

Proposal for ESS Open Source strategy implementation roadmap

v1.0 Oslo 25-27 July 2022

*Authors: Harri Lehtinen <harri.lehtinen@stat.fi>, Romain Tailhurat
<romain.tailhurat@insee.fr>, Jon Folkedal <jon.folkedal@ssb.no>, Neville De Mendonca
<neville.demendonca@ons.gov.uk>, Trygve Falch <trygve.falch@ssb.no>*

Table of Contents

Goal	2
Summary of the strategy	2
Culture	2
Foundation	2
Communication	3
Fostering	3
Process	3
Governance	3
Funding, grants	4
Product management	4
Peer review process	4
Documentation	4
Types and specificities	4
Techniques	5
Foundation	5
Repository structure	5
Handling changes	5
Accepting code from contributors	5
Building and distributing	5
Architectural principles	6

Goal

A [draft proposal for an ESS open strategy](#)

(https://ec.europa.eu/eurostat/cros/system/files/03._os_ess.docx) has been presented at the DIME-ITDG meeting in June 2022. The participants of the meeting have validated the strategy and proposed that an informal group or a task force be created to convey additional work on the subject.

The goal of this document is to provide a very short summary of the strategy and to propose a roadmap for the task force.

This roadmap is presented as a non prioritized list of topics that could be addressed by the task force. It is structured around three main domains:

- the cultural requirements for developing and maintaining Open Source projects,
- the process that will govern the ESS-OSS (European Statistical System - Open Source Software)
- the technical foundation of the projects.

As much as possible, this roadmap will point to existing documentation and references that will be starting points to the actions taken by the task force. This document will also specifically make recommendations on different topics that should be taken into consideration when moving forward. The recommendations are based on industry wide best practices, but also rely on previous work from various ESS-Net projects, like SCFE (https://ec.europa.eu/eurostat/cros/content/projects-deliverables_en) and I3S (https://ec.europa.eu/eurostat/cros/content/projects-deliverables-0_en), as well as from extensive work done by the UNECE community, specifically CSPA (<https://statswiki.unece.org/display/CSPA>).

Summary of the strategy

- 1) OSS is recommended as the default option ("OSS by default")
- 2) Possible to derogate from the default OSS option in justified cases
- 3) Adoption of OSS should be encouraged and promoted through training, hackathons, conferences, etc.
- 4) All code developed by the NSI should be published as OSS
- 5) A culture of openness and transparency

Culture

Foundation

We believe that going Open Source in general and for the ESS in particular is only possible with the right culture - which may or may not be already existing at NSI. This section is pointing at action that could help understand what is this cultural environment and to build one.

- Give examples of open cultures and organization: in order to provide a clear and shared understanding of what is an open cultures
- Clarify current status in the ESS, UNECE, NSI's and create a way forward
- Reach out to existing pre ESS-OSS works and projects (ONS, Insee, Statistics Norway)
- Elaborate on what is "Coding in the open" (see <https://gds.blog.gov.uk/2017/09/04/the-benefits-of-coding-in-the-open/>)

Communication

The ESS-OSS strategy will benefit from an integrated communication capability, meaning that it will manage to promote the strategy itself at different levels and communicate results (e.g. reuse and collaboration).

- Propose a way to promote OSS projects in an open and flexible way
- Devise a publication process (with link to the peer review process)
- Find a way to promote the ESS OSS initiative and communicate results to statisticians and IT staff
- Arrange presentation of the strategy, roadmap and principles in various setups (top management meeting, conferences, ad hoc meeting)
- Build bridges to other communities (eg scientific communities), identify most potential communities

Fostering

Fostering here means helping non Open Source savvy people get on the train and also building a perennial community.

- Establish a training path for teams and people to start with Open Source, with a focus on domain matter experts and statisticians
- Propose events where the ESS OSS community could gather

Process

Governance

- Build the governance model: Central, top-down ESS governance VS distributed and bottom up organization. For example: GitHub Guide (<https://opensource.guide/leadership-and-governance/>)
- Describe the role of Eurostat
- Describe the role of NSI's including non-ESS NSIs
- Describe the role of UNECE
- Describe the role of other actors (including ONAs or private sector businesses)

Resource management

- How to manage a transparent management of resources in a development team
- How to handle financing outside of grants
- How to manage transparent prioritization

Funding, grants

- How to organize grants for Open Source projects?
- How to resource/finance top-down governance?
- Should there be user support, and if so, how should it be financed?

Product management

Open Source projects could benefit from opening their management, which may include product thinking as any other software product.

- Describe what should be an open project management for ESS OSS projects (including feature roadmap, product vision, etc.)
- Provide a guide to handling user feedback and support
- Product lifecycle (e.g. handling legacy API and feature set)

Peer review process

We propose that some kind of peer review process should be part of the ESS-OSS in order to build trust especially regarding statistical packages.

