Alliance Software Management Plan (SMP) Template.

2024-08-06

Digital Research Alliance of Canada | Alliance de recherche numérique du Canada

Introduction

Software has become an essential part of modern research and is increasingly recognized as an important output of research. Despite years of wide adoption of Data Management Plans (DMPs) to ensure good data management practices are followed, in Canada there were currently no clear guidelines that instruct various stakeholders how to plan for a piece of software to be properly developed, maintained, shared, and reused. To address this gap, the Alliance launched the Alliance Software Management Plan (SMP) Template (EN/FR).

WHAT IS AN SMP?

A Software Management Plan (SMP) is a structured document detailing how a specific software project is developed, maintained, and curated. It ensures that the software remains accessible, (re)usable and sustainable over time, benefiting developers, maintainers, and others involved in the software lifecycle. Ideally, an SMP is drafted at the project's outset but can also enhance existing projects by summarizing established practices and fostering continuous improvement.

BENEFITS OF USING THE ALLIANCE SMP TEMPLATE

The SMP can help funders to encourage, support and promote software recognition and good practices (e.g., FAIR4RS), facilitate culture change and, support communities, ensuring consistent adherence to certain software management standards and policies. It can also support and upskill researchers to develop higher-quality software, which will likely be an important part of a researcher’s portfolio in the future.

If you are a **Funder**, a **Policy Maker**, an **Institutional Administrator**, a **Publisher**, or a **Service Provider**: you are encouraged to use this SMP template

- as a basis or inspiration for guidelines or policies;
- to assess software against funding requirements and quality standards (e.g., compute platform, PaaS, IaaS, SaaS);
- to ensure organizational support (e.g., support infrastructure, training program, support personnel such as software stewards/professionals) for SMP implementation;
- to ensure the software is properly managed as described in the SMP; or
- to ensure published research can be reproduced with the software (together with associated data and documentation) that was used to generate the results.

If you are a **Software Developer, Software User, Researcher,** a **Software Manager,** or a **PI of a Group:** you are encouraged to use this SMP template

- to inform the proposal writing process;
- as a guide to adopt best practices in ongoing or new software development efforts;
- to assess software quality and technical level in order to maximize the chances of adoption;
- to assess software compatibility with the overall project scope, deliverable, and timelines;
- as a checklist prior to publication; or
- to evaluate software reliability (e.g., interoperability, flexibility, extensibility vs. reinventing) for ongoing or new research projects.

ALLIANCE SOFTWARE MANAGEMENT PLAN (SMP) TEMPLATE

NOTE:

This is the Alliance Software Management Plan (SMP) Template (version 1) consisting of 18 questions, including 13 mandatory questions (marked with *) and 5 optional questions.

Glossary

Software: Software in this SMP template refers to two categories: 1). software tools (e.g., software programs, languages, libraries, scripts, computational code, models, electronic lab notebooks, repository software, workflow management tools, etc.); and 2). software platforms (variously referred to as research infrastructures, virtual science labs, virtual research environments (VREs), or Science Gateways, etc.).

Software Versioning: In software development, software versioning is a type of numbering or naming scheme that helps software development teams keep track of changes they make to software project code by identifying successive versions. The changes may include new functions, features, or bug fixes. Occasionally, entirely new functions and features are released based on developments across multiple versions. As such, it assists in the creation and management of multiple software product releases. Modern computer software is often tracked using two different software versioning schemes: 1). an *internal* version number that may be incremented many times in a single day, such as a revision control number, and 2). a *release* version that typically changes far less often, such as semantic versioning or a project code name.

Version Control: Version control (also known as source control, revision control) is the practice of tracking and managing changes to software code over time. Version control is also a way to ensure efficient and collaborative code sharing and editing among multiple developers on different versions of the software at any given time within the larger system. Version control systems (VCS), sometimes known as SCM (Source Code Management) tools or RCS (Revision Control System), are software tools that help software teams manage changes to source code over time.

