

Practical guide to Software Management Plans




netherlands
eScience center

DOI: [10.5281/zenodo.7038280](https://doi.org/10.5281/zenodo.7038280)



Table of content

| | | |
|------|---|----|
| 1. | About this Document | 4 |
| 2. | Introduction | 6 |
| 2.1. | Research software supports Open Science | 8 |
| 3. | What is Research Software? | 10 |
| 4. | Benefits of a Software Management Plan | 12 |
| 5. | Core Requirements for Developing a Software Management Plan | 14 |
| 5.1. | List of core requirements | 14 |
| 5.2. | Guidance | 18 |
| 5.3. | SMP rubric | 19 |
| 6. | Implementation Examples | 20 |
| 6.1. | Using subsets of the core requirements to define customised SMP templates | 20 |
| 6.2. | Types of software that require different levels of management | 30 |
| 6.3. | SMP templates generated for three software management levels | 31 |
| | Acknowledgements | 34 |



1. About this Document

This document provides guidance to create a Software Management Plan (SMP) template. It is intended for anyone who is involved in the development, support, and/or management of research software, including researchers, research software engineers, research supporters, funders, and policy makers in fields involving research software as a scholarly output.

What is a Software Management Plan?

An SMP is a document that describes how a specific software project is developed, maintained, and curated. The goal of an SMP is to ensure that the software is usable and maintainable in the long term. An SMP is written by the developers, maintainers, and/or other stakeholders of a software project.

An SMP template is a document that prescribes which information is required or expected in an SMP, in the form of specific questions to be answered by the project maintainers. An SMP template can be provided by, for example, research groups, research organisations, and funding agencies to ensure that researchers consistently adhere to certain software management standards and policies when developing research software.

How to read this document

The document is divided into six sections. The introduction in Section 2 explains how there is growing recognition of the importance of research software. Section 3 discusses the definition of research software used in this guide, and is followed by Section 4, which highlights the benefits of SMPs. Section 5 describes core requirements for SMPs and provides resources to guide researchers and research support staff in fulfilling these requirements. Finally, Section 6 provides a framework for implementing the core requirements into SMPs. It also guides the reader in choosing suitable subsets of requirements to create SMP templates for different types of software that require different levels of management (low, medium, and high). This section also includes an example SMP template for each of the three management levels.

2. Introduction

Researchers increasingly rely on software in their research.^{1,2,3} For example, a survey by the Software Sustainability Institute, carried out among UK researchers, found that 92% of academics use research software, and 7 out of 10 researchers deemed it impossible to conduct their research without it. Research software comes in many forms and applications, from scripts that generate and collate data to large libraries that build complex reports.^{4,5} Research software is used in all research domains, from astronomy⁶ to theology⁷, and in all phases of research.

1 <https://www.software.ac.uk/blog/2014-12-04-its-impossible-conduct-research-without-software-say-7-out-10-uk-researchers>

2 <https://newscience.org/how-software-in-the-life-sciences-actually-works-and-doesnt-work/>

3 <https://blog.esciencecenter.nl/evidence-for-the-importance-of-research-software-1cb4a4907713>

4 <https://github.com/ESMValGroup/ESMValTool>

5 <https://github.com/garretti403/SciencePlots>

6 <https://www.astropy.org/>

7 <https://github.com/historical-theology/awesome-theology>



Many significant scientific discoveries strongly rely upon the use of software. For example:

- In 2016, a three-dimensional geographic information system (3D GIS) was used to generate virtual reconstructions of architecture along the Via Appia, contributing to researchers' ability to understand and reconstruct complex archaeological sites.⁸
- In 2019, the first ever image of a black hole was created by the Event Horizon Telescope. This breakthrough was made possible by means of software that combined data from a network of telescopes around the globe.⁹
- In 2020, the deep learning programme AlphaFold was used to predict the 3D structure of proteins based on their amino-acid sequence for the first time with an accuracy that approaches experimental structure determinations. This has been called the breakthrough of a lifetime for its field.¹⁰

8 <http://mappingtheviaappia.nl/3dgis/>

9 <https://numfocus.org/case-studies/first-photograph-black-hole>

10 <https://www.nature.com/articles/d41586-020-03348-4>

2.1. Research software supports Open Science

There is growing consensus across different stakeholders, from research performing organisations to research funders, that research software must be recognised as an important output of research.¹¹ Stimulated by technological innovations and the democratisation of science, researchers, funders, and governments have launched initiatives to advance transparency under the broad umbrella of the Open Science movement. In its most basic form, Open Science urges researchers to make all outputs of research, including primary and the intermediate outputs, publicly accessible.

