Software
Sustainability
Institute

Checklist for a Software Management Plan

Michael Jackson (ed.)

Version 1.0

10.5281/zenodo.2159713

10 December 2018

Please cite as: Michael Jackson (ed.) (10 December 2018). Checklist for a Software Management Plan (Version 1.0). Zenodo. doi:10.5281/zenodo.2159713. Web site: https://www.software.ac.uk/software-management-plans

Introduction

Research software can take many guises. It can be a 50 line bash shell script for manipulating and filtering files, a collection of 100 line R scripts for running a bioinformatics analysis, 10,000 lines of Java for medical image analysis or 100,000 lines of Fortran for computational fluid dynamics. It may be written in scripting languages such as Unix shell, Python, R or MATLAB or in "traditional" programming languages such as C, C++, Fortran or Java. But, whatever guise it takes, research software is an integral part of the modern research ecosystem.

When developing research software, it is easy to focus only on goals and activities such as collaborating with other researchers, writing papers, attending conferences and applying for funding. Together, the demands of daily research practice can all conspire to prevent proper planning for the development of research software.

A Software Management Plan (SMP) can help you to define a set of structures and goals to understand your research software including what you are going to develop; who the software is for (even if it is just for yourself); how you will deliver your software to its intended users; how it will help them; and how you will assess whether it has helped them, and contributed to research, in the ways that you intended. An SMP also helps you to understand how you can support those who wish to, or do, use your research software; how your software relates to other artefacts in your research ecosystem; and how you will ensure that your software remains available beyond the lifetime of your current project.

Though an SMP can be of most benefit when starting a project to develop research software, there are benefits to adopting one on a project that is already underway. An SMP provides a way to draw together and summarise research software-related aspects that have already been decided and doing so can reveal additional aspects or options that weren't considered, or weren't applicable, when the project began.

This checklist will help you to write an SMP. It consists of sections that cover the key elements that an SMP should include. Within each section questions are posed to help you to complete that section. Complementary guidance and links to other resources are also provided. Not all questions are relevant to all projects and the extent to which a specific question can be answered may depend both upon the nature of your research software and its current state of development. Consider your SMP as a living document, to be reviewed and revised as the development of your research software progresses.

Use of this checklist

The Software Sustainability Institute provides this checklist on an "as-is" basis. It makes no warranties regarding any information provided within and disclaims liability for damages resulting from using this

Acknowledgements

This checklist has its origins in Chue Hong, Neil (2014) "Writing and using a software management plan", The Software Sustainability Institute https://www.software.ac.uk/resources/guides/software-management-plans, 2014. The following people offered valuable advice and guidance which contributed to both the content and form of this checklist - Mario Antonioletti, The Software Sustainability Institute; Neil Chue Hong, The Software Sustainability Institute; Peter Cock, The James Hutton Institute; Steve Crouch, The Software Sustainability Institute; Robert Davey, The Genome Analysis Centre; Carole Goble, The Software Sustainability Institute; Catherine Jones, STFC; Sarah Jones, The Digital Curation Centre; Katrin Leinweber, Technische Informationsbibliothek; Mark Plumbley, Centre for Vision, Speech and Signal Processing, University of Surrey; Chris Rawlings, Rothamsted Research; Marta Ribeiro, The Digital Curation Centre; John Robinson, The Software Sustainability Institute; Shoaib Sufi, The Software Sustainability Institute.

Changes

- 1.0 (2018-12-10) Curtailed checklist to manageable form consisting of sections formerly in a minimal software management plan section, plus questions relating to support, other research objects and deposit and preservation. doi:10.5281/zenodo.2159713
- 0.2 (2018-10-12) Changed checklist formatting and presentation. doi:10.5281/zenodo.1460504
- 0.1 (2016-03-18) First draft. doi:10.5281/zenodo.1422657

What software will you develop?

Questions to consider:

- Are you developing new software from scratch? If so, then what will your software do?
- Are you refactoring, extending, optimising, or parallelising existing software written either by yourself or by others? If so, what are the significant changes that you plan to make?
- Will your software have a name, if it does not already have one? Do you have one in mind? Is this name unique and meaningful and not in violation of any existing trademarks held by others?

