ELIXIR Software Management Plan for Life Sciences

Renato Alves¹, Dimitrios Bampalakis², Leyla Jael Castro³, José M. Fernández⁴, Jennifer Harrow⁵, Mateusz Kuzak⁶, Eva Martín del Pico⁴, Fotis Psomopoulos⁷, and Allegra Via⁸

1 European Molecular Biology Laboratory, Meyerhofstraße 1, 69117 Heidelberg, Germany, ELIXIR-DE/de.NBI **2** National Bioinformatics Infrastructure Sweden, ELIXIR-SE, Husargatan, 75237 Uppsala, Sweden **3** ZB MED Information Centre for Life Sciences, Cologne, Germany **4** Barcelona Supercomputing Center, ELIXIR-ES, Plaça Eusebi Güell, 1-3, Barcelona, 08034, Spain **5** ELIXIR-Hub, South Building Wellcome Genome Campus, Hinxton, Cambridge, CB10 1SD, UK **6** The Netherlands eScience Center, Amsterdam, NL **7** Institute of Applied Biosciences, Centre for Research and Technology Hellas, 6th km Charilaou-Thermis rd, Thermi, Thessaloniki, 57001, Greece **8** IBPM-CNR, c/o Department of Biochemical Sciences "A. Rossi Fanelli", Sapienza University of Rome, P.le Aldo Moro 5, 00185 Rome, Italy

Abstract

Data Management Plans are now considered a key element of Open Science. They describe the data management life cycle for the data to be collected, processed and/or generated within the lifetime of a particular project or activity. A Software Management Plan (SMP) plays the same role but for software. Beyond its management perspective, the main advantage of an SMP is that it both provides clear context to the software that is being developed and raises awareness. Although there are a few SMPs already available, most of them require significant technical knowledge to be effectively used. ELIXIR has developed a low-barrier SMP, specifically tailored for life science researchers, aligned to the FAIR Research Software principles. Starting from the Four Recommendations for Open Source Software, the ELIXIR SMP was iteratively refined by surveying the practices of the community and incorporating the received feedback. Currently available as a survey, future plans of the ELIXIR SMP include a human- and machine-readable version, that can be automatically queried and connected to relevant tools and metrics within the ELIXIR Tools ecosystem and beyond.

Introduction

Data Management Plans (DMPs) are a cornerstone of good data management and are now considered a key element of Open Science practices. A DMP describes the data management life cycle for the data to be collected, processed and/or generated within the lifetime of a particular project or activity. Conversely, a Software Management Plan (SMP) can help formalise a set of structures and goals that ensure the software is accessible and reusable in the short, medium and long term. Although it has a management perspective, the main advantage of an SMP is that it provides clear context to the software that is being developed. In that sense, it addresses several aspects of the software development process such as (a) supporting reproducibility and reusability of the software, (b) allowing funding agencies to have a better grasp of the envisioned development process (as well as the achieved milestones), (c) increasing the awareness of the existing community standards that can/should be used, and (d) ensuring that the software can be easily accessed by the wider community.

There are a few flavours of SMPs already available in one form or another. The Software Sustainability Institute (SSI) offers a very detailed checklist (Institute, 2018) that is further complemented by an online [sustainability evaluation service \(SES\)](#). Several journals (such as [SoftwareX](#) and the [Journal of Open Source Software \(JOSS\)](#)) have checklists that are expected to be filled in by the software authors before any submissions addressing most of the key points

BioHackathon series:
[BioHackathon Europe 2020](#)
Virtual conference 2020
Software Management Plans

Submitted: 20 Oct 2021

License
Authors retain copyright and
release the work under a Creative
Commons Attribution 4.0
International License ([CC-BY](#)).

Published by [BioHackrXiv.org](#)

of an SMP. Finally, there are funding agencies (such as the [Wellcome Trust](#)) that expect a plan for the management of research output, including software, in submitted applications.