The ability to reproduce results in order to assess the reliability of findings is an integral part of the research process, as is the possibility of building upon those results. For the sake of research transparency, reproducibility, reuse, and recognition, research software should be shared by the authors of a study in such a way that it can be used to obtain the same results as in the original work or extend that work.

Open Source software has a long history of contribution to the advancement of research. The values of quality and integrity, and principles of transparency and reproducibility, sit at the heart of the UNESCO Recommendation on Open Science.¹² The same Recommendation identifies Open Source software as one of the key elements of open scientific knowledge. Open Source software, alongside practices laid out in the FAIR principles, promotes collaboration among researchers, stakeholders, and the public. Furthermore, it enables software creators to extend or expand upon existing software instead of creating (yet another) stand-alone piece of software. By encouraging researchers to do so, research organisations and funders can save on investment of funds and researchers' time.

¹¹ "Software must be recognised as an important output of scholarly research" arXiv:2011.07571

¹² <https://unesdoc.unesco.org/ark:/48223/pf0000379949.locale=en>

Multiple initiatives, such as the Research Software Alliance (ReSA), and the European Open Science Cloud (EOSC), are looking into improving research software management, reusability, and sustainability,^{13,14} with the aim of contributing to Open Science. Meanwhile, research institutions and funders are working on the necessary policies,¹⁵ provision of support, and guidance to include research software as part of the rewards and recognition system; and to facilitate the adoption of good practices.

Ideally, research software should be as open as possible, but as closed as necessary. All research software, whether open or closed source (for example, in case of security concerns or commercial interests), can benefit from using SMPs.



¹³ <https://www.rd-alliance.org/groups/fair-research-software-fair4rs-wg>

¹⁴ <https://www.eosc.eu/advisory-groups/infrastructures-quality-research-software>

¹⁵ TU Delft Research Software Policy: <https://doi.org/10.5281/zenodo.4629661>

3. What is Research Software?

There are ongoing efforts to define what research software is.^{16,17,18,19} Creating a formal definition of research software is beyond the scope of this document. For the purposes of this document, we will utilise the definition of research software from the FAIR for Research Software Working Group (a joint initiative of RDA, ReSA and FORCE11):²⁰

¹⁶ The Four Pillars of Research Software Engineering <https://doi.org/10.1109/MS.2020.2973362>

¹⁷ On the evaluation of research software: the CDUR procedure <https://doi.org/10.12688/11000research.19994.2>

¹⁸ The Research Software Encyclopedia: A Community Framework to Define Research Software <http://doi.org/10.5334/jors.359>

¹⁹ Engineering Academic Software (Dagstuhl Perspectives Workshop 16252) <http://doi.org/10.4230/DagMan.6.11>

²⁰ Defining Research Software: a controversial discussion <https://doi.org/10.5281/zenodo.5504015>



Research Software includes source code files, algorithms, scripts, computational workflows and executables that were created during the research process or for a research purpose. Software components (e.g., operating systems, libraries, dependencies, packages, scripts, etc.) that are used for research but were not created during or with a clear research intent should be considered software in research and not Research Software.



This is admittedly a broad definition of what research software can include and is provided as a starting point for people involved in research software projects. Scripts, notebooks, source code, executables, containers all may be considered research software depending on the context, but there is no universal agreement. The readers of this guide may decide for themselves whether this definition suits their purposes.

4. Benefits of a Software Management Plan



Research software is an integral part of the research process and several aspects of its development, maintenance and curation should be planned for. Data Management Plans (DMPs) have been used for many years to ensure that good data management practices are followed.^{21,22} In recent years, SMPs are also becoming increasingly common.^{23,24,25}

An SMP is a document detailing how research software will be managed, usually as part of a project. An SMP makes explicit what research software does, who it is for, what the outputs are, who is responsible for the release and to ensure that the software stays available to the community (and for how long).

An SMP can help to establish a structured way of developing research software. By asking relevant questions, an SMP can also help to ensure

21 Practical Guide to the International Alignment of Research Data Management, <https://doi.org/10.5281/zenodo.4915861>

22 Data Management Plan Catalogue, <https://libereurope.eu/working-group/research-data-management/plans/>

23 Checklist for a Software Management Plan, <https://doi.org/10.5281/zenodo.1422656>

24 Checklist for a Software Management Plan, <https://doi.org/10.5281/zenodo.1422656>


25 ELIXIR Software Management Plan for Life Sciences, <https://doi.org/10.37044/osf.io/k8znb>

that research software is (and remains) accessible and reusable. More specifically, an SMP can help to:

- Explain why developing new software is necessary. New software should not be developed when it would be more cost-efficient and beneficial for the overall community to contribute to existing software.²⁶
- Make the research software reusable and sustainable. An SMP encourages software developers to think about, for example, technical choices (such as programming language or operating system dependencies); whether the right documentation and metadata are provided (e.g. to allow for reproduction or extension of an analysis); and to ensure that the software is findable and adequately licensed for reuse for an extended period of time.
- Plan for the necessary resources. Various types of resources exist: financial, human, infrastructure, etc. Whenever reusing, creating or building upon research software in a research project, additional resources might be needed. The questions in an SMP can help to predict which resources will be needed for developing and maintaining the software (e.g., hiring research software engineers, training), for making the software available to others (e.g., infrastructure) and for making and keeping the software accessible over time.
- Allow for verification of work that went into software implementation. When a project is funded to build software, the funders and the community at large should be able to know if the project's plans regarding the software have been carried out.

Ideally an SMP should be drafted at the beginning of a research project. However, even for existing projects, it is valuable to create an SMP as it helps to summarise established practices and stimulate reflection and evaluation in software development. Drafting an SMP with multiple stakeholders in larger projects can help develop or strengthen common ways of working.

26 A Guide for Publishing, Using, and Licensing Research Software in Germany, <https://doi.org/10.5281/zenodo.4327147>



5. Core Requirements for Developing a Software Management Plan

Depending on their specific context (e.g. institutional policies and regulations), the creators of SMP templates may choose which of these requirements will help them to adequately manage the research software that they are responsible for. Different levels of management will need different sets of requirements (see Section 6 for further details).

5.1. List of core requirements

This section lists the requirements that an SMP should include. These requirements cover different aspects that research software needs in order to fulfil its purpose.

The requirements for an SMP are:

- **Purpose** - clearly state the purpose of the software. Provide general information such as: what problem does it solve, who is the intended audience, what are its advantages and limitations, etc. A clear explanation of the purpose of the software helps the developer focus on its specific needs.
- **Version control** - use a version control system. Adequate versioning of research software facilitates management of research software, allowing for the identification of specific versions of the software.
- **Repository** - deposit releases of your software in an appropriate repository. This should preferably be a publicly accessible repository, providing globally unique, persistent, and resolvable identifiers to each release.²⁷ The most important consideration is that potential users of the software are able to get a copy they can use.²⁸
- **User documentation** - explain clearly what the software does and how it should be used.
- **Software licensing and compatibility** - assign a licence specifying conditions of use for your software, including patenting information (if relevant). Preferably the licence should be as open as possible, and as closed as necessary. Software licences must be compatible with the licence of external components (dependencies, libraries, etc.) that the software uses.
- **Deployment documentation** - explain system requirements (e.g. dependencies) for deploying the software and instructions for installation and testing.
- **Citation** - include relevant information indicating how your software should be cited.
- **Developer documentation** - explain how the software can be modified (docstrings, in-line comments, etc.), tested, and contributed to (governance, code of conduct, contributing guidelines, etc.).

²⁷ A Persistent Identifier (PID) policy for the European Open Science Cloud (EOSC) <https://doi.org/10.2777/926037>

²⁸ This satisfies the F1.1, F1.2 and A1.1, A1.2 FAIR principles for Research software

- **Testing** - incorporate tests to ensure your software continues to work as intended. Different types of testing (unit, functional, integration, linting, typing, regression, etc.) could be used. Tests in turn should also be documented. Coverage tools should also be used to assess the extent of the tested code.
- **Software Engineering quality** - make sure your software adheres to relevant code quality standards (styling, modularity, etc.) and uses tools for collaborative development to measure code quality.
- **Packaging** - use appropriate package managers to allow users to install/deploy your software with ease.
- **Maintenance** - make sure there are arrangements in place for the maintenance and reuse of your software. This could be through a community of developers who will continue to maintain it, or by including the maintenance of software as part of future projects, or by increasing the user base. Whenever suitable, develop a retirement strategy for your software.
- **Support (during the project)** – plan resources for support-related activities such as training, hiring research software engineers, infrastructure, hardware, etc. The level of support should be in line with promises made regarding the level of service provided by your software (e.g. service level agreements).
- **Risk analysis** - consider other factors that could have an impact on your software. For example compliance with privacy policies, security considerations, reliability requirements, portability / vendor lock, etc.

These requirements are not presented in any particular order. There are many ways of ordering, grouping, and prioritising them; that is a task left to the creators of SMP templates. As an example, Figure 1 shows how these requirements could be grouped by their focus.

