eosc | Infrastructures for Quality Research Software Task Force

Research Software Lifecycle

Deliverable 1 of the Subgroup 1 “On the Software Lifecycle” of the Task Force on Infrastructure for Quality Research Software

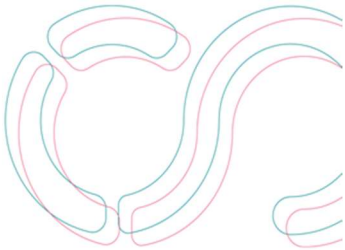
Contributors

- Guy Courbebaisse, CGE and INSA Lyon - CREATIS Lab., France, https://orcid.org/0000-0001-6181-2000
- Bernd Flemisch, University of Stuttgart, Institute for Modelling Hydraulic and Environmental Systems, Pfaffenwaldring 61, 70569 Stuttgart, Germany, https://orcid.org/0000-0001-8188-620X
- Kay Graf, Friedrich-Alexander-Universität Erlangen-Nürnberg, Erlangen Centre for Astroparticle Physics, Nikolaus-Fiebiger-Str. 2, 91058 Erlangen, Germany, https://orcid.org/0000-0002-1921-5568
- Uwe Konrad, Helmholtz-Zentrum Dresden-Rossendorf, Germany, https://orcid.org/0000-0001-8167-9411
- Jason Maassen, Netherlands eScience Center, Vrije Universiteit Amsterdam, the Netherlands https://orcid.org/0000-0002-8172-4865
- Raphael Ritz, Max Planck Computing and Data Facility, Giessenbachstr. 2, D-85748 Garching, Germany, https://orcid.org/0000-0003-4615-6804

Table of Contents

EOSC Association AISBL

Rue du Luxembourg 3, BE-1000 Brussels, Belgium
+32 2 537 73 18 | info@eosc.eu | www.eosc.eu
Reg. number: 0755 723 931 | VAT number: BE0755 723 931



Abstract

Software developed in the process of doing research is receiving increased attention. It is now more and more often considered a genuine research output next to scientific articles and research data publications. Based on representative user stories we identify and characterize the different phases and stages that the research software development process can go through thereby defining the "Research Software Lifecycle". Different approaches to software development such as product-, project- or platform-orientation are also outlined. We close with recommendations on EOSC infrastructure components needed to support the identified processes and platforms.

Introduction

This document constitutes Deliverable 1 of the SubGroup 1 "On the Software Lifecycle" of the EOSC Task Force "Infrastructure for quality research software"[1]. In order to reach the goals of the Task Force, it is mandatory to achieve a common understanding on the current processes in research software engineering, particularly the research software lifecycle. The aim of the present document is to illustrate this lifecycle and how its instantiations for particular software projects are influenced by varying developer groups and their intentions. We focus on the software developed and maintained in research which can be embedded into open research infrastructures such as EOSC. We point out possible connections to the existing and planned EOSC infrastructure to support the research software lifecycle. In general, research software

[1] Charter of the EOSC Task Force on Infrastructures for Quality Research Software

should be as open as possible and reflect the FAIR principles in its design, especially the Interoperability and Reusability.[2]

While various software development lifecycle models exist[3], the specifics of research software[4] should be discussed. Various considerations associated with software engineering processes in research software have been put forward. This document takes inspiration from these sources, e.g. the DLR Software Engineering Guidelines[5] provides a useful definition of "application classes", ranging from class 0 (personal use, small scope) to class 4 (mission critical). Depending on the class, stricter guidelines regarding software engineering, documentation, testing, change management, etc. apply. The Turing Way[6] and CODE REFINERY[7] contain extensive information on best practices for research software development and reproducible research. A characterization of research software projects and their transitions between different application classes have been analyzed for NSF-funded software.[8] Four different organizational configurations have been identified: author group, laboratory, tool group and peer production.

The rest of this document is organized as follows. We first describe different user stories encountered in research software development, ranging from an individual researcher creating software for personal use up to a possibly large scientific community developing an well-established service. This is followed by elaborating on different approaches in software development driven by different orientations of developers or maintainers. This sets the stage for presenting the research software lifecycle, where we provide recommendations for facilitating and implementing the individual steps. We conclude by drawing explicit connections from the research software lifecycle to the EOSC.

User stories

In this chapter, user stories are introduced to set the stage for the later discussion of the software and research lifecycle.