Guidance:

When you come to choose a name for your software, see the Software Sustainability Institute's "Choosing project and product names" (https://software.ac.uk/resources/guides/choosing-project-and-product-names).

Who are the intended users of your software?

Questions to consider:

- Is there just one type of user or are there many?
- Is your software for those new to your research field, for experts in your field, or for both?
- What, if any, software installation and configuration skills, knowledge and expertise will your users need? Will they need to be familiar with building and installing software via the command-line?
- What software development skills, knowledge and expertise will your users need? Will they need to write their own code to be able to use your software?
- What are the popular operating systems, web browsers, compilers or language versions used by your intended users? Would supporting additional operating system, web browsers, compilers or language versions encourage new users to adopt your software?

Guidance:

Your intended users could include yourself, your research group, your department or institution, your project, your research community, other research communities, or the general public.

If your software is a framework that allows users to develop their own plug-ins or is a library that users can use within their own research software, your users will need some experience of software development to get the most from your software.

If the environments popular with your intended users are not those you would usually prefer, then target the environments of your intended users. Do not expect your intended users to migrate to your preferred environment as this puts a barrier between them and your software. For example, if Windows is the prevalent operating system of your intended users then it would be unreasonable to expect them to migrate to Linux just to use your software. Software that compiles or runs under two or more of Windows, Unix/Linux and Mac OS X will have a larger potential user community than software that is restricted to a single operating system. For web-based applications, those which can run within two or more of Google Chrome, Microsoft Internet Explorer, Mozilla Firefox and Softonic Safari have a larger potential user community that those that restrict users to a single browser.

How will you make your software available to your users?

Questions to consider:

- Will you release your software at all?
- Will you release binaries, libraries or packages?
- Will you release your source code? Do your funders, or other stakeholders, require you to release your source code?
- Under what kind of license will you make your software available?
- How will users access your software?
- Will users have to register to access or use your software?
- Will your software be accessed as an online service or via a web portal?
- Will you charge users a fee to access or use your software? What will the revenue generated by these fees be used for?

Guidance:

Making your software available to other researchers allows them to inspect, replicate, reproduce and reuse your research, as manifested in your software, in the short term and to inspect, for the historical record, in the long term.

Releasing binaries such as executables, Docker or Singularity containers or virtual machines can allow other researchers to replicate and reuse your research. Depending on the implementation (e.g. how configurable they are) they may also support reproducibility. However, they do not allow others to inspect your software to understand exactly what was run and how your results were produced. A complementary paper can provide this information, but papers do not produce the results, the source code that is run does.

Releasing your software as open source software gives other researchers the freedom to inspect your software, to read its source code and understand exactly what was run and how your results were produced and to validate your results. Open source software also supports replicability, reproducibility and reuse. Releasing your software as open source could encourage its adoption by others and, in turn, increase the potential impact of your research, as embodied in your software. Even if your funders or other stakeholders do not require you to release your source code, consider releasing your source code anyway if they have no objections. See OSS Watch's "Benefits of Open Source Code" (http://oss-watch.ac.uk/resources/whoneedssource).

A software licence is an explicit, and legally-binding, statement of what others can, and cannot, do with your software and any obligations upon them. If your software has no licence then others have no explicit permission to use, modify or redistribute your software. See the Software Sustainability Institute's guide on "How to choose a software licence" (https://softwaresaved.github.io/software-deposit-guidance/HowToChooseSoftwareLicence.html).

There are many ways in which you can distribute your software. These include:a binary executable that can be run directly; bundled in an installer program; an archive (.zip or .tar.gz) of binary executables or libraries, or as Python or R packages; an archive of source code; via a download link on a web site; via e-mail from you; via access to a source code repository hosted at your institution or on a third-party site such as GitHub (https://github.com), GitLab (https://gitlab.com), BitBucket (https://bitbucket.org), LaunchPad (https://launchpad.net) or Assembla (https://www.assembla.com); via a digital repository such as Zenodo (https://zenodo.org) or figshare (https://figshare.com) or one provided by your institution or community.

Building or compiling software can be complicated and time-consuming. If you can, provide your software in a form that allows it to be deployed and used without requiring your users to build it. This saves your users both time and effort and can be especially valuable if your users are not software developers.