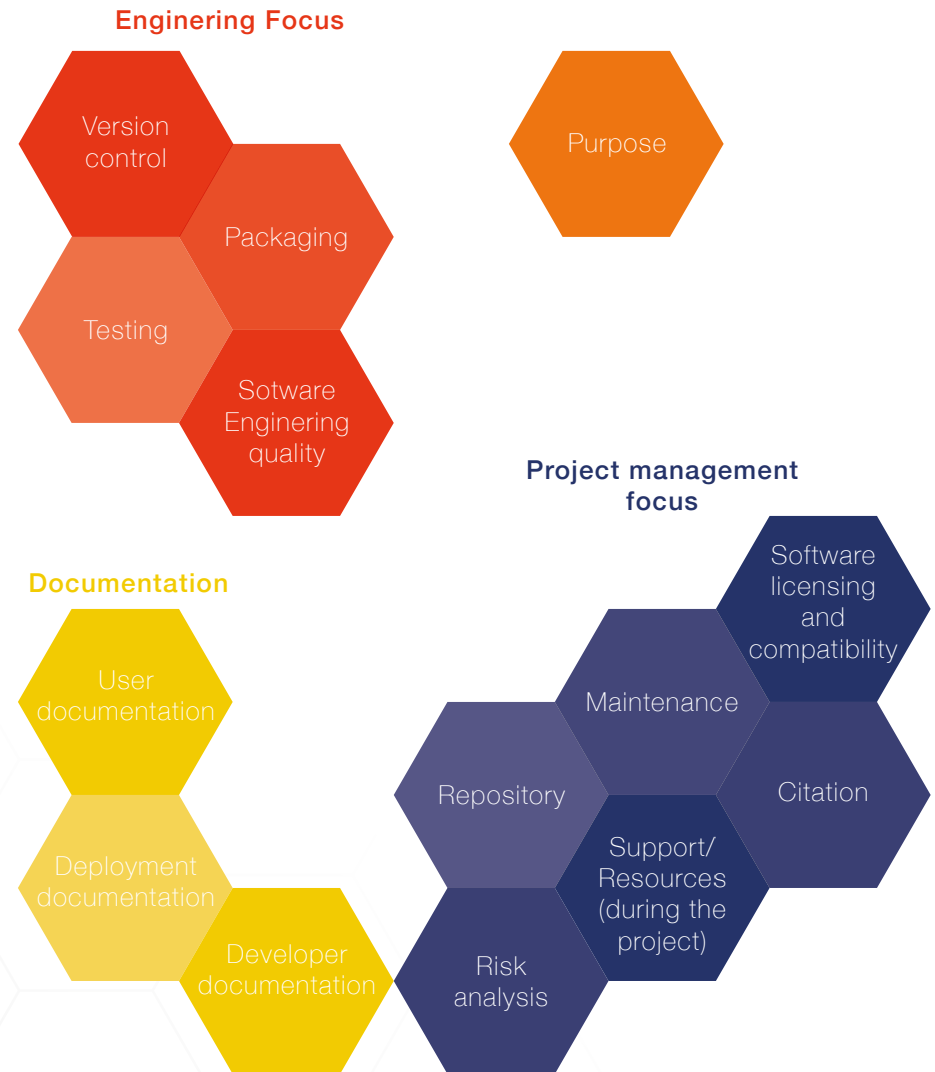


Figure 1. Software Management Plan requirements grouped by their focus.

5.2. Guidance

Detailed guidance on how to fulfil these requirements may depend on the specific needs of each research domain or institution. For example, each institution may have a specific licensing policy. The following table provides some useful resources, but it is only meant as a starting point and it should be adapted to the specific needs of each SMP.

| Requirement | Reference resources |
|---------------------------|---|
| Version control | The Turing Way guide on Version Control |
| Repository and registries | List of software registries , Software Heritage |
| Licensing | Free Software Foundation , choosealicense.com and The Turing Way guides on Software licences and Licence Compatibility |
| Citation | The Turing Way guide on Software citation |
| User documentation | The eScience Center's guide on writing documentation , How to Respond to Code of Conduct Reports and Code of Conduct Facilitators |
| Deployment documentation | |
| Developer documentation | |
| Testing | The Turing Way guide on Code testing |
| Packaging | The Turing Way guides on packaging systems (language specific guidance such as Python Packages and R Packages) and containers |

Table 2. Reference resources for different SMP requirements.

Other resources:

[Cookiecutter Data Science](#), [Productivity and Sustainability Improvement Planning](#), [Five Recommendations for FAIR Software](#).