A key downside of the aforementioned SMPs is that they tend to be rather complex, occasionally requiring deep technical knowledge of the software development process. In order to address these drawbacks, ELIXIR has designed a simplified version of an SMP, tailored for Life Science oriented projects but still general enough so as to be widely used. The primary goal of the ELIXIR SMP is to encourage wider adoption by Life Science researchers, and be as inclusive as possible to the various levels of technical expertise, while also having an explicit connection to the FAIR principles for Research Software (Hong et al., 2021, @lamprecht_towards_2020). A common theme in Life Science researchers is the wide differences in background expertise, with the vast majority of researchers being self-taught research software developers. Having an SMP with a relatively low barrier in technical knowledge, while maintaining all the best practices expected in research software development, may both encourage wider adoption of these practices as well as increase the awareness of the multiple aspects involved in research software development.

Software stages

We framed the definition of the ELIXIR SMP and the related stakeholders, as listed in the Methods section, by connecting them across the distinct software stages which are described below.

Stage 1: Inception (concept, proposal writing, planning and inception)

This stage starts with the idea of the research project that includes software. In this stage the proposal for the project is written, planning and first steps for the software development are made.

What is important? During this stage, the decisions are made about the license of the software, where the software will be stored during development (e.g. [GitHub](#), [GitLab](#), etc.), what kind of versioning system will be used and what kind of input and output standards the software will have.

Why is it important? The license will determine the rules for access and use of the software, while version control is used for tracking changes in the software. The ability to track each change as it was made, and to reverse changes when necessary, can make all the difference between a well managed and controlled process and an uncontrolled 'first come, first served' system. It serves as a mechanism for due diligence for software projects [<http://oss-watch.ac.uk/resources/versioncontrol>]. The versioning system and location of the codebase will determine how it can be accessed by others. Also the choice of the versioning system will narrow down available online services that can be used to share code (e.g. [BitBucket](#) only supports [Git](#)). Using an open GitHub repository will make it easier for others to reuse and contribute. On the other hand, using a local institutional repository (e.g. local GitLab instance) may be a solution preferred by the specific organisation.

The choice of the input and output formats will greatly affect the level of adoption. Choosing standards in the field (e.g. [GA4GH](#), or resources found in registries such as [FAIRSharing](#), [EDAM ontology](#) (Ison et al., 2013), etc) will make it easier for adopters to integrate the tool in their own workflows and will allow for reuse with their own datasets.

Stage 2: Construction (prototyping, construction and implementing core functionality)

This is the stage in which core functionality is developed in an iterative process. The software will most likely be developed with the use of toy example input data. During this stage,

developers will document the code while writing it (inline documentation, docstrings etc.). It is also important to ensure that the code functions as intended (e.g. via unit tests or doctests).

What is important? During this stage, the main development of the software is taking place. This is when the testing approach needs to be defined (incl. the list of the types of test implemented and relevant test parameters) - and by extension, include also sample example input and output files. Additionally, the actual scope, goals and purpose of the software needs to be defined and documented.

Why is it important? Sample input and output files are necessary for the development of functioning software, and for ensuring that it works as intended. Having the purpose (scope, and possible constraints) of the software clearly documented is necessary in focussing the development effort and avoiding feature creep. It is also crucial towards defining functional tests. Tests can also act as documentation by showing how the software can be executed and configured, but they should not be a replacement for the in-depth documentation.

Stage 3: Application (release and quality assessment)

In this stage the software can be readily applied to all relevant data, still used mostly by the development team (or the lone developer). The primary focus at this stage is the quality assessment of the software, i.e. provide a battery of tests, ideally fully automated, that ensure valid execution and behaviour. Also, care should be given to providing proper instructions to assure the software can be easily installed and set up in common scenarios.

What is important? Documented information on how the software can be reliably used by people external to the development group or person. Providing tests as extensively as possible (ideally through automated processes), in order to ensure that the documented functionality is stable, as well as identifying potential bugs and/or issues. Communication channels should also be made available for users to report any problems or pose questions about usage and documentation (e.g. issue tracker, dedicated email address), also clarifying the process to address issues/bugs (incl. the option of not being possible to address them, e.g. repo status).

Why is it important? In order to ensure adoption/use of the software outside the original developers, clearly communicated information must be provided. Having working tests (incl. having tests as part of the installation process) helps ensure that when something is fixed, it will not break other things or if it does, it will not go unnoticed. These are key aspects, essential for reproducibility and they enhance the overall trust that the software will work as intended.

