We are in the process of publishing the principles. If you want to make new comments or changes please do in the following documents:

- Publication - FAIR OSSP
- Publication supplementary material - Fears

FAIR Open Source Software Principles

Draft version 0.3

- The previous content and comments collected in V0.2 are available as an appendix at the end of this document.
- Contributions are logged in this spreadsheet. If you think you contributed and your name is missing please add a comment in the spreadsheet including your name, contribution as well as all the personal details requested.
- Please make sure you're logged into Google, otherwise we cannot attribute your contributions.
- The expected release of "version 1.0" is October 25.
- For more information … https://github.com/SoftDev4LS/open-source-software

Introduction

Background

Software is fundamental for building scientific knowledge[1]. Open source can contribute to the **quality** and **sustainability** of software and thus improve research. Open source increases transparency, discoverability and visibility as well as engagement, recognition and trust among the community. It maximises the reproducibility of results produced by the software and facilitates its reusability. Opening code to the public is also an opportunity for developers to showcase their work, thus it contributes to the adoption of best practices[2].

Like open access and open data, open source is a mechanism that can be used by organisations, especially funders, to contribute to better software, science and innovation. It is highly recommended that organisations involved in producing software have a policy adopting open source software principles.

---

[1] "UK Research Software Survey 2014 - Zenodo." 2015. 19 May. 2016 <http://zenodo.org/record/14809>

[2] Leprevost, Felipe da Veiga et al. "On best practices in the development of bioinformatics software." *Frontiers in genetics* 5 (2014): 199.

In a research environment, software should not just be a means to an end but a collective intellectual product: an asset that can be reused by the research community to become more efficient in the production of research outcomes, which strengthens the case for further funding to potentially fund future developments.

Purpose and scope

This manuscript aims to describe a common and minimum set of open source software principles to improve quality and sustainability of scientific software. It also aims to encourage organisations and projects to adopt these practical principles and turn them into the basis of their open source policy.

Like the FAIR Guiding Principles for scientific data management and stewardship, these principles also aim to make software and source code Findable, Accessible, Interoperable and Reusable (FAIR)[3].

These principles do not aim to describe the details of how to develop software but rather state the fundamentals of open source software development. They should be applied using common sense and following complementary guidelines in software development. To show users how to implement these principles, we are working on a separate manuscript with a set of recommendations and *good-enough* practices.

The principles are accompanied by a list of arguments addressing common questions and fears raised by the community when open sourcing software.

Audience

The target audience of this document is not just software developers but leaders and managers of scientific organisations, and the teams and projects participating in the decision-making of how to develop software. Though this policy has mostly been developed in, and received feedback from, the life science community, the document and its principles generally apply to all research fields.

Contribution

This policy is open to comments and contributions from domain experts in software development who care about quality and sustainability of software and software development. Information about this policy is available in the following repository: https://github.com/SoftDev4LS/open-source-software

[3] Wilkinson, Mark D et al. "The FAIR Guiding Principles for scientific data management and stewardship." *Scientific data* 3 (2016).

Principles

Findable - Source code easily discoverable

Facilitate the discoverability of software projects by registering software, source code repository and license in a public registries relevant to a specific community[4]. eg. bio.tools[5] or biojs.io[6] in the life sciences. Making your source code findable:

- Increases the visibility of the project, software, its successes and its contributors
- Increases the chances of collaboration and reusability

Accessible - Publicly accessible open source code from day one

Start a project in the open from the very first day in a publicly accessible repository, since the longer a project is run in a closed manner, the harder it is to open source later. Adopting open source practices does not mean to assume responsibility for managing the community. Opening code and exposing the software development life cycle publicly from day one:

- Encourages contributions from the community
- Exposes work for community evaluation and suggestions
- Facilitates the discovery of existing software development projects
- Provides a historical record of contributions from the start of the project
- Increases the chances of collaboration and reusability
- Facilitates the adoption of best practices by showcasing the development process
- Increase transparency through community scrutiny
- Promotes trust in a software project

If there is a compelling reason why the code cannot be open source, still inform the community about the software and software development process. If this policy is endorse still provide information about the software project in a publicly accessible repository.