5.3. SMP rubric

In addition to the SMP template, it is recommended to produce an “SMP assessment rubric.” Here you give examples of acceptable responses and unacceptable responses, per question. Rubrics provide guidance and an opportunity to reach out for support and/or learn about the subject. It takes effort to make a rubric (because it must match a template 1:1), but they are very helpful as guidance for researchers and for SMP evaluators. Examples of rubrics (for data management plans) can be found in the Science Europe Guide²⁹ and NWO DMP assessment rubric.³⁰



²⁹ <https://www.scienceurope.org/our-resources/practical-guide-to-the-international-alignment-of-research-data-management/>

³⁰ https://zenodo.org/record/3629157#_Yw5o8HZBwuU

6. Implementation Examples

6.1. Using subsets of the core requirements to define customised SMP templates

In Section 5 we defined a set of core requirements that are important in the software development process and that can, in principle, be included in any SMP. However, software exists in many forms - from single purpose scripts to mission critical frameworks - which means that not all requirements are necessarily applicable to every category of software. In practice, it might be useful to define SMP templates based on subsets of the core SMP requirements (Section 5.1). In the following sections we illustrate how to create such SMP templates using software management levels.

A software management level consists of a set of the core requirements that should be considered when developing a certain type of software. These requirements can be applicable before, during, and after the formal software development (project) period. Software management levels provide a recipe for grouping the core requirements into subsets and generating an appropriate SMP template. To determine which set of core requirements are relevant to a software management level, three important factors should be considered:

1. **Purpose.** What is the current reason or expected end-use for developing the software?
2. **Reliability.** The effect of software failure and/or non-maintenance on:
 - Risk of harm to self or others. This includes injury, privacy violation, bias, and inappropriate content.
 - Reputation. For example to self, institution or other.
 - Research, either your own or of others. This effect could be due to an obvious software failure (“crash”) or a hidden one, for example, returning inconsistent numerical results on different operating systems.
3. **Maintenance.** The long-term effort needed to maintain the software as long as it might be used as a standalone tool or dependency. This includes maintenance functions that can extend beyond the lifespan of the original development project and includes fixing bugs, dependency management, operating system compatibility, and security issues.

Using these factors we define three typical management levels (low, medium, high) that underlie the software examples (Section 6.2) and example SMP templates (Section 6.3).

It should be noted that, in practice, each organisation is responsible for defining its own management levels,³¹ and as a software project evolves, so can the management level that applies to it. For a low management level, organisations could also decide to include the SMP as part of the DMP template.

6.1.1. Management level: low

Purpose. This software is typically developed for a specific analysis (e.g. drawing a graph) or one-off project (e.g. practical examples in a course). The developer is the primary user and it is not intended to be used beyond a defined period or in a different context.

Reliability. This software is generally smaller in terms of lines of code and due to its restricted scope the output can easily be judged to be correct, either visually (the graph looks correct) or basic input/output testing (it gives an expected output for a defined input). Good software practices (e.g. version control and user documentation) are highly recommended.

Maintenance. As this software is not intended to be used by others, either directly or as a dependency, its influence beyond the scope for which it was intended is likely small. While measures to enable its reuse (documentation, versioning, archiving) are appropriate, no additional maintenance planning is required.

³¹ Examples of alternative software classifications include the German Aerospace Center's guidelines <https://doi.org/10.5281/zenodo.1344611> and Konrad Hinsén's scientific software stack <https://hal.archives-ouvertes.fr/hal-02117588>.

| Core requirement (Section 5.1) | Example SMP question(s) (Section 6.1) |
|--------------------------------------|---|
| Purpose | Please provide a brief description of your software, stating its purpose and intended audience. |
| Version control | How will you manage versioning of your software? |
| User documentation | How will your software be documented for users? Please provide a link to the documentation if available. |
| Deployment documentation | How will you document the installation requirements of your software? Please provide a link to the installation documentation if available. |
| Software licencing and compatibility | What licence will you give your software? Does your software respect the licences of libraries and dependencies it uses? |

Table 1. Core requirements of an SMP and examples of associated questions for a low level of software management.

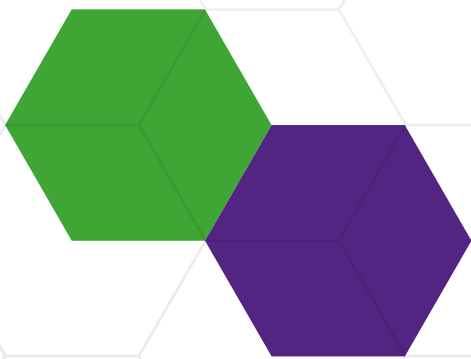


6.1.2. Management level: medium

Purpose. Software of this level is typically developed as part of a research project or is the primary output of a research project. Although usually developed for a single purpose, it incorporates functionality that may be of use to others, either as a standalone tool, library, or module in an existing tool.