If you don't need to monitor access to your software, or to restrict access to authorised users only, consider allowing anonymous access to and use of your software.

Providing free and anonymous access to your software gives users immediate access to it to "give it a go". This can also help make your software more appealing than competing software that has similar functionality, but which requires registration or charges a fee for its use.

If you want, or need, users to pay a fee before users can access or use your software, you need to tell some users why they need to pay this fee and what they get in return. Your fee might be needed to help fund your time to develop your software, to fund your time to provide support for your software, or to help pay for any infrastructure or third-party dependencies you use.

How will you support those who use your software?

Questions to consider:

- How much effort will you have available to support your users, if any?
- What level of support will you offer, if any?
- How will users ask for help?
- How will you manage their requests for help?
- Where will you publish information about the nature and level of support available?
- Will users be able to see what other users have asked and the associated answers?

Guidance:

No one has a right to expect support for freely provided software. However, if you release your software, or make it available for use, then you will get questions about how to use it. An ignored request for help or advice, accidental or not, can lead to a disgruntled user who may bad-mouth your software or you or merely stop using it.

Responding to requests for help does not imply that you must spend time in fixing bugs or implementing features when they ask, it merely acknowledges that you are aware that they got in touch.

How much effort you or your group have available to support users will be up to you. You are also free to choose to provide no support if you do not have the time or effort available. It is always good to make clear what, if any, support you will provide clear on your website or in your documentation, e.g. "we will reply to all e-mails within a week and will let you know when, or if, we can address your issue". A user will always want their problem to be solved as quickly as possible, and may become disgruntled, and might even stop being a user, if this is not the case. If you are clear and honest about the level of support you can provide, then they have fewer grounds for complaint.

If you can offer help and support then there are many ways in which you can have your users request help and support, including:an e-mail to you or to an e-mail list or forum, or creating an issue in a ticketing or issue tracking system (https://en.wikipedia.org/wiki/Issue_tracking_system).

There are many ways to publish information about the help and support requested by other users, and how these were resolved, including:e-mail archives, lists of frequently asked questions, or a publicly-visible ticketing or issue tracking system. Encouraging users to search these resources before getting in touch can help users to help themselves and reduces the overhead you need to spend providing support.

See the Software Sustainability Institute's "Supporting open source software" (https://software.ac.uk/resources/guides/supporting-open-source-software) and "Top tips for managing support requests" (https://www.software.ac.uk/blog/2014-01-21-top-tips-managing-support-requests). Many of the points apply not just to supporting open source software, but any software.

How will your software contribute to research?

Questions to consider:

- Will it help your users, or yourself, to conduct research more easily? Produce results more rapidly? Produce results to a higher degree of accuracy or a finer level of detail? Exploit the power of super-computers?
- Will it help your users, or yourself, to further research by enabling research that cannot be done at present?
- Will your software, in some form, implement a novel solution to a research problem?
- What are the limitations of similar research software that already exists? How will your software be better?
- What are the foreseen benefits for each type of user?

Guidance:

When developing research software, it is good to be able to explain how it will contribute to research, whether this is research to be done by yourself or by others using your research software.

How will your software relate to other research objects?

Questions to consider:

- Which published research objects relate to your research software?
- Are there published papers that describe the research that your software will enable?
- Will your software extend or depend upon other research software?
- Will your software consume data sets or invoke online services?
- Will you publish papers about the research you have done to which your software has contributed?
- Will you publish any data sets produced by your software?

Guidance:

Taken together, research software, alongside data, facilities, equipment and an overarching research question can be viewed as a research activity or experiment, worthy to be published. Understanding, capturing and publishing information about the research objects related to your software (e.g. publications, data, other research software, facilities, experimental equipment, services) allows others to see your software not in isolation but as an integral part of addressing a research question.

Consider not just research outputs directly produced by your or your project but those produced by others who have used your software and have cited it in their own papers or software.

For more on research objects, see "The FAIR Guiding Principles for scientific data management and stewardship" (https://doi.org/10.1038/sdata.2016.18) encouraging the publication and use of digital research objects which are Findable, Accessible, Interoperable and Reusable, and researchobject.org (http://www.researchobject.org/) which explores machine-readable representations of these objects.

How will you measure your software's contribution to research?