Software Release Life Cycle (SRLC): The Software release life cycle (SRLC) is a set of milestones that describe various stages in a piece of software's sequential release timeline, from its conception to its public distribution. It typically consists of several stages, such as pre-alpha (referring to the early stage of development where the software is still in its design and development phase), alpha (which represents the first formal testing phase using internal resources), beta (during which it's tested by a larger group of users outside of the organization to find and fix potential bugs or issues), release candidate (for further refinement and testing), and production release (also called stable release, marking the stable and complete version of the product ready for use by end-users). Once released, depending on the method of release, there are stages such as, release to manufacturing (RTM, also known as "going gold", when a software product is ready to be delivered), general availability (GA, is a marketing stage when it becomes available for purchase), and release to the web (RTW, or web release, is a means of software delivery utilizing the internet for distribution).

Purpose

*** Provide a brief description of your software, stating its purpose, intended audience, and the problem it solves.**

[Type answer here]

Example Answer:

"3D Slicer is an open-source platform for the analysis and display of information derived from medical imaging and similar data sets. Such advanced software environments are in daily use by researchers and clinicians and in many nonmedical applications. 3D Slicer is unique through serving clinical users, multidisciplinary clinical research terms, and software architects within a single technology structure and user community. Functions such as interactive visualization, image registration, and model-based analysis are now being complemented by more advanced capabilities, most notably in neurological imaging and intervention. These functions, originally limited to offline use by technical factors, are integral to large scale, rapidly developing research studies, and they are being increasingly integrated into the management and delivery of care. This activity has been led by a community of basic, applied, and clinical scientists and engineers, from both academic and commercial perspectives." example source: https://research-software-directory.org/software/3d-slicer

Guidance:

Consider the advantages and limitations of the software. What is the current reason or expected end-use for developing the software?

Documentation and Metadata

Please describe the architecture of the software (platforms). Are there some high-level principles that will be or were used as an inspiration for the design of the software (platforms)?

[Type answer here]

Example Answer:

http://ireceptor.irmacs.sfu.ca/architecture

Guidance:

You can also insert a system architecture diagram outlining the hardware and software components of the proposed project, clearly differentiating between parts of the system that already exist and those parts that will have to be added/modified as part of this project, and/or show how parts would interact with users and other resources, as appropriate. The C4 model is an example of an approach for visualizing software architecture. Examples of High-level principles for design include FAIR for research software (FAIR4RS) principles, DevOps/MLOps, RESTful API, etc.

*** Please describe how your documentation supports users and developers of your software (platforms). Provide links to existing documentation if available, including any documentation best practices you follow.**

[Type answer here]

Example Answer:

Documentation for compiling/building/running the source code will be created and made available (e.g., https://slicer.readthedocs.io/en/latest/developer_guide/build_instructions/index.html). Documentation for the end-users, including tutorials will also be provided and made accessible (e.g., https://discourse.slicer.org , https://www.slicer.org).

Example source: https://github.com/Slicer/Slicer?tab=readme-ov-file#readme

Guidance:

For example, 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); how will you document the installation requirements of your software? Please list the software dependencies and components in your Software/Research Software Platforms. Please provide a link to the installation documentation if available and comment on any additional costs to users such as license fees, compute and storage resources or the need to purchase extra equipment.

*** How will you make sure that documentation is created or captured consistently throughout your project?**

[Type answer here]

Example Answer:

Dedicated documentation (for users/developers) will be created, shared and updated with tracked version control along with the source code. In case of multiple documents, a centralized summary document will also be created to summarize, organize, and track all other project documentation. This ensures easy access, version control, and visibility for all team members.

Guidance:

Providing insights into the research software development steps, methodology, and any relevant information necessary helps to understand the development process for reproducibility purposes; it also helps to ensure consistent documentation creation or capture throughout the research software development project, documentation.