[2] Barker, M., Chue Hong, N.P., Katz, D.S. et al. Introducing the FAIR Principles for research software. Sci Data 9, 622 (2022). https://doi.org/10.1038/s41597-022-01710-x
[3] Khan, N.A. (2021). Research on Various Software Development Lifecycle Models. In: Arai, K., Kapoor, S., Bhatia, R. (eds) Proceedings of the Future Technologies Conference (FTC) 2020, Volume 3. FTC 2020. Advances in Intelligent Systems and Computing, vol 1290. Springer, Cham. https://doi.org/10.1007/978-3-030-63092-8_24
[4] Gruenpeter, M., Katz, D.S., Lamprecht, A. et al. Defining Research Software: a controversial discussion (Version 1). Zenodo. https://doi.org/10.5281/zenodo.5504016
[5] DLR Software Engineering Guidelines, https://doi.org/10.5281/zenodo.1344612
[6] The Turing Way, https://the-turing-way.netlify.app/welcome
[7] Training and e-Infrastructure for Research Software Development, https://coderefinery.org
[8] The Transition Project, https://hannahcohoon.com/transition

1. Individual creating research software for own use (e.g. a PhD student)

   Based on a research question, software is created by a single person with the specific aim of answering the research question and producing research output (paper, dataset, etc). The planning and development process is kept light. Basic software engineering practices (version control, basic documentation, basic testing) should be applied. More advanced software engineering practices like issue tracking, code style, test coverage, Continuous Integration and Delivery (CI/CD), code reviews, code quality checkers, etc. are often not used. Once the desired result is obtained, all research outputs should be published (paper, dataset, workflow, software, etc.) to ensure reproducibility of the results. After publication, the software component is often not actively maintained, although in some cases it may serve as a basis for the next cycle (i.e., based on feedback which triggers further research questions). This type of software can be picked up by other single persons or teams, where a transfer of intellectual property and access rights needs to be ensured. A prerequisite for such a transfer is to equip the software tool with a license, preferably an open-source one[9].

2. A research team creating an application or workflow for use within the team

   Research software is created by a team to answer a series of research questions (often as part of a larger research project). To enable development by multiple team members and ensure longer term maintainability, both basic software engineering practices (version control, basic documentation, basic testing) and a selection of more advanced software engineering practices (issue tracking, semantic versioning, code style, test coverage, CI/CD, code reviews, code quality checkers, etc.) should be used. Proper planning of development is needed to ensure regular releases of the software. These releases are used to answer research questions and produce research outputs (papers, datasets, etc). To ensure reproducibility of results, versioned releases should be archived, and properly cited in papers/dataset/etc. After publication, the development cycle continues to answer further research questions and provide improvements/fixes to the existing code.
   Examples for this category are software solutions to simulate, operate and analyze the outputs of large scientific experiments or - more generally - measurement devices and their outputs.

3. A team / community developing (possibly broadly applicable) open source research software

[9] Cf. § 47 of the European Parliament discharge report 2018
https://www.europarl.europa.eu/doceo/document/TA-9-2020-0088_EN.pdf

Software is created by a team (possibly distributed over multiple organizations) to answer a broad range of research questions. Different team members may have different objectives and/or may represent a community of users. External users may depend on that software with or without directly contributing to its development. Both basic and advanced software engineering techniques should be used to ensure a smooth development process, quality and long term maintainability. Proper development planning should be used to organize the team members and ensure regular releases. This development cycle is often not directly driven by a (single) research cycle. Instead, community feedback (e.g. issue tracking) is used to drive maintenance and development decisions.

Examples - by far not exhaustive - in different communities are: Astropy[10] (astronomy); ESMValTool[11] (Earth sciences); VASP[12], FHI-aims[13] (material research); Gromacs[14] (molecular dynamics); Neuron[15], Nest[16], Genesis (Neuroinformatics); root (particle physics); Scikit-Learn (machine learning)[17]; Sat4J (boolean satisfaction and optimization)[18]. <any more examples can be found at, e.g., the Research Software Directory[19].