Reliability. This software may have a direct influence on other researchers (e.g. project, research group) and/or software even if this was not the primary intention when it was conceptualised. As the software is more complicated and/or larger, in terms of lines of code, than those in the lower management level, good software practices such as version control using a system such as Git and user/technical documentation is essential here. More advanced requirements, such as code auditing, automated testing of major functionality, software packaging, and distribution also need to be considered.

Maintenance. This software's functionality is useful to researchers both in and outside the project, making it suitable for distribution. It will have a lifespan longer than the project in which it was developed and therefore long-term sustainability becomes more important. Software management requirements for this level include providing information on software archiving and citation as well as strategies for post-project maintenance and support.



| Core requirement (Section 5.1) | Example SMP question(s) (Section 6.1) |
|--------------------------------------|--|
| Purpose | Please provide a brief description of your software, stating its purpose and intended audience. |
| Version control | How will you manage versioning of your software? |
| Repository | How will you make your software publicly available? If you do not plan to make it publicly available you should provide a justification. |
| User documentation | How will your software be documented for users? Please provide a link to the documentation if available. How will you document your software's contribution guidelines and governance structure? |
| Software licencing and compatibility | What licence will you give your software? How will you check that it respects the licences of libraries and dependencies it uses? |
| Deployment documentation | How will the installation requirements of your software be documented? Please provide a link to the installation documentation if available. |
| Citation | How will users of your software be able to cite your software? Please provide a link to your software citation file (CFF) if available. |
| Developer documentation | How will your software be documented for future developers? |
| Testing | How will your software be tested? Please provide a link to the (automated) testing results. |
| Software Engineering quality | Do you follow specific software quality guidelines? If yes, which ones? |
| Packaging | How will your software be packaged and distributed? Please provide a link to available packaging information (e.g. entry in a packaging registry, if available). |
| Maintenance | How do you plan to procure long term maintenance of your software? |

Table 2. Core requirements of an SMP and examples of associated questions for a medium level of software management.

6.1.3. Management level: high

Purpose. There are various types of software that require a high level of management, for example software developed and distributed for users other than the developers or software that has a direct (or systematic) impact on something it interacts with. For instance, research results could be directly affected by the functioning of simulation software or training of machine learning models while physical effects could occur from the use of medical or engineering control software.

Reliability. As software of this level was designed, or has evolved, to be “mission critical,” reliability is of utmost importance. All possible actions should be taken to ensure reliability, which includes software architecture design, code standards, the use of comprehensive cross-platform automated unit and functional testing frameworks, dependency management, and code auditing. In addition, legal development requirements, such as traceability, right to use, right to inspect, right to distribute, etc., and process documentation should be implemented as required (for example, software medical devices may require ISO or EC certified management processes).

Maintenance. There is no defined maintenance period associated with this class of software as it must be maintained as long as it is in use. In order to maintain a high level of reliability, maintenance strategies, including funding and/or community development plans, should be in place. Build and release pipelines should be implemented so that not only source code availability but compiled software is maintained on evolving software/hardware platforms, OS, CPU, GPU etc.

| Core requirement (Section 5.1) | Example SMP question(s) (Section 6.1) |
|--------------------------------------|--|
| Purpose | Please provide a brief description of your software, stating its purpose and intended audience. |
| Version control | How will you manage versioning of your software? |
| Repository | How will you make your software publicly available? If you do not plan to make it publicly available, you should provide a justification. |
| User documentation | How will your software be documented for users? Please provide a link to the documentation if available. How will you document your software’s contribution guidelines and governance structure? |
| Software licencing and compatibility | What type of licence will your software have? How will you check that it respects the licences of libraries and dependencies it uses? |
| Deployment documentation | How will the installation requirements of your software be documented? Please provide a link to the installation documentation if available. This documentation should include a complete and unambiguous description of dependencies to other software, datasets, and hardware. |
| Citation | How will users of your software be able to cite your software? Please provide a link to your software citation file (CFF) if available. |
| Developer documentation | How will your software be documented for future developers? |
| Testing | How will your software be tested? Please provide a link to automated testing results |
| Software Engineering quality | Do you use a software quality checklist, such as https://bestpractices.coreinfrastructure.org/en or equivalent? |
| Packaging | How will your software be packaged and distributed? Please provide a link to available packaging information (e.g. entry in a packaging registry, if available). |

| | |
|---------------|--|
| Maintenance | What level of support will be provided for users of the software and how will this support be organised? |
| Support | How do you plan to procure long term maintenance of your software? |
| Risk analysis | Describe the main external factors that should be considered by developers and users of the software. These could include data privacy, information security, etc. |

Table 3. Core requirements of an SMP and examples of associated questions for a high level software management.