Stage 4: Production (production software working on real-world data in a scalable and stable manner)

In this phase, the software is used to generate publishable research data in a consistent, stable and reproducible way. The software should be easily reusable by other researchers with their datasets. Often this is the time when the software is being deployed on a High-Performance Computing infrastructure (e.g. local cluster, supercomputer etc.) or a cloud infrastructure. To ensure the software can be easily and successfully deployed, soft and hard dependencies need to be captured (e.g. versions of libraries or packages, databases and other resources, listing required / recommended packages, etc). This is often done by bundling and packaging all necessary software through different approaches, including static binaries, software containers, virtual machines or any other solution that encapsulates the entire required environment. Specific dependencies on hardware (e.g. CPU optimisations, GPU model, etc) needed by the different execution profiles should also be documented clearly.

What is important? The most crucial aspect here is to ensure the (re-)use and reproducibility of the software, by clearly documenting any OS/Software/hardware dependencies. This can be

achieved through different methods (incl. registry and package managers, containerisation, etc.).

Why is it important? Software often depends on libraries and their versions that may not be available on some systems. Packaging the software into easy-to-use forms (e.g. [R](#) / [python](#) / [conda packages](#), pre-compiled binaries, containers or virtual machines, etc.) can greatly facilitate its (re)use and adoption, as well as ensure (to an extent) its reproducibility.

Stage 5: Publication (Publishing software and/or research results obtained with the software)

At this stage the software can be used to generate usable results for the project. In a research project, and independently of the results obtained with the use of the software, the developer(s) are encouraged to publish the software to a software-oriented journal (e.g. [JOSS](#), [Software X](#), etc), so that potential users can become aware of it and cite it when used. Additionally, software may also be deposited in archiving services (such as the Software Heritage Foundation - [SWH](#)), or general repositories (such as [Zenodo](#)), ensuring a persistent identifier (such as Digital Object Identifiers - [DOI](#)). This is also when other researchers will find software through the publication of the respective research project, and start using it for their own research or for reproducing original results.

What is important? Documenting information about the way people should cite the software (both in the README as well as including one of the standard formats for citation, e.g. [Citation File \(CFF\)](#), [BibTex](#)) and include the authors' [ORCIDs](#) (i.e. the unique identification of each author) in the citation information. Independently of how the software is published and made available, the authors should create releases with a persistent global unique identifier pointing to each released version of the software. Developers should keep an up to date CHANGELOG to communicate changes in the way the software works for the growing user base.

Why is it important? Researchers developing software and RSEs are rarely rewarded for their work. Providing citation information is essential to communicate how the authors want the work to be referenced by their peers, and lead the way to acknowledgement and recognition of their effort. Researchers need to be able to find (via the persistent identifier) a specific version of software that has been used to generate published results. This is crucial in order to facilitate reproducibility of research results. As modifications and corrections are introduced, users should have sufficient information (e.g. through the CHANGELOG file) to assess the impact of the changes on ongoing projects that rely on the software.

Methods

The starting point for the creation of the aforementioned SMP was the four recommendations for Open Source Software (Jiménez et al., 2017). These recommendations are meant to encourage best practices in research software development. In order to incorporate the feedback of the ELIXIR community and the best practices used there, interviews were conducted during a dedicated project within the Europe BioHackathon 2019. The common practices and ideas collected from the interviews were structured around the four FAIR Research Software principles (findability, accessibility, interoperability and reusability) (Wilkinson et al., 2016), based on the area they were more relevant. Additional care was given towards ensuring that the main low-effort high-gain actions were captured, in order to ensure that the resulting questions had a low knowledge barrier for the expected users. The first version of the SMP, grouping the practices as described above, was presented in a dedicated webinar organised by ELIXIR in July 2020. Following that, two iterations of revision by the ELIXIR community were conducted. Specifically, developers from the ELIXIR community were invited to review and comment on the questions and corresponding options, ultimately leading to a consensus version of the SMP that incorporated all proposed changes.

By design, the scope of the ELIXIR SMP was defined as follows: the main location where a project's software code is hosted must be associated with one, and only one, SMP, while a given SMP may be connected to zero or more projects. This definition allows for a higher versatility of the SMP, supporting both complex funded research projects, as well as individual collaborative efforts. In terms of stakeholders, the ELIXIR SMP considers two different groups: those who require access to the SMP but are not responsible for filling it (such as funders, policy makers, service providers, software managers and publishers), and those who are expected to actually fill in and maintain the SMP (such as developers, researchers and principal investigators).