Testing and Deployment Strategy

Please describe or outline how you intend to test and deploy your software (platform) to ensure it: (1) meets the requirements that guided its design and development, (2) is usable and performs its intended function/s, and (3) can be installed and run in its intended environment.

[Type answer here]

Example Answer:

https://github.com/Slicer/Slicer/tree/main/Testing testing documentation on Windows

(https://slicer.readthedocs.io/en/latest/developer_guide/build_instructions/windows.html#test-slicer), MacOS (https://slicer.readthedocs.io/en/latest/developer_guide/build_instructions/macos.html#test-slicer), and Linux (https://slicer.readthedocs.io/en/latest/developer_guide/build_instructions/linux.html#test-slicer)

Guidance:

Examples of deployment details include dev/production environment, dependencies on vendors-specific features, e.g. OpenShift vs vanilla kubernetes. Testing strategy includes details on unit, functional, integration, regression tests, and coverage tools, etc.

Version Control & Release Management

*** How do you plan to manage versioning for your software? What are specific functionalities provided by the version control system/platform?**

[Type text here]

Example Answer:

An (internal) (re)version number will be used throughout the version control management. We will use Git for version control. Git provides distributed version controls paired towards tracking and managing changes to software code *following the Gitflow workflow with feature, release, and hotfix branches. Release tags (e.g., https://github.com/aces/cbrain/tags) will be used to mark specific versions. The main branch (master) will be protected, requiring code reviews and passing automated tests before merging.*

A release version, e.g., Semantic versioning (SemVer) will be used upon versioning the product/major release of the software.

Guidance:

Modern software development is often tracked using two different software versioning schemes: 1). an internal version number that may be incremented many times in a single day, such as a revision control number, and 2). a release version that typically changes less often, such as semantic versioning or a project code name.To manage internal versioning for your software, utilize an issue tracking and/or version control system/platforms such as Git,, Mercurial, and Apache Subversion (SVN), Beanstalk, BitBucket, and GitHub.

What strategies/high-level principles/mechanisms/practices/tools do you plan to utilize for your software release life cycle (SRLC)?

[Type text here]

Example Answer:

During the pre-alpha phase, our source code will be hosted on GitHub for developer team collaboration.GitHub also offers Continuous integration/Continuous delivery (CI/CD) to automate build, test, and deployment. Major releases will be scheduled quarterly, with minor updates as needed. GitHub will also be used to manage the release cycle, including a staging environment for pre-release testing (alpha/beta/release candidate) and stability (production release). The source code repository on GitHub will be private at the pre-alpha phase, and then made public from the beta stage, with the possibility of transitioning to a public or institutionally-hosted GitLab instance later at the production release phase .

Guidance:

Utilize tools such as GitHub, GitLab, BitBucket, and Jenkins to automate the software release process. Define a clear software release life cycle with scheduled major updates and frequent minor updates. Ensure thorough testing in staging environments and maintain detailed release documentation to facilitate deployments. Continuous integration/Continuous delivery (CI/CD) is a powerful strategy that allows a solid quality assurance as it automates the software development process from conception through deployment. Automated tests play a crucial role in CD as it ensures that new code changes meet the established standards.

Sharing and Reuse

*** Describe how you will make your software publicly available during and after development/upgrading, including through repositories like GitHub, Zenodo, or Figshare. Provide a rationale if open access is not feasible.**

[Type text here]

Example Answer:

During the development phase, the source code will be created, stored, managed and shared on GitHub with version control. At the end of the project, the Software/Software Platform will be shared and published on Zenodo or Figshare and be minted a DOI for open access.

Guidance:

In order to make your software publicly available, you need to deposit your software (either under development or product code) in an appropriate repository. This should preferably be a publicly accessible repository, providing globally unique, persistent, and resolvable identifiers to each release.