Questions to consider:

- What evidence do your funders and other stakeholders expect you to present to show that your software has contributed to research?
- Will you measure who has downloaded your software?
- Will you measure who has used your software?
- Will you gather information on publications that describe research which your software has helped to enable, including those by other researchers?
- Do you intend to have a recommended reference or citation for your software?

Guidance:

There are many ways to quantify interest in your software, including how many people have shown interest in your software, how many have used your software, and what they have used it for. These include:number of downloads; number of forks, if hosted on GitHub, GitLab or BitBucket; number, and nature, of code contributions such as bug fixes, enhancements or extensions (e.g. number of pull requests or e-mails with patches); number of e-mail list or forum members; number of support requests such as e-mails, bug reports, feature requests or open issues; papers you and your users have published to which the use of your software contributed; citations of these papers; blog posts by others about how they used your software; numbers of attendees at conference or workshop talks, demonstrations, poster sessions or tutorials.

In the same way that it is expected that others cite your papers if they have used them within their research, you can request that others cite your software in any publications derived from the use of your software. This helps you to get credit for your research as manifested in your software. You can search for these citations in papers, publications, other software, documentation, web pages and blog posts to gather information relating to impact of your software. You can also ask others to let you know if they publish any such papers. This can help you to demonstrate the impact of your software and its contribution to research to your funders, employers, stakeholders and your community.

Consider providing a CITATION.cff file with your source code, with your citation expressed in the Citation File Format (CFF) (https://citation-file-format.github.io/).

See the FORCE11 Software Citation Working Group' Software Citation Principles (https://www.force11.org/software-citation-principles) for more information on the value and uses of software citation.

Meeting users at conferences and workshops, contacting them via email, and creating short questionnaires (e.g. via Jisc Online Surveys (https://www.jisc.ac.uk/online-surveys) distributed via your website or Twitter, all are effective ways to gather information on how users have used, and have benefited from, your software.

Where will you deposit your software to guarantee its long-term availability?

Questions to consider:

- Will you deposit releases of your software into a digital repository?
- What digital repositories are appropriate for you to use?
- Has your funder or publisher recommended or mandated a digital repository to use?
- Does the digital repository give you a unique persistent digital identifier for your deposit?
- Can the digital repository accommodate the size of your deposit?
- Are the policies of the digital repository acceptable to you?
- Is the longevity of the digital repository acceptable to you?
- Is the digital repository free or do you have to pay a fee? If there is a fee, is this a one-off payment and can you afford it?

Guidance:

Research software has a lifetime and this lifetime may extend far beyond that of your project or the period for which you actively develop and maintain it. Your user community may continue to need it, use it, or develop it, long after your project has completed and you have moved on to other projects.

Repository hosting services such as GitHub, BitBucket, GitLab, Assembla or Microsoft Visual Studio Team Services (https://www.visualstudio.com/team-services/) that host source code repositories are great for the development of active software. However, they are not so well-suited for ensuring long-term availability. They don't create unique persistent digital identifiers (https://en.wikipedia.org/wiki/Persistent_identifier) for your repositories, but only provide URLs. Your repositories could later be deleted by yourself or your institution, or the repository hosting service may cease to operate, breaking links to these URLs,

Depositing your software within a digital repository ensures that the exact versions of your software upon which you have published your research results are retained and remain available for future use both by yourself and by other researchers. You enable other researchers, and your future self, to inspect, replicate, reproduce and reuse your research, as manifested in your software, in the short term and to inspect, for the historical record, in the long term. It allows your software to remain available beyond the lifetime of your current project or your current employment at a specific institution.

Digital repositories can provide unique persistent digital identifiers for software which can be cited by others which, in turn, can help you to get attribution and credit for your research software when it is used by others.

See the Software Sustainability Institute's "Software Deposit Guidance for Researchers" (https://softwaresaved.github.io/software-deposit-guidance/) and Neil Chue Hong, Steve Crouch, Simon Hettrick, Tim Parkinson, Matt Shreeve, "Software Preservation Benefits Framework", The Software Sustainability Institute and Curtis+Cartwright Consulting Ltd, 7 December 2010 (https://www.software.ac.uk/attach/SoftwarePreservationBenefitsFramework.pdf).

Licence