In terms of structure, the SMP comprises seven areas relevant for software quality, each of which is composed of a collection of targeted questions. The main areas are Accessibility, Documentation, Testing, Interoperability, Versioning, Reproducibility and Recognition, and the questions vary in complexity and requirements; in some instances a Yes/No answer is expected (i.e. Software without a license can not be used by others. Does your software have a license?), whereas others expect a selection from a predefined list (i.e. What version control system do you use?) or even free text (i.e. How do you define language-specific dependencies of your software and their version? For instance [Maven](#) for [Java](#), requirements.txt or environment.yml for Python, package.json for [JavaScript](#)).

Results

Our results are two-fold. On one side, we have identified the stakeholders involved in SMPs. On the other hand, we have defined the questions needed for an SMP for life sciences.

SMP stakeholders

We identified two main stakeholders' groups corresponding to users and adopters. In addition, we defined a group of distinguished personas that are out of the scope of the SMP's stakeholders, specifically the (a) **software users** who might also be contributors (e.g. by submitting bugs), and (b) the **end-users** of the software, who have no involvement with the software development process.

For each group of identified stakeholders we answered two questions: (i) how the stakeholders would use the SMP and (ii) at which stage of the software development the stakeholders would need the SMP most.

Group 1: Users / Benefiter / Enforcers

This group includes stakeholders that require access to the SMP for a given software, but are not responsible for filling it in.

Funders e.g. ELIXIR, Wellcome Trust

- How would this role use the SMP?
 - Use the SMP in order to assess whether a software fits the funding requirement (e.g. be in Stage 4).
 - Evaluate if the proposal addresses essential aspects of Research Software Management (incl. "scoring" the software based on their alignment to best practices)
 - Ensure that the software is well-done (reproducible, reusable/open, robust) and funding is not wasted
- At which stages of the software development (as listed below) would this role need the SMP most?
 - Depending on the funding scheme; it's usually at the Inception stage (Stage 1), but it can also be at the Production stage (Stage 4). In all cases, this role would expect the full SMP filled in (either defining what is already in place, or the plan to do this).

Policy Makers e.g. European Commission can use SMPs as a way of “enforcing” their recommendations and policies)

- How would this role use the SMP?
 - Use the SMP as a basis, inspiration or requirements for their guidelines or policies in their domain of action
- At which stages of the software development (as listed below) would this role need the SMP most?
 - During early phases of the development of similar policies

Service providers (e.g. Compute Platform, PaaS, IaaS, SaaS use an SMP in order to ensure compatibility to the infrastructure offered)

- How would this role use the SMP?
 - A service provider would use the SMP to assess if the software meets the quality and reproducibility standards of the platform. The SMP can therefore also be used as a requirement.
 - To assess the level of maintenance effort necessary to provide access to the software on their platform
 - The different stages described in the SMP can also be used to label software according to their expected reliability (e.g. tools in experimental phase)
- At which stages of the software development (as listed below) would this role need the SMP most?
 - When adopting or integrating software into existing platforms or in early stages of the decision process.

Software manager not necessarily contributing to the software - making sure principles are enforced

- How would this role use the SMP?
 - To assess if the software is reliable for ongoing research projects or during the conceptualisation phase of new research projects.
 - To assess if the development process of software is compatible with the timeframe of research projects
- At which stages of the software development (as listed below) would this role need the SMP most?
 - While software is at its early stages of development (application or younger). The more mature a software the less likely that it will pose an obstacle to research goals.

Publishers e.g. [F1000](#) should use an SMP in order to ensure software “quality” / availability?)

- How would this role use the SMP?
 - To ensure research published can be reproduced by peers with the software that was used to generate the results.
- At which stages of the software development (as listed below) would this role need the SMP most?
 - At any stage when authors submit software to be published. e.g. checklist used by JOSS
 - A publisher may also require that software is in the “production” or “publication” stage to be considered for publication.

Group 2: Adopters

This group includes stakeholders that are primarily responsible for filling a SMP for a given software.

Developer/Researcher the person writing the software needs to be aware of the basic principles to follow

- How would this role use the SMP?