More information about how to mitigate fears of open-sourcing your software project are available as an appendix of this policy.

---

[4] Ménager, Hervé et al. "Using registries to integrate bioinformatics tools and services into workbench environments." *International Journal on Software Tools for Technology Transfer* (2015): 1-6.
[5] Ison, Jon et al. "Tools and data services registry: a community effort to document bioinformatics resources." *Nucleic acids research* 44.D1 (2016): D38-D47.
[6] Gómez, John et al. "BioJS: an open source JavaScript framework for biological data visualization." Bioinformatics (2013): btt100.

Interoperable and Reusable - Licensed source code for use and reuse

Provide documentation and guidelines for other projects and software to use and redistribute source code. Adopt a suitable open source license and include it within a publicly accessible repository, and also ensure software complies with third party software licenses. Having a licence attached to source code:

- Clarifies the responsibilities and rights placed on third parties wishing to use, copy, redistribute, modify and/or reuse your source code
- Protects the software's intellectual property
- Provides a model for long-term sustainability
- Enables using the code in jurisdictions where "code with no license" means it cannot be used at all.

If your institution or project already has a license policy, use the license suggested in it. Otherwise, we advise choosing a OSI-approved Open Source License following these guidelines.

Open development

Open sourcing your software does not mean the software has to be developed in a collaborative public manner. Although it is recommended, these principles do not mandate a strategy for collaborating with the community. Instead they are looking for software projects to clarify their plans to engage with the community. Thus these principles assume the project will be clear and transparent about how to contribute to the project, its governance model, and its communication channels.

Fears of open sourcing & some ways to handle them

Principal Investigator Perspective

- "My group's software (unique methodology, specificity, knowledge) gives our research a competitive edge that will be lost if we released our code in the open."

    - Research is way more than the software that facilitates it, and the software usually lags the ideas
    - If your edge is only the software, it's just a matter of time until you're no longer competitive
    - Your research is probably held back by the bugs you haven't found yet that others would
- “It’ll be impossible to open source software which we have developed in collaboration with companies / industry because it will be too complicated to resolve the legal issues”
    - You could consult an IP specialist such as OSS-Watch
    - Using an OSI-approved license will make it easier as commercial IP legal departments recognise them
- “Getting our code ready to open source will take a lot of time and effort”
    - The expected time benefits often outweigh the initial investment
        - You
        - (See: Taverna move to Apache incubator process)
    - It will help your internal development team
        - Enhances skills and CV
    - If your software provides value and others use it, that will strengthen your case for more funding, which will sponsor that time and effort
- “An open source license will put off life sciences companies”
    - Many life sciences companies use open source software - they care about functionality and support
        - Examples:
- “We’ll have to support users, perhaps who are not in our field”
    - Supporting users can help foster collaboration and reputation in related fields, which can lead to new scientific collaborations and funding
    - Research funders want to see impact outside of your specific field
    - You are likely being affected by the same issues those users would want fixed
    - You can resolve issues early that may affect your own research in the future
- “I will be liable if something bad happens when someone uses my software”
    - A good OSI-approved license will disclaim/waive liability better than having no license or an untested academic license
- My reputation will suffer if someone uses my software incorrectly, e.g. if someone publishes a paper based on results that use my software, but then if the paper must be withdrawn or retracted, my software may be blamed
- “People will scoop me”
    - Open-sourcing will tend to lead to more collaborations (leading to more publications) than other people scooping you
- “I’ll have to pay for the cost of infrastructure”
    - There are many providers who allow you to use their resources for free if you are an open source project
    - Opening your source code does not require infrastructure on your part, you can just upload it to a repository hosting site like GitHub