6.1.4. Summary of SMP templates developed for three management levels

| Core requirement (Section 5.1) | Software management level (Section 6.1) | | |
|--------------------------------------|--|-------------------------------------|-----------------------------------|
| | Management level: Low (6.1.1) | Management level: Medium (6.1.2) | Management level: High (6.1.3) |
| Purpose | X | X | X |
| Version control | X | X | X |
| Repository | | X | X |
| User documentation | | X | X |
| Software licencing and compatibility | | X | X |
| Deployment documentation | | X | X |
| Citation | | X | X |
| Developer documentation | | X | X |
| Testing | | X | X |
| Software Engineering quality | | X | X |
| Packaging | | X | X |
| Maintenance | | X | X |
| Support | | | X |
| Risk analysis | | | X |

Table 4. Core requirements of an SMP for software grouped by management level.

6.2. Types of software that require different levels of management

This section gives examples of software types and groups them according to the software management levels as defined in Section 6.1.

6.2.1. Software that requires low level management

- A script that is used to create and format a single figure for a publication, for example, when using a plotting package such as *ggplot2 (R)* or *Matplotlib (Python)*.
- Software written as part of a project to automate an administrative or routine process, e.g. monitoring a process or generating document templates.
- Software written specifically for the analysis of a single experiment, data processing, and presentation of its results.³²

6.2.2. Software that requires medium level management

- Software that implements a new or higher performance algorithm and can be applied to different input data.
- Simulation software that implements one or more models and/or numerical methods, e.g. computational fluid dynamics, chemical interactions, planetary evolution, partial differential equation solvers, numerical integration, etc.

6.2.3. Software that requires high level management software

- Software used in production on which an institute, department, or instrument depends on for their operation, e.g. software that is used for pre-processing all data coming from a particular telescope.
- Software that cannot be rewritten during a project's lifetime. If one requires functionality from high-impact software, replacing it may threaten a project.

³² If this is a pipeline usable by others for different experiments it likely requires medium level management.

6.3. SMP templates generated for three software management levels

This section contains examples of SMP templates that match the software management levels defined in Section 6.1. These templates should be adjusted to match the specific needs of an organisation using this guide.

6.3.1. Sample SMP template for Management level: low

This SMP template is for software with low management level.

1. Please provide a brief description of your software, stating its purpose and intended audience.
2. How will you manage versioning of your software?
3. How will your software be documented for users? Please provide a link to the documentation if available.
4. How will you document the installation requirements of your software? Please provide a link to the installation documentation if available.
5. What type of licence will your software have?
6. Does your software respect the licences of libraries and dependencies it uses?

6.3.2. Sample SMP template for Management level: medium

This SMP template is for software with medium management level.

1. Please provide a brief description of your software, stating its purpose and intended audience.
2. How will you manage versioning of your software?
3. How will you make your software publicly available? If you do not plan to make it publicly available you should provide a justification.

4. How will your software be documented for users? Please provide a link to the documentation if available.
5. How will you document your software's contribution guidelines and governance structure?
6. What licence will your software have?
7. How will the installation requirements of your software be documented? Please provide a link to the installation documentation if available.
8. How will users of your software be able to cite your software? Please provide a link to your software citation file (CFF) if available.
9. How will your software be documented for future developers?
10. How will your software be tested? Please provide a link to the automated testing results.
11. How will you check that your software respects the licences of libraries and dependencies it uses?
12. How will your software be packaged and distributed? Please provide a link to available packaging information (e.g. entry in a packaging registry, if available).
13. How do you plan to procure long term maintenance of your software?

6.3.3. Sample SMP template for Management level: high

This SMP template is for software with a high management level.

1. Please provide a brief description of your software, stating its purpose and intended audience.
2. How will you manage versioning of your software?
3. How will you make your software publicly available? If you do not

plan to make it publicly available you should provide a justification.

4. How will your software be documented for users? Please provide a link to the documentation if available.
5. How will contribution guidelines and governance structure of your software be documented?
6. What licence will your software have? Please provide a valid SPDX-Licence-Identifier.
7. How will the installation requirements of your software be documented? Please provide a link to the installation documentation if available.
8. How will users of your software be able to cite your software? Please provide a link to your software citation file (CFF) if available.
9. How will your software be documented for future developers?
10. How will your software be tested? Please provide a link to automated testing results.
11. How will you check that it respects the licences of libraries and dependencies it uses?
12. How will your software be packaged and distributed? Please provide a link to available packaging information (e.g. entry in a packaging registry, if available).
13. What level of support will be provided for users of the software and how will this support be organised?
14. How do you plan to procure long term maintenance of your software?
15. Describe the main external factors that should be considered by developers and users of the software. These could include data privacy, information security, etc.