- As a guide to implement best practices in ongoing or new software development efforts
- As a quality filter when considering if existing software meets the needs for new research projects
- At which stages of the software development (as listed below) would this role need the SMP most?
 - At all stages, from inception to publication

PI of a group can use the SMP to ensure compliance with best practices)

- How would this role use the SMP?
 - To inform the proposal writing process
 - As a guide to ensure any software developed in the group meets an acceptable quality and technical level in order to maximise the chances of adoption
 - To ensure the team composition supports skills necessary to develop software according with best practices
 - To ensure software can be reused in the future, enabling building on vs reinventing
 - To increase the bus factor of the project
 - As a checklist prior to publication
- At which stages of the software development (as listed below) would this role need the SMP most?
 - At all stages, from inception to publication

Organisation (Fundee) Vested interest that in-house research output meets criteria and is sustainable - select appropriate SMP. This is the instance where an Institution is attempting to create a policy for internal use (ensure that any “badged” research software produced under it, aligns to the expectations of the selected SMP)

- How would this role use the SMP?
 - To ensure there is supporting infrastructure at the organisation level
 - To ensure there is a training program / supporting personnel, supporting skill necessary to implementing the SMP (e.g. software stewards)
- At which stages of the software development (as listed below) would this role need the SMP most?
 - At all stages

ELIXIR SMPs

Here we present the ELIXIR SMP, as a result from the BioHackathon Europe 2019 and 2020 as well as the ELIXIR All Hands 2021. This version is the result of the community-based revision process described in the [Methods section](#).

Accessibility / License

1) How can the software be accessed by third parties?	checklist Publicly available (e.g. GitHub, via URL, etc.) Not relevant/applicable to me Other Yes/No
2) Software without a license cannot be used by others. Does your software have a license?	
3) If yes, what is the license of your software?	SPDX List + Other + NA

Documentation

1) What type of documentation is available, provided with the software and delivered under the same conditions? Please provide a URL (when available).	checklist	Please provide the corresponding URL, if available
	User documentation	
	Programmers documentation	
	API documentation	
	README file	
	Release notes	
	Docstring/comments in the source code	
	CHANGELOG	
	Other (for more than one, please add multiple rows)	
	None	
2) Is the purpose of the software stated in the documentation?	Yes/No	
3) Does the documentation describe how to	test the software	Yes/No
	use the software	Yes/No
	build the software	Yes/No/NA
	deploy the software	Yes/No/NA
	install the software	Yes/No/NA

Testing

1) What type of testing do you use?				
	None	Yes		No
	Unit	Yes (Manual)	Yes (Automatic)	No
	Integration	Yes (Manual)	Yes (Automatic)	No
	Functional	Yes (Manual)	Yes (Automatic)	No
	End-to-end	Yes (Manual)	Yes (Automatic)	No
	Linting	Yes (Manual)	Yes (Automatic)	No
	Regression	Yes (Manual)	Yes (Automatic)	No
	Frontend - GUI	Yes (Manual)	Yes (Automatic)	No
	Other (e.g.)			

2) Are sample data and/or parameters that can be used to test the software available with the source code?	Yes	No	NA
--	-----	----	----

Interoperability

1) Do you use existing and standard input/output formats?	Yes	Yes (partially)	No
If yes (either all, or partial), please select the standards you are using from the list. If it's not listed, please use the FAIRsharing registry to provide both name and URL. If it's not listed in FAIRsharing, please provide a name and a URL that is as machine actionable as possible.	Name (for each standard)	URL (for each standard)	Description (optional) (for each standard)

Versioning

1) Do you use a version control system?	Yes	No
2) What version control system do you use?	Multiple selection with Other as Text Git Mercurial Subversion CVS Other	
3) Do you use Semantic Versioning (for more information go here: https://semver.org/)?	Yes	No

Reproducibility

1) Do you provide releases of your software?	Yes	No
2) How do you define language-specific dependencies of your software and their version? For instance Maven (for Java), requirements.txt or environment.yml (Python), package.json (JS)	Text	NA
3) How do you capture the environment (operating system, system-level libraries) necessary to run the software (e.g. containers, conda, virtual machine)?		
4) Do you provide input and output examples that can be used to reproduce the functioning of your software?	Yes	No
If yes, where can they be found?	URL - Please provide the URL, if available	
5) Do you state how to report bugs and/or usability problems by the software user(s)? This could be a simple sentence in the software documentation, stating an email, or an issue tracker, or a direct messaging protocol, or even a statement that bug reporting/issues are not supported.	Yes	No