*** Specify the type of license for your software, preferably an open license. If not open, provide a justification. Consider hybrid licensing for software with multiple components.**

[Type answer here]

Example Answer:

LGPL-3.0 license (https://github.com/aces/cbrain#GPL-3.0-1-ov-file)

BSD Licenses (https://github.com/Slicer/Slicer?tab=License-1-ov-file)

Guidance:

Document the licensing and related terms of use for the software as applicable, with the strong recommendation to consider open source licensing. You can refer to Open Source software licenses: https://opensource.org/licenses/, or Choose an open source license: https://choosealicense.com/, or consult with a legal professional for advice if needed. However, if an open license isn't suitable due to specific project requirements or constraints, provide a clear justification for choosing a more restrictive license. In the case where some hybrid software (platforms) is integrated by several software components, multiple licences might co-exist. In that case, a hybrid licensing scheme is applied on a case-by-case basis. Please specify the licence for the integrated Software (platform). Describe your approach to licensing, outlining the specific terms and conditions for usage, modification, and redistribution of the software. You can also add a customized licence.txt if needed.

*** What steps will be taken to help the research community know that your software exists?**

[Type text here]

Example Answer:

A PID (e.g., DOI) will be created for the Software/Software Platform, which can then be cited in related data/paper publications.

Guidance:

To promote awareness of your software within the research community, steps can include leveraging academic networks, conferences, and online platforms, as well as publishing documentation, tutorials, and research articles showcasing its utility. Consider assigning a Persistent Identifier (PID) for your software to enhance traceability and citation. If your software lacks a PID, registering with a PID service is advisable to improve its visibility and recognition within the academic community. Refer to FAIR for Research Software (FAIR4RS) Principles for more details.

*** How will users of your software be able to cite your software? Please provide a link to your software citation file (CFF) if available. (https://citation-file-format.github.io/)**

[Type text here]

Example Answer:

A PID (e.g., DOI) will be created for the Software/Software Platform, which will then be cited in related data/paper publications. CFF for 3D slicer: https://github.com/Slicer/Slicer/blob/main/CITATION.cff

Guidance:

To enable users to cite your software, provide clear guidelines on how to reference it in academic publications, including the preferred citation format and any specific details required for accurate attribution. If available, offer a Citation File Format (CFF) for easy inclusion in citation management systems. If a CFF is not yet available, prioritize creating one to streamline the citation process for users. Refer to https://cite.research-software.org/ for more information.

Preservation and Long-Term Maintenance

*** How and where will your software be archived, after the software is developed?**

[Type answer here]

Example Answer:

We will archive our source code in the Software Heritage archive (https://www.softwareheritage.org/) to guarantee long-term accessibility and preservation. iReceptor Gateway is archived at: https://archive.softwareheritage.org/browse/origin/directory/?origin_url=https://github.com/sfu-ireceptor/gateway

Guidance:

Refer to the Software Heritage archive (https://docs.softwareheritage.org/#landing-preserve) for step-by-step guidance on how to archive your source code/, and https://archive.softwareheritage.org/ to browse the archive.

*** How do you plan to support long-term maintenance of your software?**

[Type answer here]

Example answer:

Long-term maintenance of our software will be ensured through regular updates and archival practices. We will maintain a dedicated repository on platforms like GitHub or GitLab, where versioned releases and documentation will be stored. Source code for iReceptor gateway is maintained on GitHub: https://github.com/sfu-ireceptor/gateway, with major releases stored here: https://github.com/sfu-ireceptor/gateway/releases.Source code for iReceptor gateway is maintained on GitHub: https://github.com/sfu-ireceptor/gateway, with major releases stored here: https://github.com/sfu-ireceptor/gateway/releases.

Guidance:

Plan for the sustainability of your software by outlining strategies for long-term maintenance. Consider versioning, documentation, and archival solutions to maintain your software's integrity and availability for future use.

If using an existing software component/platform, how would you deal with upgrades / patches to third-party software packages that you might use?