Acknowledgements

This guide is the result of a joint initiative by the Netherlands eScience Center and the Dutch Research Council (NWO) to develop (national) guidelines for software management plans (SMPs). Over the course of 2022, a working group composed of experts in research software, representing different research organisations in the Netherlands, developed the guidelines. An international sounding board, representing national and international stakeholders in research software, provided input to the working group throughout the process of creating the guidelines. The working group also received input from the wider research community via a workshop and an open consultation round.

Working Group members

- Carlos Martinez-Ortiz, Netherlands eScience Center (<https://orcid.org/0000-0001-5565-7577>)
- Paula Martinez Lavanchy, TU Delft (<https://orcid.org/0000-0003-1448-0917>)
- James Meakin, RadboudUMC
- Brett G. Olivier, Vrije Universiteit Amsterdam (<https://orcid.org/0000-0002-5293-5321>)
- Laurents Sesink, Centre for Digital Scholarship, Leiden University (<https://orcid.org/0000-0001-7880-5413>)

Working Group coordinators

- Maaike de Jong, Netherlands eScience Center (<https://orcid.org/0000-0003-4803-7411>)
- Maria Cruz, NWO (<https://orcid.org/0000-0001-9111-182X>)

The authors would also like to thank the members of the Sounding Board:

- Anton Akhmerov, TU Delft (<https://orcid.org/0000-0001-8031-1340>)
- Zoé Ancion, Agence Nationale de la Recherche (<https://orcid.org/0000-0002-6554-8179>)
- Jonathan de Bruin, Utrecht University (<https://orcid.org/0000-0002-4297-0502>)
- Antica Culina, Ruder Boskovic Institute (IRB) and NIOO-KNAW (<https://orcid.org/0000-0003-2910-8085>)
- Christopher Erdmann, American Geophysical Union (AGU) (<https://orcid.org/0000-0003-2554-180X>)
- Marjan Grootveld, DANS (<https://orcid.org/0000-0002-2789-322X>)
- Fotis E. Psomopoulos, Institute of Applied Biosciences (INAB), Center for Research and Technology Hellas (CERTH) (<https://orcid.org/0000-0002-0222-4273>)
- Vera Sarkol, CWI (<https://orcid.org/0000-0002-8950-3178>)
- Caspar Martijn van Leeuwen, SURF (<https://orcid.org/0000-0003-4407-6675>)
- Jurgen J. Vinju, CWI and TU Eindhoven (<https://orcid.org/0000-0002-2686-7409>)

And everyone who provided input during the consultation rounds: Ablikim Abudukerim, Renato Alves, Heather Andrews, Jisk Attema, Sandrine Auzoux, Marianna Avetisyan, Celine Barthelemy, Burcu Beygu, Celine Blitz Frayret, Loek Brinkman, Hugo Buddelmeijer, Jael Castro, Pascal de Boer, Gerco de Jager, Rick de Klerk, Jelle de Plaa, Martine de Vos, Fares Dhane, Andrea Frielink-Loing, Manuel Garcia, Emilio Garcia, Olga Giraldo, Pieter Willem Groen, David Groep, Patricia Herterich, Maarten Hijzelendoorn, Tom Honeyman, Rob Hooft, Dorien Huijser, Matus Kalas, Daniel Katz, Adam Kewley, Maurits Kok, Jacko Koster, Arina Kudriavtseva, Frank Loeffler, Pablo Lopez-Tarifa, Bora Lushaj, Jason Maassen, Sjoerd Manger, Mattia Mazzucchelli, Margriet Miedema, Jurgen Moers, Neha Moopen, Elisa Yumi Nakagawa, Raymond Oonk, Robert Oostenveld, Esther Plomp, Reinder Radersma, David Rogers, Jacob Rousseau, Dan Ruddman, Vera Sarkol, Douwe Schulte, Anita Schürch, Hugh Shanahan, Russell Shipman, Sandor Spruit, Alexander Struck, Jan van den Brand, Richard van Hees, Vera van Noort, Petra van Overveld, Peter Verhaar and Qian Zhang

[DOI: 10.5281/zenodo.7038280](https://doi.org/10.5281/zenodo.7038280)

This work is licensed Creative Commons Attribution 4.0 International.



netherlands
eScience center