Recognition

1) Do you include citation information (i.e. how to cite your software in the form of citation.cff, codemeta.json or bibtex)?	Yes	No
---	-----	----

2) Do the releases have a persistent global unique identifier (such as release on Zenodo with DOI, snapshot or/and release referenced on Software Heritage with SWHID)? For more information on PIDs and the differences between the various types (intrinsic, extrinsic, etc), please see here .	Yes	No
3) Does the citation information contain ORCIDs of (at least one of) the authors	Yes	No
4) Is the software registered in a domain-specific registry	Yes	No
If yes, please list the corresponding registries (Name and URL for each)	Name and URL	NA

Conclusions and future work

We have presented here the result of two versions of the BioHackathon Europe, 2019 and 2020, regarding SMPs for life sciences. Although our SMP is mainly intended for Research Software Engineers, we have reduced some barriers so that part-time developers and experimentalists using pieces of code for their research can also use and benefit from SMPs.

We will continue working on a machine-actionable version of our SMP so data can be gathered (semi)automatically and exchanged with repositories and registries collecting information related to research software, for instance bio.tools and GitHub. We will also work on adoption within the ELIXIR community and will hopefully reach other communities via publications, workshops and conferences.

Ultimately, using an SMP for research software yields significant benefits: on one hand, it helps ensure software reproducibility and reuse, while on the other it promotes the adoption of best practices for software development by developers with non-computer science background. The actual process of developing the SMP was an example of a fruitful and positive community effort: it was the result of continuous discussions and rigorous consultation with the community, always ensuring the feedback was carefully considered and addressed. Next steps in the ELIXIR SMP include efforts to make it easily usable by the community, engaging software developers in its adoption and aligning it to other ELIXIR tools such as [OpenEBench](#). To this end, a dedicated tool for the semi-automated construction of a human- and machine-readable SMP will be implemented and training materials for the SMP adoption and use will be created.

- Make explicit connection to FAIR in future plans
- Metrics

GitHub repository

- SMP GitHub repo: <https://github.com/elixir-europe/smp>

Acknowledgements

- This work was funded/supported by ELIXIR, the research infrastructure for life-science data and through ELIXIR BioHackathon Europe

References

- Hong, N. C., Katz, D. S., Barker, M., Lamprecht, A.-L., Martinez, C., Psomopoulos, F. E., Harrow, J., et al. (2021). *FAIR Principles for Research Software (FAIR4RS Principles) RDA* (RDA endorsed output). doi:[10.15497/RDA00065](https://doi.org/10.15497/RDA00065)
- Institute, T. S. S. (2018). *Checklist for a Software Management Plan*. Zenodo. doi:[10.5281/zenodo.1460504](https://doi.org/10.5281/zenodo.1460504)
- Ison, J., Kalas, M., Jonassen, I., Bolser, D., Uludag, M., McWilliam, H., Malone, J., et al. (2013). EDAM: An ontology of bioinformatics operations, types of data and identifiers, topics and formats. *Bioinformatics*, 29(10), 1325–1332. doi:[10.1093/bioinformatics/btt113](https://doi.org/10.1093/bioinformatics/btt113)
- Jiménez, R. C., Kuzak, M., Alhamdoosh, M., Barker, M., Batut, B., Borg, M., Capella-Gutierrez, S., et al. (2017). Four simple recommendations to encourage best practices in research software. *F1000Research*, 6, 876. doi:[10.12688/f1000research.11407.1](https://doi.org/10.12688/f1000research.11407.1)
- Lamprecht, A.-L., Garcia, L., Kuzak, M., Martinez, C., Arcila, R., Martin Del Pico, E., Dominguez Del Angel, V., et al. (2020). Towards FAIR principles for research software. *Data Science*, 3(1), 37–59. doi:[10.3233/DS-190026](https://doi.org/10.3233/DS-190026)
- Wilkinson, M. D., Dumontier, M., Aalbersberg, I. J., Appleton, G., Axton, M., Baak, A., Blomberg, N., et al. (2016). The FAIR Guiding Principles for scientific data management and stewardship. *Scientific Data*, 3(1), 160018. doi:[10.1038/sdata.2016.18](https://doi.org/10.1038/sdata.2016.18)