Developer Perspective

- “My software is such a bunch of hacks that no-one will want to contribute to it”
    - If it’s closed, no-one can contribute to it anyway. If it’s useful and open, then people could still contribute, even if the code quality is not great
    - Even inspiration (*how things were done once*) is worth sharing
- “I am the expert in this field. I am the best person to build this software and don’t see how letting other people contribute will help my software”
    - You may be the expert in your field, but the software is just a tool you are building. You are probably not the expert in all aspects of the software you are creating.
    - Providing open code and (open) training might help others to become experts as well
    - Documentation should help train new contributors: code comments, usage, development setup
- Someone takes my software and sets it up as a service, but wrongly. This would reflect badly on my original software (e.g. they used the wrong data).
    - Way #1: You can use trademark enforcement of the name to make it clear that this is not your or your software’s responsibility
    - Way #2: Document your setup [better], automate it (use Docker or similar tools); keep communication channels in order
- “I should be doing science, not open-sourcing software”
    - Science is fundamentally based on accuracy and trust in results. Open source code can be more trusted (cf security community) than black box solutions (see Shining light into black boxes paper)
    - Don’t overdo it if you don’t have time, just get what you have out there, it takes very little time
- “I won’t receive any credit for my work”
    - Open-sourcing has been shown to improve citation and research impact in certain communities
    - Consider publishing your work in The Journal of Open Source Software, see http://joss.theoj.org/about for more information
- “I won’t be able to publish a paper about my research / about my software”
    - There are now many avenues for publishing software papers, and many encourage open source licenses

Appendix: Open source policy 0.2

The basics

1. Open source your project unless you cannot (open by default)
    a. Unless there is a compelling reason not to do so, provide the source code in a publicly accessible source code repository as early as possible
    b. Work to mitigate fears from others in your project about open-sourcing
        i. See "Fears of open sourcing & some ways to handle those (which we came up with and should publish somewhere)"
2. Attach a suitable license to the source code repository, and ensure compliance with third party software licenses
    a. Article on open source license compatibility
        i. http://www.wegtam.net/article/combination-and-compatibility-open-source-software-licenses
    b. Unless there is a very good reason not to, you should use an OSI-approved Open Source License.
        i. If your institution or project already has license policy, use that one.
        ii. Otherwise we advise choosing one of the license here (link), depending of your preferences
    c. If you cannot use an OSI-approved Open Source License, at least adhere to template of OSI-approved Open Source License or Free for academic use Licence.
3. Open development should be expected from the start of the project, if possible
   Read and apply 10 simple rules for Ten Simple Rules for the Open Development of Scientific Software
    a. Development should be done in a publicly visible version-controlled repository
        i. This can be an institutionally hosted repository or externally hosted e.g. on GitHub, BitBucket
        ii. You should use semantic versioning (see http://semver.org/) appropriately at a later stage of the project (e.g. when the first external user becomes involved)
    b. You should make it clear what is being communicated via which channels "Channels should be well documented"
        i. For guidance of what channels are useful see this document
        ii. Produce user documentation, a README file, and how to install the software
    c. All decision-making should be transparent and on the recognised project communication channels
        i. Avoid "private" decisions (e.g. discussing with your co-developer in the coffee room without then recording the decision publicly)
        ii. Have a public roadmap
        iii. Strive to include the community in decision-making, where possible
    d. Be clear how people can contribute to your project
        i. Make it clear you really want people to contribute

    ii. Make it clear how to set up your development environment and build the software
    iii. Good practice is to create a CONTRIBUTING file, that describes how to contribute (some more details on GitHub docs)
    iv. Create guidelines for common contribution tasks
    v. Have a code “style guide” and build enforcement into your development workflow
        1. For examples: https://github.com/google/styleguide/
        2. Use linters to enforce adherence to the style guide (example: eslint)
    vi. Make sure contributions are allowed and compatible
        1. Contributions should be in scope
        2. Contributions should be checked for license compatibility (see: http://www.gnu.org/licenses/license-list.html)
        3. Contributors should have the right to contribute (see copyright)
    vii. Decide if you need a formal Contributor Agreement
    viii. Consider labelling issues by “ease of engagement” to encourage new contributors
e. Ensure your software has accessible and pragmatic documentation
   Different aspects of documentation
f. Your project should have a Code of Conduct in your repository and linked from your website (“Be nice”)
    i. Make people feel appreciated for their contributions - thank them
    ii. Don’t assume you know why someone did something, check why
    iii. Examples are here: contributor covenant