- Establish a peer review process where applicable. Should be considered as mandatory for Statistical Methods. See (<https://ropensci.org/software-review/>)
- Make validation/certification visible and fun! "Gamification" using GitHub badges f.ex. (<https://img.shields.io/badge/ESS--OSS-Validated!-success>, <https://img.shields.io/badge/ESS--OSS-Assessing-blueviolet>)
- Establish other quality related guidelines

Documentation

- Make it explicit that english is the mandatory language, also clearly define how to handle multilingual documentation
- Define general requirements for documentation (language, running the tests, software deployment)
- Highlight differences in documenting different types of projects (e.g. a statistical methods implementation as an R package VS a Javascript webapp)

Types and specificities

- If necessary, provide distinction between types of projects (e.g. do we need the same guidelines for Onyxia, PxWeb and sdcTable?)
- Provide a classification of Open Source assets (Platforms and utilities VS Applications and statistical libraries and methods)

Techniques

Foundation

- Explain what is needed to form an open technical backbone for ESS-OSS projects (open languages, open models, open dependencies)
- It's critical that source code is easily available to see for everyone. (eg. one should not need to have an account to inspect the open source code)
- Define acceptable source code management tools (Is Git the only solution?)
- Propose a list of possible Open Source project platforms (Only GitHub? Or any platform?)

Repository structure

- Treat your public code repository with as much love as your homepage. A tidy repository will make it easier to attract people to your projects, but it will also be easier for potential collaborators to pick up and join projects. This include an easy to find documentation, product descriptions etc.
- Consider the structure of the project, and arrange your repository accordingly, depending on the project type. Examples could be Monorepo (<https://en.wikipedia.org/wiki/Monorepo>) or Poly Repo/Multi-repo if you have loose coupling between main project and modules and/or utilities.

Handling changes

ESS OSS projects should handle evolution with care and keep users from dealing with breaking changes.

- Highlight the principles for handling changes in code
- Illustrate the impact of breaking changes in project
- Provide methods to guard against breaking changes (API design, test suites)
- Describe how to provide reproducible results when it makes sense (e.g. a new machine learning technique provided via a Python module)

Accepting code from contributors

- Make sure changes can be submitted to the project using the repositories built in mechanisms
- Make sure there is a proper review policy in place to review code contributions
- Have clear guidelines and rules for the project on code contribution and review process
- Explicit code of conduct and contributing guide
(https://github.com/tidyverse/dplyr/blob/main/.github/CODE_OF_CONDUCT.md,
<https://dplyr.tidyverse.org/CONTRIBUTING.html>)

Building and distributing

- Project should have a clear versioning strategy (branching, features, documentation, releases)
- Project should be easy to build and run
- Project should provide binary builds of the projects (available for download)
- Project should be cross platform (if applicable)
- Binaries should be distributed using well known distribution channels depending on the project (f.ex. CRAN, MavenCentral, PyPi, Docker HUB or similar).
- Projects should have automated builds using CI/CD tools that run tests and quality measurements of the code.
- Consider defaulting to a permissive license instead of a “Copyleft” license, as it gives you more freedom. Regardless it should be decided to have one default permissive license (Apache 2.0/MIT) and “Copyleft” license (EUPL) (See <https://joinup.ec.europa.eu/collection/eupl/solution/joinup-licensing-assistant/jla-find-and-compare-software-licenses>)

Architectural principles

These principles are proposals, and are listed in no particular order. They should be strongly considered, as they represent the best practices used in the IT industry.

- Consider modularising software for different modalities; Separate the UI from the business logic of the software and make sure the functionality (of the application/module/library) can be instrumented through code.
- All services should be secure by design in that they will perform the required functionality but be resistant to misuse or exposing data to unauthorized parties whilst supporting operational management and diagnostics.
(<https://www.ncsc.gov.uk/collection/cyber-security-design-principles>,
<https://www.oreilly.com/content/dont-build-death-star-security/>)
- Use principles like “API first” when building products
(<https://swagger.io/resources/articles/adopting-an-api-first-approach/>)
- Strive for simplicity and robustness in the code (readable, testable, extendable)
- Use **Domain Driven Design** principles when designing services and components.
(<https://martinfowler.com/bliki/DomainDrivenDesign.html>)

- Create user interfaces that follow accessibility guidelines, and requirements.
(<https://www.w3.org/standards/webdesign/accessibility>)
- Follow the **SOLID** principles when designing software where applicable
(<https://en.wikipedia.org/wiki/SOLID>)
- Make sure user interfaces can be multilingual and easy to extend to more languages
(<https://unece.org/fileadmin/DAM/stats/publications/BuildingMultilingualApplications.pdf>)