4. A team or community creating a research service

A service platform is a set of software components which is used to provide services for a large number of users, most of whom make use of these offerings via the Internet (e.g. cloud services). These software components can run at one place or be distributed on virtualized hardware. They are mostly open source, but the services offered are not necessarily free. It's also not uncommon to have closed source services which provide access to open data.
The goal in developing and operating such a platform is to create a sustainable and scalable set of services for a defined target group. Added value compared to local software solutions arises, among other things, from the fact that data storage, computing capacities and communication options are offered in addition to core functions such as modeling, data analysis, project management or software

[10] Astropy, https://www.astropy.org/
[11] ESMValTool, https://www.esmvaltool.org
[12] The Vienna Ab initio Simulation Package (VASP), https://www.vasp.at/
[13] FHI-aims, https://fhi-aims.org/
[14] Gromacs, https://www.gromacs.org/
[15] NEURON, https://neuron.yale.edu/neuron/
[16] NEST simulator, https://www.nest-simulator.org/
[17] https://scikit-learn.org/stable/
[18] https://www.sat4j.org/
[19] Research Software Directory, https://www.research-software-directory.org/software

development. The need for development, deployment and continuous operation (24x7) as well as flexible scalability requires special software development methods such as DevOps[20] and CI/CD[21]. The sustainability approach also requires strict development and quality guidelines, clear licenses, an open development environment, a clear recognition to contributors as well as continuous funding or a viable business model.
Examples of such platforms in research are, e.g., D4Science (a solution supporting the development of Virtual Research Environments), HIFIS[22] (generic digital service platform), VIP[23] (medical imaging platform), VISA[24] (infrastructure for analysis) or VISPA[25] (physics data analysis platform). Such services or platforms are more and more collected in the EOSC portal[26].

While user stories 1 and 2 are usually managed by individual persons or groups of persons with a common research interest and goal, 3 and 4 need a stricter organizational form, e.g. forming a management or by effectively applying the role of a CTO, hired or taken by the original author/PI. The persons and groups are supported in their development and maintenance goals by different foundations or organizations like the Apache Software Foundation[27], the eScience Centre of the Netherlands[28], the HEP Software Foundation[29], or Software Heritage[30]. If a software development project leaves the domain of research - e.g. by its complexity, industry involvement or global application - it is not covered by the use cases addressed here.

It is important to notice that software projects may evolve significantly over time, and can hence move from one category to another. It is often the case that what has become broadly used research software today, either maintained by a large community, or by a small number of core developers, actually started at the beginning as an individual effort to address a research question in a small team or community (see for example the evolution of Sat4J[31]).
Recommendations and/or requirements on research software must take into account this evolutionary nature of research software.

[20] DevOps - Wikipedia, https://en.wikipedia.org/wiki/DevOps
[21] CI/CD - Wikipedia, https://en.wikipedia.org/wiki/CI/CD
[22] Helmholtz Federated IT Services (HIFIS), www.hifis.net
[23] Virtual Imaging Platform (VIP) - EOSC Marketplace, https://marketplace.eosc-portal.eu/services/virtual-imaging-platform
[24] Virtual Infrastructure for Scientific Analysis (VISA), https://marketplace.eosc-portal.eu/services/visa-virtual-infrastructure-for-scientific-analysis
[25] Visual Physics Analysis (VISPA), https://vispa.physik.rwth-aachen.de
[26] EOSC Portal - Marketplace Processing & Analysis , https://marketplace.eosc-portal.eu/services/c/processing-analysis
[27] The Apache Software Foundation, https://www.apache.org/
[28] Netherlands eScience Centre, https://www.esciencecenter.nl/
[29] The HEP Software Foundation (HSF), https://hepsoftwarefoundation.org/
[30] Software Heritage, https://www.softwareheritage.org/
[31] https://www.sat4j.org/allabout.php

Approaches in Software Development - Products, Projects or Platforms

Different approaches in the development of software - for products, projects and platforms are pursued in science and industry. In the following we describe the main aspects of those three approaches mostly adopted by the group of developers or maintainers that need to be taken into account for the full software life cycle as described in the next section.

Product Orientation (a typical case in Industry)

- Roles and responsibilities are more separated between different departments and people, e.g.: marketing, requirements engineering, product management, project management, development, quality assurance including professional software engineering, sales, after sales, service and training
- Project planning and controlling (implementation, milestones, development models, goals and targets in terms of time effort and costs) usually high, roadmap and rollout of new products and releases planned well in advance
- Cost / benefit orientation and -calculation throughout the development cycle
- Quality assurance in different phases (development, product documentation, field test, pilot projects, release)
- Visibility and recognition is high but also strong pressure with respect to time, budget and success
- Often closed community and mostly closed source, prominent exceptions being e.g. Kubernetes[32], Docker[33] and TensorFlow[34]

Project Orientation (a typical case in Science):

- Usually starts with an idea to solve a problem (in most cases a challenge for the own team or company), sometimes based on a vision, requirements evolving over time
- Less clear goals and targets in terms of time, effort and costs
- Team and budget changing over time
- Little visibility and recognition but sometimes low pressure with respect to milestones, budget and success
- In research often embedded in an open community (strong cooperation), mostly open source

The four user stories mentioned above show examples of project-based SW development in research, which in many cases is not yet implemented in a sustainable way. There is also a

[32] Kubernetes (K8s), https://kubernetes.io/
[33] Docker, https://www.docker.com/
[34] TensorFlow, https://www.tensorflow.org/

project oriented "internal SW" development in industry. It is not that much focused on marketing and sales, but still cost and milestone oriented.

Platform Orientation (in Industry and Science)

- Cost / Benefit orientation: calculated throughout the cycle, with an overall aim of increasing the number of users and developers (rather than directly maximizing monetisation)
- Project Planning and Control: usually high with a clear roadmap for releases (to allow coordination with developers creating products based on the platform)
- Open Community and often open source
- Usually agile development techniques are employed
- Highly automated test and deployment process
- Takes care of data and various services around the core software
- Provides a certain level of sustainability for platform users, also data privacy is an issue
- Develops support structures and resources (e.g., ticket system, help desk, documentation/wiki)

By introducing "application classes" (see Introduction), the requirements of these approaches can be taken into account in early phases of the Software Life Cycle in order to achieve a more professional software development in research. This is supported by the definition of concrete minimum standards and measures for the application classes in the guidelines and policies for software development for an increasing number of research institutions.

Research Software Lifecycle

In Fig. 1, a graphical representation of the research software lifecycle is presented. It represents one loop cycle which usually is repeated over the lifetime of a research software project. During a specific instantiation of one cycle, not necessarily all of the depicted six steps are taken into account. We remark that the figure is necessarily not exhaustive due to the complexity and individuality of the different approaches adopted by different persons, groups and communities. The different steps and interactions of the graph are described below.

More recently, software management plans[35] (SMPs) have been proposed for managing software lifecycles. Like data management plans (DMPs), SMPs are documents describing what the aim of the software is, how the research software will be managed, both during a project's lifetime and after a project has ended. By making this information explicit, SMPs help to set up a proper development cycle for research software, provide insight into the necessary resources

[35] Introduction to Software Management Plans, https://github.com/softwaresaved/introduction-to-software-management-plans

during development, and ensure the long-term accessibility of the software is properly considered.

As there currently is no universally accepted SMP template, several organizations have created their own[36], as well as checklist[37] and recommendations[38] [39] on how to create SMPs. Important aspects to be considered when setting up a SMP are to classify the software, choose the necessary stages from Figure 1 as well as the appropriate license, the organization of IP handling and the transition between different software classes. As those steps are usually not plannable a priori, an SMP often evolves during a project.

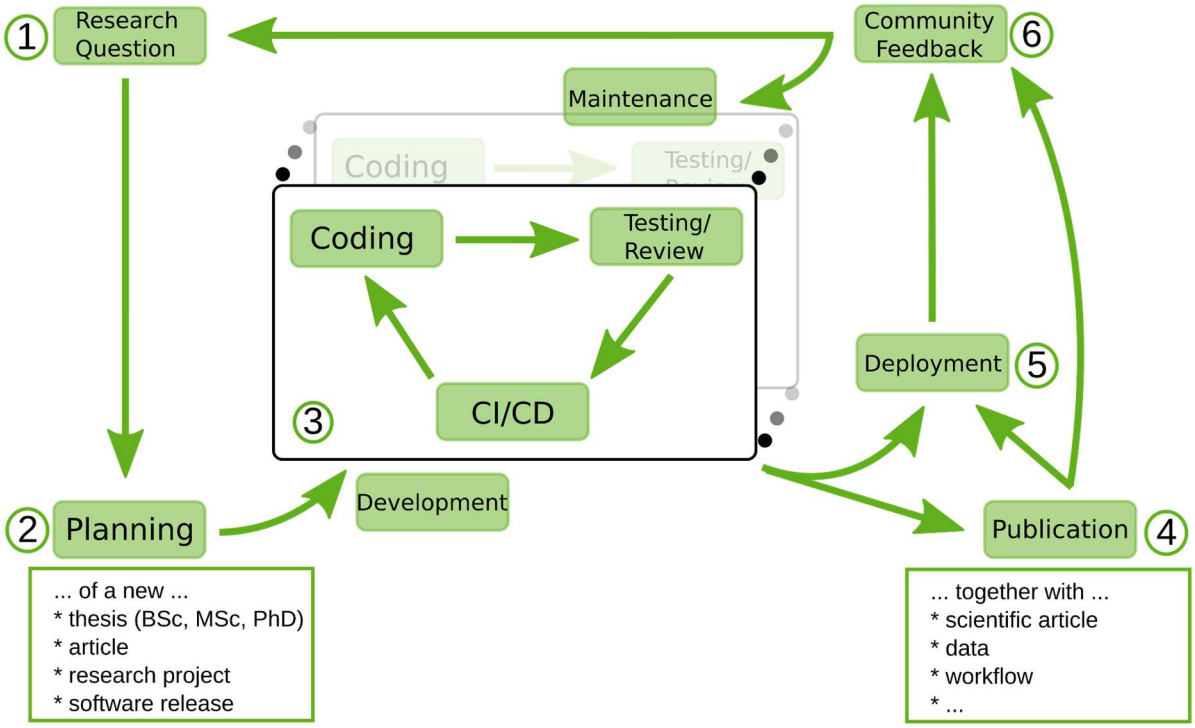


Figure 1: Graphical representation of the research software lifecycle

1 Initialization

A research question, a new service or feature request (e.g. data service, workflow automation) or tenders from funders (e.g. from EU) initiate the planning phase.

[36] Netherlands eScience Center Software Sustainability Protocol, https://doi.org/10.5281/zenodo.1451751
[37] Checklist for a Software Management Plan, https://zenodo.org/record/2159713
[38] Writing and using a software management plan, https://www.software.ac.uk/resources/guides/software-management-plans
[39] Practical guide to software management plans, https://doi.org/10.5281/zenodo.7248877

2 Planning

Planning research software development usually happens in the context of a student thesis (BSc, MSc, PhD), or for a larger research project including preparations of concrete scientific articles. Also new releases for longer running research software projects often trigger a "start over".

The planning stage can be facilitated by or include a requirements analysis (for all levels), a value proposition (for software products) or a market or community survey in order to understand the audience (for software products or platforms). Also a roadmap for the initial development and later stages can be instrumental.

3 Implementation

Depending on the context, each cycle illustrates the typical workflow for the implementation of, e.g. a new feature as part of software development or a bugfix as part of software maintenance.[40]

Especially in bigger projects, all stages within one cycle might be very elaborated, while several cycles coexist at the same time corresponding to the parallel implementation of new features or bug fixes. In single-person projects such as student theses, implementation is naturally performed in a more sequential manner. Reviews by third parties might be missing and, typically, no CI/CD services or frameworks are employed.
The implementation stage can be facilitated, amongst others, by version control and version control management systems (incl. release management); issue tracking and code review services; testing and CI/CD services and infrastructure.
The practices and facilitating tools may differ for different stages the software is in, see e.g., the application classes in the DLR Software Engineering Guidelines.[41]

4 Publication

While software is worth being published per se, currently in research, the publication of research software usually accompanies the completion and publication of a document such as a student thesis, scientific article or project report. Further ingredients such as input and result data, containers, notebooks, or workflow descriptions should ideally yield an integrated research compendium/object. Research software should in itself be citable and should be referenced in the document. Research software engineers (RSEs) often play an important role in developing the software, without necessarily becoming co-author of the resulting papers or datasets. It is therefore important to give them recognition for their contributions through software citation. In

[40] Topics related to research software quality, such as code styles, testing schemes and vulnerability checks are addressed in detail by this task force, in the *Ensure Software Quality* deliverable
[41] DLR Software Engineering Guidelines, https://doi.org/10.5281/zenodo.1344612

addition to publishing the software itself, various dedicated journals exist that offer the service of publishing research software papers[42].

The publication/deployment stage of research software can be facilitated by, e.g., (i) CD frameworks, (ii) publication services, (iii) archival services or (iv) training and support services, (v) research software directories[43].
A link between research software and other research objects can be established within research software directories, requiring proper IDs, e.g. by minting DOIs via Zenodo[44] or the Software Heritage[45] IDs.

At the latest during the publication preparation - but preferably at an as early stage as possible - metadata should be added to the software. Several standards are in use and under harmonization via several EOSC-related groups[46], currently used standards are the Citation File Format[47] and CodeMeta[48].
The findability can be increased by adding a citation to software and referencing in accompanying articles.

5 Deployment and Platform Integration

Software is not only provided as source code and executable, but also via direct use in platforms, see user story 4. This targets software as a service that adds some specific issues compared to an on-premise software publisher, e.g. interfaces to authentication and authorization, data, monitoring, scalability, accounting, continuous maintenance.
Such integration can be started by the software developer by providing software and data within Jupyter notebooks, containers or virtual machines as a first or intermediate step towards a software as a service.

[42] The Journal of Open Source Software, https://joss.theoj.org/
The Journal of Open Research Software (JORS), https://openresearchsoftware.metajnl.com/
Software X, https://www.sciencedirect.com/journal/softwarex
[43] Research Software Directory, https://github.com/research-software-directory
Research Software Directory, https://research-software-directory.org/
Research Software Directory, https://helmholtz.software/
[44] Zenodo, https://zenodo.org
[45] Software Heritage, https://www.softwareheritage.org/
[46] EOSC Task Force FAIR Metrics and Data Quality
https://www.eosc.eu/sites/default/files/tfcharters/eosca_tffairmetricsanddataquality_draftcharter_20210614.pdf
EOSC Task Force Semantic Interoperability
https://www.eosc.eu/sites/default/files/tfcharters/eosca_tfsemanticinteroperability_draftcharter_20210614.pdf
[47] Citation File Format, https://citation-file-format.github.io/
[48] The CodeMeta Project https://codemeta.github.io/

Here, the deployment and integration process of the specific software is meant, the platform itself is a necessary ingredient but often not in the focus of the software development lifecycle. Both - the integration and the platforms - are a necessary ingredient to the EOSC integration.
Open source publication should be preferred before integration to a platform but other options are also possible, keeping in mind that for software as a service a valid business model may be needed (e.g. there are models of having a community edition for free and a professional edition with a support fee).

6 Community Feedback

The form, amount and timeliness of community feedback to research software can vary substantially depending on the size, maturity and targeted application range and audience. In addition to the dissemination of the scientific articles describing (and advertising) the software, presentations at conferences and other meetings potentially help to initiate and grow a user community.
The community feedback stage can be facilitated by appropriate communication channels (helpdesk, issue tracker, email lists, chat applications …) or even an established means to let the community contribute to or assess the quality of the software in general.

Continuation or Termination

Feedback and reuse can either lead to new research questions initiating another full cycle (from 1 to 6 above), trigger smaller implementation cycles in the context of software maintenance or termination of the development/service.
Particularly in this stage, the user stories can be changed and also the application classes can be newly defined.

Research Software Lifecycle and the EOSC Infrastructure

An **Infrastructure for Research Software in EOSC** should support all user stories and the research software lifecycle as described above in consistency with the EOSC rules of participation[49]. To allow broad support for the software and services, infrastructure components for aggregation, communication, development and archival are required.
There are two categories of actions as outcome of the software lifecycle needs.
One is in the area of governance, outreach and support:

- Communicate the added value of the use and integration of research software and software related services in EOSC

[49] EOSC Rules of Participation Compliance Monitoring Task Force, https://www.eosc.eu/advisory-groups/rules-participation-compliance-monitoring

- Provide support on the transition of software projects and services into the EOSC ecosystem
- Support the scientific communities on the transition between the different levels of software classes and their sustainability, especially respecting time-limited funding schemes
- Support teaching and training activities[50] on software development and engineering including coordinating efforts across Europe; not only for practitioners but also following the "train the trainer" approach to expand the reach

The other for technical implementations:

- Provide and support open, sustainable software development platforms and the federation of established platforms to foster broad collaboration and cross-fertilisation
- Provide a support infrastructure for ensuring and improving research software quality, including the application of appropriate metadata, licenses, and quality metrics
- Provide software directories, repositories, registries and archives - in the sense of Archives, Publishers, and Aggregators of the SIRS Report[51]
- Integrate or link community, national, and global infrastructures (e.g. GitHub[52], OpenAire[53], Software Heritage[54], Zenodo[55]) in the EOSC ecosystem guaranteeing the validity of identification and access control used
- Provide collaborative support tools for interactions between developers and the user communities, preferably well integrated into the software service and development platform

Research software should be considered as a first class citizen of open science, the resulting requirements must be considered in the implementation of a EOSC infrastructure for research software.

[50] See also the charter of the EOSC Task Force on Data stewardship curricula and career paths
[51] European Commission, Directorate-General for Research and Innovation, Scholarly infrastructures for research software : report from the EOSC Executive Board Working Group (WG) Architecture Task Force (TF) SIRS, Publications Office, 2020, https://data.europa.eu/doi/10.2777/28598
[52] Git Hub, https://github.com/
[53] OpenAIRE, https://openaire.eu
[54] Software Heritage, https://www.softwareheritage.org/
[55] Zenodo, https://zenodo.org