- Use existing tools to help manage your software, rather than creating your own (see link on OSS-Watch website, http://oss-watch.ac.uk/resources/communitytools)

Developing the community

4. Work to build a community around your software
    a. Document what your software does!
        i. Publish use cases
    b. Publicise your software through conventional and non-conventional means
        i. Consider nominating “evangelists” from your community
        ii. Use videos to help. Consider getting recognised scientists to record / appear in these
        iii. Consider the types of users you wish to attract when presenting your software at conferences, meetings, etc. to grow its community e.g. end-users, developers, user/developers.
    c. Create opportunities for participation
        i. For instance hackathons, tutorials, workshops
    d. “Piggy-back” on existing events that your community attends

        i. e.g. BOSC, ISBE

5. Publish your software in a DOI-issuing repository
    a. You should publish under the same license
    b. You should get a DOI for each major release
        i. And versions associated with publications coming from your project
    c. Zenodo and FigShare both enable DOIs for software https://guides.github.com/activities/citable-code/
    d. Advertise the DOI at the top level of your repository and in a CITATION file

Striving for the meritocratic bazaar

6. Become more open as your project matures
    a. Be transparent about your current governance model
    b. Formalise your communication channels for your contributors
    c. Open up your decision-making
        i. For examples of different open source governance models see: http://oss-watch.ac.uk/resources/governancemodels
    d. Formalise roles in the project and the “promotion path”
        i. How do people become associated with key roles?
    e. Recognise that “you are not your code”
        i. Be able to separate criticism of your software from criticism about yourself

Notes from the all hands breakout group:

- Who:
    - Define stakeholders
        - Ie. comms, developers, UX, …
        - Define goals and roles
            - Based on what is needed for projects
    - Define types of community
- Link data with software
    - Get data community involved and aware
- Promotion/support/communication
    - Many channels
        - Social media
        - Events
            - Conferences
        - Newsletters
    - Identify channel of target audience
        - Look for those with impact
    - Lots of effort to maintain channels
        - Consider if is it worth it
    - Effort vs impact matrix to help prioritisation
    - What is the purpose

        - Difference between give information (out) vs getting feedback (in)
    - champions/ambassadors to reach out
        - Identify them
        - Role on providing support
        - Benefits for champions
            - recognition/credit
- Use ontology/model for communities
    - To help describing community
        - ENVRI model?
    - To help discovery and connection of communities
- Support the promotion of the outcomes and services of the community
    - Within the community
    - With related communities
    - Outside the community
- Documentation (guidelines)
    - Content
        - What the software does, what is it for, how to use it, how to contribute
    - Types
        - Users, developers
    - Steps
        - Start with a quick start
        - More elaborated documentation
            - Document use cases
    - Identify who and what to document
- Software discovery yes, but not marry to one technology (ie. DOI)
    - Identifiers important
    - But maybe more important is to have software discoverable in a software registry or publication
    - Allow freedom but provide recommendation of technology/system
        - Ie. registration of tools, omics, tools,
        - Ie. identifiers: DIO, EPIC,

Fears of open sourcing & some ways to handle them

(See Victoria Stodden's work on identifying fears: http://stanford.edu/~vcs/papers/SMPRCS2010.pdf)

- "My software is such a bunch of hacks that no-one will want to contribute to it"
    - If it's closed, no-one can contribute to it anyway. If it's useful and open, then people could still contribute, even if the code quality is not great
    - Even inspiration (*how things were done once*) is worth sharing
- Someone takes my software and sets it up as a service, but wrongly. This would reflect badly on my original software (e.g. they used the wrong data).

    - Way #1: You can use trademark enforcement of the name to make it clear that this is not your or your software's responsibility
    - Way #2: Document your setup [better], automate it (use Docker or similar tools); keep communication channels in order
- "My group's software (unique methodology, specificity, knowledge) gives our research a competitive edge that will be lost if we released our code in the open."
    - Research is way more than the software that facilitates it, and the software usually lags the ideas
    - If your edge is only the software, it's just a matter of time until you're no longer competitive
    - Your research is probably held back by the bugs you haven't found yet that others would
- "Other people need the 'right data' to be able to use my software, and I can't give this to them because it's protected/confidential/etc so my software is of no use"
    - You can provide simulated/reference data which you should have in any case to help with testing
- "I am the expert in this field. I am the best person to build this software and don't see how letting other people contribute will help my software"
    - You may be the expert in your field, but the software is just a tool you are building. You are probably not the expert in all aspects of the software you are creating.
    - Providing open code and (open) training might help others to become experts as well
    - Documentation should help train new contributors: code comments, usage, development setup
- "It'll be impossible to open source software which we have developed in collaboration with companies / industry because it will be too complicated to resolve the legal issues"
    - You could consult an IP specialist such as OSS-Watch
    - Using an OSI-approved license will make it easier as commercial IP legal departments recognise them
- "Getting my code ready to open source will take a lot of time and effort"
    - The expected time benefits often outweigh the initial investment
        - You
        - (See: Taverna move to Apache incubator process)
    - It will help your internal development team
        - Enhances skills and CV
    - If your software provides value and others use it, that will strengthen your case for more funding, which will sponsor that time and effort
- "An open source license will put off life sciences companies"
    - Many life sciences companies use open source software - they care about functionality and support
        - Examples:
- "I'll have to support users" / "I'll have to support users who are not in my field"
    - Supporting users can help foster collaboration and reputation in related fields, which can lead to new scientific collaborations and funding

    - Research funders want to see impact outside of your specific field
    - You are likely being affected by the same issues those users would want fixed
    - You can resolve issues early that may affect your own research in the future
- “I will be liable if something bad happens when someone uses my software”
    - A good OSI-approved license will disclaim/waive liability better than having no license or an untested academic license
- My reputation will suffer if someone uses my software incorrectly, e.g. if someone publishes a paper based on results that use my software, but then if the paper must be withdrawn or retracted, my software may be blamed
    - ??? be it open or not, software will be distributed and used by people
    -
- “I should be doing science, not open-sourcing software”
    - Science is fundamentally based on accuracy and trust in results. Open source code can be more trusted (cf security community) than black box solutions (see Shining light into black boxes paper)
    - Don’t overdo it if you don’t have time, just get what you have out there, it takes very little time
- “I won’t receive any credit for my work”
    - Open-sourcing has been shown to improve citation and research impact in certain communities
    - Consider publishing your work in The Journal of Open Source Software, see http://joss.theoj.org/about for more information
- “People will scoop me”
    - Open-sourcing will tend to lead to more collaborations (leading to more publications) than other people scooping you
- “I won’t be able to publish a paper about my research / about my software”
    - There are now many avenues for publishing software papers, and many encourage open source licenses
- “I’ll have to pay for the cost of infrastructure”
    - There are many providers who allow you to use their resources for free if you are an open source project
    - Opening your source code does not require infrastructure on your part, you can just upload it to a repository hosting site like GitHub

Notes from the all hands breakout group:

- People have particular technical problems, documentation for developers for FAQ of how to solve life science specific problems that are not usually found in standard OS communities: Large reference datasets, different use scenarios (commandline + web interfaces + Web services), database updating strategies, etc
- Detach yourself from the code, it does not describe you, everybody know that in the real world you often cannot do great code
- We are not trained CS, we are researchers, just get it out there
- You can organize a code review within your group, or some safe environment
- Document it in a way that installation is no problem.

- API > Installable > Dockerize > VM For reusability. But Docker and VM can really help distribute your functionalities when is not in great shape
- For niche software it's less important that the code has high standards, often its ability to document the details method is enough. On the other hand, for competitive software it's important to have good standards or users will choose some other
- Setup a channel for feedback before they report a failure to the world
- If your concern is that only you have the data and you cannot share it, just have it out so people can read it, not even so important that it runs. En even better solution is to provide simulated data