://dmp.opidor.fr>
 - <https://doi.org/10.5281/zenodo.1405614>
- RDMO
 - i.e. on <https://rdmo.mpg.de>
 - <https://doi.org/10.17617/2.3496327>
- Train-the-Trainer materials for teaching SMPs: <https://doi.org/10.5281/zenodo.10197107>

7. RDMO SMP Project by MPDL

- Team from MPDL Collections
- Result: CC0 push of an SMP catalogue as a contribution to the RDMO community
- Title: Software Management Plan for Researchers
- In German and English
- From 9 to 50 questions
- With CC0 on <https://github.com/rdmorganiser/rdmo-catalogue> available
- All questions are also available as .docx via <https://doi.org/10.17617/2.3481986>

8. Structure RDMO SMP Catalogue

- General
 - a.o. persons involved, resources
- Technical information
 - a.o. code, infrastructure, security
- Quality assurance
 - Testing, documentation, etc.
- Release and public availability
 - a.o. releases, metadata
- Legal and ethical issues
 - a.o. copyright, licenses, dual use

9. Special Features from RDMO SMP

- Open Source
 - Free to use without any restrictions
 - Set-up for your own or use existing infrastructure
 - Continuous maintenance by the community
 - No vendor login, instead many open export and import options
 - Own contributions to the community are welcome
- Individual customisation, especially the help text, for your own institution
- Scaling of the scope of questions
 - depending on the complexity of the software (from simple plot to large infrastructure)
- FAIR4RS-Viewer in RDMO available to FAIRify your research software

10. Selected Screenshots

Third Party Components and Libraries

Which external software components will be used? What dependencies on software libraries do exist? How do you document that?

Overview

Progress

Navigation

Question on monitoring external components

In which application class is the software categorised?

Application class 0: The focus of the software is on personal use in conjunction with a small scope. The distribution of the software within and outside the own institution is not planned.

Application class 1: The software is only developed within a narrow scope. It is to be further developed and used beyond personal purposes.

Application class 2: The software is intended to ensure long-term development and maintainability. It is the basis for a transition to production status.

Application class 3: For the software it is essential to avoid errors and to reduce risks. This applies in particular to critical software and that with product characteristics.

Initial question on the application class of the code and corresponding scaling of the SMP question to be answered

Software Development Requirements

Are there institutional requirements for software development?

For the development of software in a scientific context, the rules of good scientific practice at one's own institution or organisation apply. Software might not explicitly mentioned there. Nevertheless, these rules for scientific results can apply to the development (and sharing) of software.

Below you will find a collection of examples of institutional regulations related to software:

- Budisch & Funk (2022): Software Licensing and Copyright Policy for Research Software CODE @ Max Planck Institute for Meteorology, <https://hdl.handle.net/21.1116/0000-0000-8091-A>.
- European Space Agency ESA Open Source Policy, <https://esa.esa.int/esa-opensourc-policy>.
- Akhmerov et al. (2021): TU Delft Research Software Policy, <https://doi.org/10.5281/zenodo.4629662>.

Question about the preconditions for software development

Do you apply specific coding standards? How do you take care about code quality control?

When developing software, it can be helpful to adhere to certain standards. These can be, for example, common code conventions in the professional community or language-specific standards (e.g. Java Coding Standards, Clean Code Developer). Tools for static code analysis (e.g. Lint) can help to find problematic code. An (internal) code review can also be a suitable method of quality control.

Do you have a style guide for Python code?

Do you have a coding standards document?

Do you have a code review process?

Question on coding standards

Where will the software be stored? Does the storage place have a clear preservation policy?

An adequate place should be found for the storage of the resulting software. Depending on the requirements and possibilities, this can range from a simple, local solution to a specialised software repository.

One far-reaching option for the long-term preservation of software is, for example, Software Heritage (for more repository examples, see also this article <https://doi.org/10.48550/arXiv.2012.13117>). There are other places, that offer public, long-term availability. For example, reData can be used to search for specific repositories through which software can be published.

Giving information about the preservation

Publicly Availability

Will this software be publicly available?