[Type answer here]

Example Answer:

To deal with upgrades and patches to third-party software packages, regularly monitor vendors for updates, prioritizing critical fixes (fixing bugs and dependency management) and new features. Test updates for compatibility and security issues, and document procedures for deployment if needed. Whenever there is an upgrade/patch, the community announcement will post the news, as well as on source code website and in related documentation.

Guidance:

Software is evolving all the time, whenever there is a new version for the software that your software (especially the third-party software packages) is dependent on, what do you do to ensure that your Software/Research Software Platform is still functional?

User Support

*** After the software/research software platform has been developed, please include plans for any support-related activities (e.g., platform operations, providing user support, extending / adding functionality, facilitating adoption by new research teams, etc.) in terms of training, support staff allocation, communication channels.**

[Type answer here]

Example Answer:

Software webpage	*https://www.slicer.org/*
Download 3D Slicer	*https://download.slicer.org/*
Slicer documentation	*https://slicer.readthedocs.io/en/latest/*
Github repo	*https://github.com/Slicer/Slicer*
Build instructions	*https://slicer.readthedocs.io/en/latest/developer_guide/build_instructions/index.html*
Slicer tutorials	*https://www.slicer.org/wiki/Documentation/Nightly/Training*
Slicer forum, for community announcements and support	*https://discourse.slicer.org* *https://twitter.com/3DSlicerApp*

Guidance:

It's essential to outline plans for user support activities to ensure its effective utilization. This can include procedures for platform operations, such as monitoring system performance and ensuring data integrity, as well as implementing processes for ongoing software maintenance. Additionally, communication channels can be set up for *users to seek assistance or report issues, and support staff should be available to address inquiries and troubleshoot problems. Training programs should also be considered to educate users on software use.*

Responsibilities and Resources

What resources will you require to implement your software management plan? What do you estimate the overall cost for software management to be?

[Type answer here]

Example Answer:

The Software development will require xx FTE developer in Python, with the minimum annual salary of xxx CAD, for x years. The project requires the use of commercial MATLAB software, the subscription of which is yyy CAD per year, for y years. The overall estimates would be yyyy CAD for y years.

Guidance:

Resources may include personnel (e.g., the proposed software team composition) and cost for documentation, training, and support, as well as tools for version control, documentation management, and communication. Estimated costs can be based on personnel salaries, software licenses, and any additional expenses such as training materials or external consultants. Additionally, consider ongoing maintenance and support costs post the funding cycle.

*** How will responsibilities for managing software activities be handled if substantive changes happen in the personnel overseeing the project's software, including a change of Principal Investigator? Please consider the situation for managing software both during and after the project.**

[Type answer here]

Example Answer:

Usually, the PI will be the contact person to oversee the software management during and after the project. If the change of PI happens during the project, the announcement and replacement will be delivered to both the funder and project documentation. If this happens after the project, the new software maintainer replacement will be updated on the documentation and to the community.

Guidance:

In case of personnel changes overseeing the project's software, it is important to establish clear protocols to ensure continuity in managing software activities. During the project, designate backup personnel and document processes *to ensure transitions. During post-project, ensure comprehensive documentation and consider knowledge transfer sessions for new team members.*

Other Concerns

*** Describe the main external factors that should be considered by developers and users of the software. These could include any security-related information or concerns.**

[Type answer here]

Example Answer:

For data sharing repository development, the software team will implement sufficient cybersecurity measures to prevent any data vulnerabilities of the software platforms.

Guidance:

Make sure to conduct a thorough security risk assessment to ensure that the software incorporates robust security measures to safeguard against external threats. Also consider other factors that could have an impact on your software. For example, potential biases presented in the software algorithms to be developed, compliance with privacy policies, security considerations, reliability requirements, portability / vendor lock, etc.

Date and Sign

The authors of this document will ensure that this Software Management Plan is carried out as specified above.

Name:	[Type here]
Affiliation:	[Type here]
Date:	[Type here]
Signature:	[Type here]