Software is increasingly perceived as a relevant outcome of the scientific knowledge process. Thus, the Second French plan for Open Science foresees that software will become the third pillar in scientific publishing, alongside text and data publications.

Guideline 13 in the DFG Code “Guidelines for Safeguarding Good Scientific Practice” explicitly recommends that “Software programmed by researchers themselves is made publicly available along with the source code”.

Yes No

Question regarding public availability

Will users have the possibility to contribute to your software?

It is reasonable for software to be handed over to a scientific community for collective processing once it has reached a certain level of maturity. Such decisions should be planned well in advance. Experience shows that this can either mean a lot of communication work or no one taking notice of this possibility. For this purpose, for example, the corresponding Git system could enable external developer to contribute to the code.

Questions regarding public availability

Is (Open) Peer Review planned for the software?

For software in a scientific context, it is reasonable to subject it to an (open) review process. This can be in the form of a review or also a certification. For scientific software review, for example, there is the Journal of Open Source Software. Besides this, subject-specific standards for reviewing scientific software are also becoming established, for example in archaeology.

Further literature references on the topic may also be of interest:

- Code review checklist used at the Max Planck Institute of Psychiatry for the CodeClub, https://pad.gwdg.de/_/u/0391m1ymMxZ0bAe794g/.
- Early and Carver (2022): Developers perception of peer code review in research software development, in: Empirical Software Engineering 27, 13, <https://doi.org/10.1007/s10664-021-10053-8>.
- Borou (2013): Modelling Modern Code Review Practices in Open Source Software Development Organizations, IJDESE 13 Baltimore, <https://www.researchgate.net/publication/313105085/figure/fig/1/figure-fig1/1516221100000/Borou-2013-IJDESE-138.pdf>.

Question regarding test strategy

Which software test strategy are you going to follow? Which types of tests are planned for the project?

It can be helpful to be clear about a testing strategy in advance. This avoids the ad-hoc search for errors, but leads to more targeted, structured testing. Following on from this, you need to be clear about which test principles you want to apply. In parallel, it should be clarified whether standards (e.g. ISO/IEC 25000:2014, ISTQB) apply. Testing a software on different levels (Unit, Integration, E2E-Tests) is always useful. Depending on the project, different types of testing (security testing, functional tests, non-functional tests, performance testing etc.) are important. Ministry of Testing, as a global community for software testing, is a useful first start to approach this topic.

Unit Testing

Integration Tests

E2E Tests

Security Testing

Functional Tests

Non-Functional Tests

Performance Testing

ISTQB

ISO/IEC 25000:2014

Other:

Section of the FAIR4RS viewer

Findability

F: Software and its associated metadata are easy for both humans and machines to find

F1: Software is assigned a globally unique and persistent identifier

F1.1: Depending on the software representing level of granularity, an assigned distinct identifier is

- A single software component can be identified implicitly via the version number and the source file name.
- Versioning will be handled as follows:

F2: Software is described with rich metadata

F2.1: Different versions of the software are assigned distinct identifiers

- The following identifiers are used:

F3: Metadata clearly and explicitly include the identifier of the software they describe

- The following identifiers are used:

F4: Metadata are FAIR, searchable and retrievable

- Searchability of the software metadata is granted by the platform where the software is stored.

Accessibility

A: Software, and its metadata, is retrievable via standardized protocols

A1: Software is retrievable by its identifier using a standardized communications protocol

- A1.1: The protocol is open, free, and universally implementable
- A1.2: The communication protocol is determined by the platform where the software is stored.
- A1.3: The protocol allows for an authentication and authorization procedure, where necessary
- A1.4: The communication protocol is determined by the platform where the software is stored.

A2: Metadata are accessible, even when the software is no longer available

- A long-term accessibility of the software metadata is granted by the platform where the software is stored.

Interoperability

I: Software interoperates with other software by exchanging data and/or metadata, and/or through interaction via application programming interfaces (APIs), described through standards

I1: Software reads, writes and exchanges data in a way that meets domain-relevant community standards

- The following coding standards and practices are followed:

I2: Software includes qualified references to other objects

- The software refers on the following external services:

