**Project acronym:** EVERSE
**Project full title:** European Virtual Institute for Research Software Excellence
**Call identifier:** HORIZON-INFRA-2023-EOSC-01
**Grant Agreement Number:** 101129744
**DOI:** 10.5281/zenodo.15856368

# EVERSE
# Reference Framework for Research Software Quality

*Version 2.1*

Author(s):

Faruk Diblen (NLeSC)
Giacomo Peru (UEDIN)

Funded by
the European Union

# TABLE OF CONTENTS

**1 Introduction** **3**

1.1 Scope and goals . . . 3

1.2 Definitions . . . 3

1.2.1 Research Software . . . 3

1.2.2 Research Software Quality . . . 4

**2 EVERSE Reference Framework** **4**

2.1 Technical Dimensions of Software Quality . . . 5

2.1.1 ISO Software Quality Dimensions . . . 6

2.1.2 Research Software Quality Indicators . . . 9

2.2 FAIRness . . . 9

2.2.1 Good enough practices . . . 11

2.2.2 Examples of tools relevant for FAIR software . . . 11

2.3 Open Source Software . . . 12

2.3.1 Good enough practices . . . 13

2.3.2 Examples of tools relevant for open source software . . . 13

2.4 Sustainability . . . 14

2.4.1 Good enough practices . . . 15

2.4.2 Examples of tools relevant to Software Sustainability . . . 15

**3 Four Views** **17**

3.1 Three-Tiers View . . . 17

Analysis Code (Tier 1) . . . 18

Prototype Tools (Tier 2) . . . 18

Research Software Infrastructure (Tier 3) . . . 19

Tier Relationships and Progression . . . 19

Quality Implications Across Tiers . . . 19

Context and History . . . 19

Alignment with Software Management Plans . . . 20

3.2 Software Lifecycle View . . . 20

Funded by
the European Union

Lifecycle Patterns by Tier . . . . . 20
Alternative RSQKit Lifecycle Model . . . . . 21
Quality Implications Across the Lifecycle . . . . . 23
Lifecycle and Quality Assessment . . . . . 23
3.3 Personas View . . . . . 23
Core Personas in Research Software . . . . . 23
Persona-Specific Quality Priorities . . . . . 24
Role Specialization and Software Tiers . . . . . 24
Persona Interactions and Collaboration . . . . . 25
Practical Applications . . . . . 25
Integration with Other Views . . . . . 25
3.4 Science Clusters View . . . . . 26
Additional Cluster-Specific Roles . . . . . 26
Quality Dimension Prioritization . . . . . 27
Integration with EVERSE Framework . . . . . 27

**4 Conclusions** **27**

Funded by
the European Union

# 1 Introduction

## 1.1 Scope and goals

The goal of this document is to provide an overview of the considerations and perspectives around research software quality being taken by the EVERSE project and to act as a guide to internal activity as well as a statement to those outside of the project of the approach being taken. The main scope is to connect various perspectives to software quality, excellence, and good practices.

The Framework is structured around four main pillars, detailed in Section 2: Technical aspects (Section 2.1), FAIR Software (Section 2.2), Open Source Software (Section 2.3) and Sustainability (Section 2.4).

To address the diverse needs of the research software community, the Framework offers four distinct views, which are detailed in Section 3:

- **Research Software Tier**: the tier of research software determines which aspects of the quality framework are most relevant and should be prioritised.
- **Research Software Lifecycle**: depending on the step in the research software lifecycle, different parts of the model are highlighted.
- **Persona**: depending on the user role, different aspects of research software quality (e.g. policy, funding, steps to improve code, training), etc. might be more relevant.
- **Science Clusters/Communities**: different existing practices/standards/services are made visible, to better reflect the particular domain-specific requirements.

The Framework's principles are adaptable to different software types, acknowledging that requirements vary significantly between simple scripts and complex applications. A simple script should be treated differently than a complex multi-user software product. As such, all these views may change depending on the software type.

## 1.2 Definitions

### 1.2.1 Research Software

"Research software" is commonly used to refer to software used and/or generated in a research context, including and not limited to scientific, non-scientific, commercial, academic and non-academic research. Our definition should refer to objects to which the FAIR principles should apply to. Furthermore, software is an important component when it comes to reproducibility, where a different team needs to use the same research.

This definition distinguishes research software from the broader category of "software in research" - general-purpose tools, libraries, or infrastructure components that support

Funded by the European Union

research activities but were not developed specifically for research purposes. Recognising that software exists on a spectrum from research-specific to general-purpose tools, the Reference Framework focuses on software where quality directly impacts research outcomes and scientific reproducibility.

For additional context and examples, see: Defining Research Software: a controversial discussion, https://zenodo.org/records/5504016.

### 1.2.2 Research Software Quality

We define "research software quality" as the degree to which software supports reliable, efficient, maintainable, and trustworthy research. Quality encompasses multiple dimensions including correctness, performance, maintainability, usability, robustness, and reproducibility, etc.

Quality research software enables researchers to trust, share, and build upon computational work, making science transparent and verifiable. FAIR software principles (Findable, Accessible, Interoperable, Reusable) represent a crucial subset of quality, ensuring software can be discovered, understood, and exercised by others whilst supporting long-term reproducibility and research integrity.

A key aspect of quality is reproducibility - the ability for others (or your future self) to run the software and obtain identical results using the same inputs and environment. This requires consistent and deterministic behaviour, comprehensive version control and documentation, automated testing, environment management, and clear, accessible code.

The EVERSE Reference Framework addresses quality through formal dimensions and indicators that assess and improve these essential aspects of research software excellence.

# 2 EVERSE Reference Framework

Research software quality assessment combines insights from the two worlds of software engineering and research. Because of software's fundamental characteristics, many quality aspects centre on technical dimensions that software engineers and developers apply universally, regardless of whether they work in academia or industry. However, research software may place different emphasis on these dimensions compared to typical industry applications. Additionally, research contexts introduce specific quality requirements that extend beyond traditional software engineering concerns.

In EVERSE, we structure the discussion around research software quality with three main concepts: Research Quality Dimensions, Research Software Quality Indicators and Research Software quality tools. Research Software Quality Dimensions represent common criteria relevant for assessing software quality, at a conceptual level (e.g., documentation, performance, etc.). At a lower level of granularity, a software quality indicator represents a specific aspect of software that can be measured (e.g., FAIRness,

Funded by
the European Union

test coverage, documentation coverage, etc.). Finally, a research software quality tool is the instrument that measures a software quality indicator against specific criteria or, alternatively, helps improve in one particular aspect (such as linters). Figure 1 shows an example of this division with an example: the "Maintainability" dimension, as defined in ISO/IEC 25010 may be associated with an indicator to check whether continuous integration tests exist in a source code repository (e.g., as defined by the OpenSSF community). Then an assessment tool like the OpenSSF Scorecard may implement the indicator by looking at whether the project runs tests before pull requests are merged in GitHub. However, a different quality assessment tool may check the same indicator against GitLab repositories (not supported by OpenSSF Scorecard).

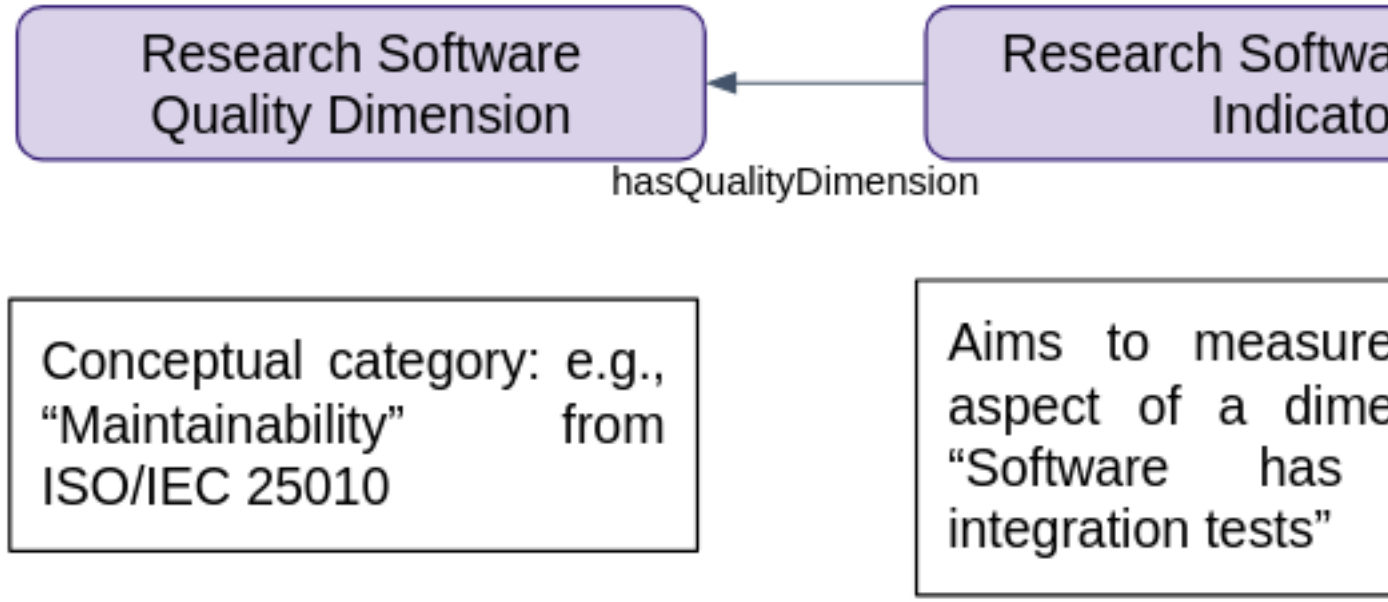


Figure 1: Figure 1: Main research software quality concepts, together with an example

The EVERSE Framework encompasses twelve main dimensions of research software quality, derived from established standards and research software best practices. These consist of nine technical dimensions based on ISO/IEC standards, plus FAIRness, Open Source Software, and Sustainability dimensions that reflect the unique requirements of research software. Each dimension is described in the sections that follow.

## 2.1 Technical Dimensions of Software Quality

Software quality has been a subject of study by scientists, developers and engineers, with each community bringing different perspectives and developing various quality dimensions, best practices and recommendations. For example, best practices driven by developers like Open SSF include a specific section for RS quality involving automation, testing and project description and documentation. From an engineering perspective,

Funded by the European Union

ISO Software quality models have been proposed [1] describing dozens of software quality dimensions ranging from ease of use to security or technical performance. Key industry stakeholders like Microsoft have proposed software quality attributes affecting key aspects of their tools [2]. Recently, a need for transparency, long term preservation (e.g., RS management plans [3]) and FAIR adoption (i.e., FAIR4RS [4]) has highlighted the importance of high quality metadata when describing RS records. In this section we briefly overview the main software quality characteristics. In this document we follow the 9 top-level ISO Software Quality dimensions.

### 2.1.1 ISO Software Quality Dimensions

At the March 2025 EVERSE General Assembly, the several available options listed above were discussed leading to the decision to adopt the ISO/IEC 25010 [1] dimensions. These ISO quality dimensions are hierarchically defined, with 9 top-level categories that have a number of more fine-grained sub-dimensions each, for a total of 39. We describe below the 9 top-level dimensions that we will use to categorize research software quality practices, along with the sub-categories defined in ISO/IEC 25010:

**Primary ISO Quality dimensions:**

- **Compatibility** represents the degree to which a product, system or component can exchange information with other products, systems or components, and/or perform its required functions while sharing the same common environment and resources. This characteristic is composed of the following sub-characteristics:
  - Co-existence - Degree to which a product can perform its required functions efficiently while sharing a common environment and resources with other products, without detrimental impact on any other product.
  - Interoperability - Degree to which a system, product or component can exchange information with other products and mutually use the information that has been exchanged.
- **Flexibility** represents the degree to which a product can be adapted to changes in its requirements, contexts of use or system environment. This characteristic is composed of the following sub-characteristics:
  - Adaptability - Degree to which a product or system can effectively and efficiently be adapted for or transferred to different hardware, software or other operational or usage environments.
  - Scalability - Degree to which a product can handle growing or shrinking workloads or to adapt its capacity to handle variability.
  - Installability - Degree of effectiveness and efficiency with which a product or system can be successfully installed and/or uninstalled in a specified environment.

Funded by the European Union

  - Replaceability - Degree to which a product can replace another specified software product for the same purpose in the same environment.

- **Functional suitability** represents the degree to which a product or system provides functions that meet stated and implied needs when used under specified conditions. This characteristic is composed of the following sub-characteristics:
  - Functional completeness - Degree to which the set of functions covers all the specified tasks and intended users' objectives.
  - Functional correctness - Degree to which a product or system provides accurate results when used by intended users.
  - Functional appropriateness - Degree to which the functions facilitate the accomplishment of specified tasks and objectives.
- **Interaction capability** represents the degree to which a product or system can be interacted with by specified users to exchange information via the user interface to complete specific tasks in a variety of contexts of use. This characteristic is composed of the following sub-characteristics:
  - Appropriateness recognizability - Degree to which users can recognize whether a product or system is appropriate for their needs.
  - Learnability - Degree to which the functions of a product or system can be learnt to be used by specified users within a specified amount of time.
  - Operability - Degree to which a product or system has attributes that make it easy to operate and control.
  - User error protection - Degree to which a system prevents users against operation errors.
  - User engagement - Degree to which a user interface presents functions and information in an inviting and motivating manner encouraging continued interaction.
  - Inclusivity - Degree to which a product or system can be used by people of various backgrounds (such as people of various ages, abilities, cultures, ethnicities, languages, genders, economic situations, etc.).
  - User assistance - Degree to which a product can be used by people with the widest range of characteristics and capabilities to achieve specific goals in a specified context of use.
  - Self-descriptiveness - Degree to which a product presents appropriate information, where needed by the user, to make its capabilities and use immediately obvious to the user without excessive interactions with a product or other resources (such as user documentation, help desks or other users).
- **Maintainability** represents the degree of effectiveness and efficiency with which a product or system can be modified to improve it, correct issues or adapt it to changes in its environment, and in requirements. This characteristic is composed of the following sub-characteristics:

Funded by
the European Union

  - Modularity - Degree to which a system or computer program is composed of discrete components such that a change to one component has minimal impact on other components.
  - Reusability - Degree to which a product can be used as an asset in more than one system, or in building other assets.
  - Analysability - Degree of effectiveness and efficiency with which it is possible to assess the impact on a product or system of an intended change to one or more of its parts, to diagnose a product for deficiencies or causes of failures, or to identify parts to be modified.
  - Modifiability - Degree to which a product or system can be effectively and efficiently modified without introducing defects or degrading existing product quality.
  - Testability - Degree of effectiveness and efficiency with which test criteria can be established for a system, product or component and tests can be performed to determine whether those criteria have been met.

- **Performance efficiency** represents the degree to which a product performs its functions within specified time and throughput parameters and is efficient in the use of resources (such as CPU, memory, storage, network devices, energy, materials...) under specified conditions. This characteristic is composed of the following sub-characteristics:

  - Time behaviour - Degree to which the response time and throughput rates of a product or system, when performing its functions, meet requirements.
  - Resource utilization - Degree to which the amounts and types of resources used by a product or system, when performing its functions, meet requirements.
  - Capacity - Degree to which the maximum limits of a product or system parameter meet requirements.

- **Reliability** represents the degree to which a system, product or component performs specified functions under specified conditions for a specified period of time. This characteristic is composed of the following sub-characteristics:

  - Faultlessness - Degree to which a system, product or component performs specified functions without fault under normal operation.
  - Availability - Degree to which a system, product or component is operational and accessible when required for use.
  - Fault tolerance - Degree to which a system, product or component operates as intended despite the presence of hardware or software faults.
  - Recoverability - Degree to which, in the event of an interruption or a failure, a product or system can recover the data directly affected and re-establish the desired state of the system.

- **Safety** represents the degree to which a product operates under defined conditions to avoid a state in which human life, health, property, or the environment is

Funded by
the European Union

endangered. This characteristic is composed of the following sub-characteristics:

  - Operational constraint - Degree to which a product or system constrains its operation to within safe parameters or states when encountering operational hazard.
  - Risk identification - Degree to which a product can identify a course of events or operations that can expose life, property or environment to unacceptable risk.
  - Fail safe - Degree to which a product can automatically place itself in a safe operating mode, or to revert to a safe condition in the event of a failure.
  - Hazard warning - Degree to which a product or system provides warnings of unacceptable risks to operations or internal controls so that they can react in sufficient time to sustain safe operations.
  - Safe integration - Degree to which a product can maintain safety during and after integration with one or more components.

- **Security** represents the degree to which a product or system defends against malicious attacks and protects information and data so that persons or other products or systems have the degree of data access appropriate to their types and levels of authorisation. This characteristic is composed of the following sub-characteristics:

  - Confidentiality - Degree to which a product or system ensures that data are accessible only to those authorized to have access.
  - Integrity - Degree to which a system, product or component ensures that the state of its system and data are protected from unauthorized modification or deletion either by malicious action or computer error.
  - Non-repudiation - Degree to which actions or events can be proven to have taken place so that the events or actions cannot be repudiated later.
  - Accountability - Degree to which the actions of an entity can be traced uniquely to the entity.
  - Authenticity - Degree to which the identity of a subject or resource can be proved to be the one claimed.
  - Resistance - Degree to which the product or system sustains operations while under attack from a malicious actor.

### 2.1.2 Research Software Quality Indicators

The EVERSE indicators repository provides the canonical, up-to-date list of quality dimensions and their associated indicators: https://everse.software/indicators/.

## 2.2 FAIRness

The FAIR Guiding Principles were originally developed for scientific data management and stewardship, with the goal of improving the Findability, Accessibility, Interoperabil-

ity and Reusability of digital assets. While the original FAIR principles focused primarily on data, there has been growing recognition of the need to apply FAIR principles to research software as well.

The FAIR Principles for Research Software (FAIR4RS Principles)[1] adapt and extend the original FAIR principles to address the unique characteristics of software. Key considerations in applying FAIR to software include:

- Software's executable nature, compared to static datasets
- Software's composite structure, often involving multiple components and dependencies
- The continuous evolution and versioning of software
- The importance of source code access for true reusability
- The FAIR4RS Principles aim to make research software:
  - Findable: software and its associated metadata should be easy to find for both humans and machines
  - Accessible: software and metadata should be retrievable via standardized protocols.
  - Interoperable: software should be able to exchange data and work with other software via common formats and APIs.
  - Reusable: software should be well-described and include clear usage licenses to enable reuse, modification and integration into other software.

Key recommendations for implementing FAIR for research software include:

1. Using persistent and unique identifiers for software and its versions
2. Providing rich metadata to describe the software
3. Using open, standardized formats and protocol
4. Providing clear licensing and usage information
5. Following community standards and best practice

By applying FAIR principles, research software can become more discoverable, accessible, and reusable by both humans and machines. This supports transparency, reproducibility, and reuse in computational research.

The FAIR4RS Principles provide aspirational guidelines that software creators and maintainers can work towards incrementally. Tools and practices continue to evolve to support FAIR software across different research domains. The ReSA Actionable FAIR4RS Task Force is developing "good enough" guidelines to help make research software FAIR, based on original suggestions from the biosciences [REF].

There are some aspects of FAIRness that have a relationship to quality. For example, documentation is usually connected to better quality.

[1] FAIR Principles for Research Software (FAIR4RS Principles), https://zenodo.org/records/6623556.

Funded by the European Union

### 2.2.1 Good enough practices

This section attempts to capture the list of the various good practices that are currently being addressed in the EVERSE project.

*This section will be further developed in the next versions of the RF.*

### 2.2.2 Examples of tools relevant for FAIR software

The following is a non-exhaustive list of tools that can help assess or improve the FAIRness of research software. For more comprehensive and up-to-date guidance on tools and services that support FAIR software principles, see the Research Software Quality Toolkit (RSQKit) and the EVERSE Tech Radar.

It's important to note that most existing FAIR software assessment tools were originally developed to assess compliance with FAIR data principles rather than the specific requirements of research software. As a result, some discernment should be used when applying these tools to evaluate a repository's compliance with the FAIR4RS principles, as software has unique characteristics—such as its executable nature, composite structure, and continuous evolution—that differ fundamentally from static datasets.

The tools listed below fall into two main categories: automated assessment tools that provide some level of FAIR evaluation, and user self-assessment tools that guide researchers through structured questionnaires. These tools provide valuable starting points for improving software FAIRness and should be used as part of a broader strategy that includes adherence to community standards, proper documentation practices, and appropriate metadata provision.

**Automated assessment tools:**

(see https://doi.org/10.5281/zenodo.13268685 for comparison between F-UJI, FAIR-enough, FAIR Checker and howfairis. None of these tools can currently be applied directly to assess a repository's compliance with the full set of FAIR4RS principles)

- Originally developed to test compliance with FAIR data principles
  - F-UJI (https://github.com/pangaea-data-publisher/fuji)
  - FAIR Checker (https://github.com/IFB-ElixirFr/FAIR-chekcer)
  - FAIR-enough (https://github.com/vemonet/fair-enough-metrics)
- Targeting research software explicitly
  - Howfairis (https://github.com/fair-software/howfairis), doesn't use FAIR4RS principles but relies on 5 recommendations for FAIR software developed by DANS
  - Extension of F-UJI by EPCC/SSI [6]

**User self-assessment based on guided questionnaires:**

- SATISFYD questionnaire for data (https://satifyd.dans.knaw.nl/)
- "Self-assessment for FAIR research software" questionnaire for FAIR4RS (https://fair-softwarechecklist.net/v0.2/)
- Five recommendations for FAIR software (https://fair-software.nl/, does not explicitly use FAIR4RS)
- FAIR data self assessment tool (https://ardc.edu.au/resource/fair-data-self-assessment-tool/)

## 2.3 Open Source Software

Research software can be published with or without open access to the source code. Open access to source code aligns better with academic research purposes than closed source software; open source software aligns with the FAIR4RS principles. It allows other researchers to directly verify the methods used to produce the results published in papers. It also makes reproducibility much easier. In addition to these research-driven reasons, publishing research software as open source software can help with long term maintenance in a cost-effective way, since interested developers can easily contribute new functionality or fix bugs. Moreover, by integrating with the greater open source ecosystem, researchers can leverage tools and support communities already available. As such, for most academic communities with limited resources, it is also a good choice from a software engineering perspective. Exceptions may be communities with very specific needs and the ability to secure long-term funding to develop and support their own software. In such cases, it's possible that the open source community does not offer enough benefits or has no interest in helping out. It must also be noted that publishing software in an open source way is not completely free of costs, especially if one wants to adhere to common quality standards, like providing some level of user support, interacting with potential contributors, dealing with licensing, etcetera.

Subdimensions and connected indicators of supporting/enabling open source software use in research software may include:

**Source code availability**

- Possible indicator: "has publicly published source code"

**Open source licensing**

- Possible indicators:
  - "uses an open source license from https://opensource.org/licenses"
  - "uses a custom open source license"

**Integration with OSS ecosystems**

Funded by the European Union

- Examples include: uses R instead of SPSS, builds on the numpy ecosystem instead of matlab, runs on Linux, not just Windows, etc.
- Possible indicators:
  - "uses OSS dependencies only"
  - "is compatible with OSS operating system(s)"
  - "runs on OS framework (like R or Galaxy or ...)"

**Alignment with open source community practices**

- Possible indicators:
  - "uses standard contributor's guidelines" (for developers, for users...)
  - "uses Contributor Covenant code of conduct" (https://www.contributor-covenant.org)
  - "uses a platform with community interaction capabilities" (GitHub, GitLab, JIRA...)
  - "issues pre-releases to gather feedback before releases"

**Alignment with open source decision making practices**

- Possible indicators:
  - "decisions and deliberations are logged publicly"
  - "a roadmap is openly maintained"
  - "the governance structure and processes are documented" (BDFL, by community committee, by company/sponsor decision...)

### 2.3.1 Good enough practices

*This section will be further developed in the next versions of the RF.*

### 2.3.2 Examples of tools relevant for open source software

The following is a non-exhaustive list of tools that can help assess or improve open source software practices. For more comprehensive and up-to-date guidance on tools and services that support open source software principles, see the Research Software Quality Toolkit (RSQKit) and the EVERSE Tech Radar.

**License management and analysis:**

- Apache RAT (https://creadur.apache.org/rat/) - tool for auditing license headers
- Tortellini (https://github.com/tortellini-tools/action) - GitHub Action for license compliance checking

**License selection guidance:**

- Choose a License (https://choosealicense.com/) - guidance for selecting appropriate licenses
- tl;drLegal (https://www.tldrlegal.com/) - plain English explanations of software licenses

**Open repositories and platforms:**

- Zenodo (https://zenodo.org/) - open repository for research software publication
- Software Heritage (https://www.softwareheritage.org/) - universal archive for software source code

Note that these dimensions overlap partially with other dimensions (e.g. source code availability could also be categorized under FAIR, and Integration with OSS ecosystems could be seen as just a specification of the Technical Dimension of Compatibility). Despite this, the open source aspect is important enough to warrant separate attention. It allows those involved in research software to better integrate into the much broader world of open source software outside of academia. Speaking the same language as the OSS world makes it easier to get outside support, attract outside talent into academia, and vice versa improves career possibilities for RSEs outside of academia.

## 2.4 Sustainability

Software sustainability can be defined as the capacity of a piece of software to continue to be available in the future, on new platforms, meeting new needs. This means that it is easy to evolve and maintain; fulfils its intent over time; survives uncertainty; and supports relevant concerns (Political, Economic, Social, Technical, Legal, Environmental).

Software sustainability encompasses multiple interconnected aspects that ensure research software remains viable and valuable over time. This includes not only technical maintainability but also community governance, funding models, and institutional support. The sustainability requirements may vary significantly depending on the software tier - analysis code may need minimal long-term planning, while research software infrastructure requires comprehensive governance and funding strategies.

Sustainability practices should be used together, supported by governance procedures. Extensive information on governance goes beyond tooling[2], and different combinations of these practices may be useful depending on the software's audience (as defined by the different views described in the next section).

Common good practices within the Science Clusters identified from discussion with the pilots include:

[2]For governance frameworks, see Code for Society (https://www.codeforsociety.org/incubator/resources/governance-bibliography) and Community Rule (https://communityrule.info/).

- Documentation hosted alongside the repositories (e.g. Read the Docs) for transparency and ease of use.
- Onboarding Practices: Onboarding processes vary but often include:
  - Workshops and tutorials to introduce new users and developers
  - Mentoring programs led by senior developers or researchers
  - Use of example scripts and user stories to demonstrate real-world applications of the software
- As concrete examples of onboarding practices:
  - ESCAPE (ACTS) conducts annual developer workshops with hands-on tutorials
  - PaNOSC (PIConGPU) provides onboarding materials like videos, documents, and personal exchanges with core team members

### 2.4.1 Good enough practices

- **Software maintenance**: regular checks of the code base to ensure quality
- **Refactoring**: examining and rewriting software to be more maintainable, without changing its behaviour
- **Clearly defined support processes and infrastructure**: providing clear workflows for dealing with future development including bugs and feature development
- **Code contribution workflows**: to manage the review and integration of new and revised code
- **CI/CD**: automated process for continuous integration and continuous delivery/deployment to enable early identification of bugs
- **Code review**: methodical assessment of code to identify bugs, increase code quality and help developers understand the code base
- **Project templating**: helps setup software projects in a standard way
- **Project health**: using metrics to ensure the software project is on track, the development team is resilient, and the community of developers / contributors is thriving
- **Project communication**: mechanisms to engage internally and externally to improve software sustainability through more efficient interactions
- **Governance processes**: the rules, principles and mechanisms to ensure software development aligns with the requirements and ethos/values, and enables compliance with any regulatory requirements

### 2.4.2 Examples of tools relevant to Software Sustainability

The following is a non-exhaustive list of tools that can help assess or improve software sustainability practices. For more comprehensive and up-to-date guidance on tools and services that support software sustainability, see the Research Software Quality Toolkit (RSQKit) and the EVERSE Tech Radar.

- **Software maintenance:**

Funded by the European Union

  - Linters: analyse software for errors, vulnerabilities and stylistic issues
  - Code coverage tools

- **Refactoring:**
  - Code Analysis tools including static program analysis tools (e.g. SonarQube, CodeScene) and profilers (e.g. gprof, ValGrind) that can be used to understand internal code relationships and areas to focus attention

- **Support processes:**
  - Issue / ticketing systems including code repository issue trackers (e.g. GitHub Issues, GitLab Issues) and helpdesk ticketing systems (e.g. ZenDesk, Zoho, Spiceworks)

- **Code contribution workflows:**
  - Pull request tooling
  - Integrations with issue / ticketing systems (e.g. GitHub, BitBucket / Jira)

- **CI/CD:**
  - CI/CD tools: these automate the building, testing and/or delivery/deployment of software when changes are made (e.g. Jenkins, Bamboo, Travis CI, GitHub Actions)

- **Code review:**
  - Version control repositories: including code review features like pull requests.
  - Standalone code review tools: (e.g. Gerrit, Collaborator)

- **Project templating:**
  - Project templating tools: e.g. CookieCutter, Yeoman, Copier

- **Project health:**
  - Software project metrics tools: community health dashboards (e.g. Augur, GrimoireLab, Org Metrics Dashboard)

- **Project communication:**
  - Mailing lists
  - Websites
  - Messaging platforms: e.g. Slack, Mattermost, Discord

- **Governance:**
  - Source code scanning tools: including License, copyright and export control compliance tools (e.g. FOSSology, SPDX tools)
  - Software Management Plans

There are many open-source tools available in this space to support these practices.

Funded by
the European Union

# 3 Four Views

The EVERSE reference framework recognises that research software quality cannot be assessed through a single, universal approach. Different stakeholders, contexts, and purposes require tailored perspectives on what constitutes quality and excellence in research software. To address this diversity, the framework provides four complementary views that help users navigate and apply quality considerations appropriate to their specific situation.

These four views are:

- **Three-Tiers View**: Recognises that different types of research software—from personal analysis scripts to broadly-used research infrastructure—have distinct quality requirements and stakeholder needs
- **Software Lifecycle View**: Acknowledges that quality considerations vary depending on where software sits in its development and maintenance lifecycle
- **Personas View**: Understands that different roles in the research ecosystem (researchers who code, RSEs, principal investigators, policy makers, trainers) have varying priorities and responsibilities regarding software quality
- **Science Clusters View**: Recognises that different research domains and communities have established practices, standards, and cultural approaches that influence how software quality is understood and implemented

Each view offers a lens through which the core quality dimensions (Technical, FAIR, Open Source, and Sustainability) can be interpreted and prioritised. Rather than being mutually exclusive, these views are designed to work together, allowing users to consider multiple perspectives when assessing or improving research software quality. The framework's strength lies in this multi-faceted approach, enabling it to serve the diverse needs of the European research software community while maintaining coherence in its underlying principles.

## 3.1 Three-Tiers View

*Diagram from "EVERSE Paving the way towards a European Virtual Institute for Research Software Excellence" presentation by F. Psomopoulos, February 2025, adapted by Aleksandra Nenadic under CC-BY 4.0 licence*

The three-tier model of research software provides a framework for understanding the diverse landscape of software in research. This model distinguishes three tiers of research software, each with distinct purposes, stakeholders, and quality requirements:

Funded by
the European Union


Figure 2: Three-tier model diagram showing the relationship between software tiers, with Analysis Code having high abundance but low reach, Prototype Tools in the middle, and Research Software Infrastructure having low abundance but high reach.

## Analysis Code (Tier 1)

This includes research software that captures computational research processes and methodology, often used in simulation, data generation, preparation, analysis and visualisation. It typically represents software created for personal use with a small scope, such as analysis scripts. Analysis code is abundant in the research ecosystem but typically has limited reach beyond its immediate creators.

EVERSE pilot examples: Individual analysis scripts used in the ESCAPE xAODAnaHelpers framework for analysing high-throughput data from the Large Hadron Collider; custom data processing scripts developed for specific ENVRI climate variable analyses.

## Prototype Tools (Tier 2)

This tier represents research software that demonstrates new ideas, methods, or models for use beyond the project in which it originated, often as a substantive intellectual contribution or proof of concept. While the term "prototype tools" is used in the three-tier framework, these are not experimental or untested tools, but rather software that bridges individual research and broader community use. These tools are designed to answer multiple research questions and are typically developed and used by more than one person within a research team or organisation.

EVERSE pilot examples: The UDPipe language processing suite from SSHOC, which processes multiple languages and can be deployed locally or as a service; machine learning tools for data compression in ESCAPE that are expanding to different use cases.

Funded by the European Union

## Research Software Infrastructure (Tier 3)

This involves software that captures broadly accepted ideas, methods and models for use in research, warranting close researcher involvement in their development. This tier represents broadly applicable research software with the widest reach, often maintained by large, possibly distributed development teams. These software packages become foundational tools that many researchers depend upon.

EVERSE pilot examples: The ACTS Common Tracking Software used across multiple particle physics experiments; the WfExS-backend workflow execution system that supports secure processing across different research infrastructures.

## Tier Relationships and Progression

The essence of the three-tier model is that software in different tiers has distinct purposes, stakeholders, and needs, requiring different approaches to quality assessment and management. Importantly, software can evolve between tiers:

- Analysis code may be published with research outputs and either conclude its lifecycle or, if found valuable by others, evolve into prototype tools
- Prototype tools may gain broader adoption and mature into research software infrastructure
- The relationship is interconnected: higher-tier software builds on the existence of lower-tier software, while successful higher-tier software validates and reinforces the value of the foundational work

## Quality Implications Across Tiers

Different tiers require different emphasis on quality dimensions:

- Tier 1 focuses primarily on functionality and reproducibility, with basic documentation
- Tier 2 requires enhanced maintainability, testing, and community interaction capabilities
- Tier 3 demands comprehensive attention to all quality dimensions, including robust security, scalability, and sustainability practices

## Context and History

The three-tier model originated from the Australian Research Data Commons (ARDC) as part of their "A National Agenda for Research Software" (2022)[10]. It emerged from the need to better categorise and support different types of research software, recognising

Funded by
the European Union

that a one-size-fits-all approach is inadequate for the varied purposes and stakeholders involved in research software.

Since its introduction, the model has gained international recognition and has been referenced in various contexts, including the FAIR Principles for Research Software (FAIR4RS Principles[11]) work. Similar tiered approaches have been adopted by other organisations, such as the German Aerospace Center (DLR)[12] in their Software Engineering Guidelines.

### Alignment with Software Management Plans

The Three-Tier Model aligns closely with practical Software Management Plan (SMP) frameworks. As outlined in the "Practical guide to Software Management Plans" (ARDC, 2022), SMPs can be tailored to low, medium, and high management levels, corresponding to analysis code, prototype tools, and research software infrastructure respectively. This alignment demonstrates how the Three-Tier Model can guide the development of appropriate management practices for different types of research software, ensuring that software management efforts are appropriately scaled to the nature and intended use of the software.

For initiatives like EVERSE, the three-tier model offers a valuable lens through which to view research software quality and excellence, suggesting that practices, tools, and assessment criteria should be adjusted based on the tier of software being considered.

## 3.2 Software Lifecycle View

As part of the EOSC Task Force "Infrastructure for quality research software", it was important to achieve a common understanding of the processes in research software engineering to assess infrastructure needs. SubGroup 1 of this task force examined the research software lifecycle and reported findings in "On the Software Lifecycle"[citation]. The aim was to illustrate how research software lifecycles vary based on developer groups and their intentions.

There is a close relationship between the software lifecycle and the Three-Tier Model described above. Depending on the tier of software, the length of the lifecycle and number of iterations through the lifecycle may vary significantly.

### Lifecycle Patterns by Tier

**Analysis Code (Tier 1)** typically follows a streamlined lifecycle directly driven by specific research questions. Development requires minimal planning and software engineering effort (limited documentation and testing). Once the immediate research question is answered, the software may be published alongside research outputs. At this point, the

Funded by
the European Union

software may either conclude its active development or, if deemed valuable by others, begin evolution toward prototype tools.

EVERSE pilot examples: Individual analysis scripts in ESCAPE's xAODAnaHelpers framework follow this pattern, being developed to answer specific physics questions and then potentially shared with the broader collaboration.

**Prototype Tools (Tier 2)** exhibit more complex lifecycle patterns with closer connections between research and software development cycles. Due to broader scope and longer lifetime, extensive development planning is needed to ensure software and research cycles align. Advanced software engineering practices (issue tracking, semantic versioning, test coverage, code reviews) become essential for team-based development and user feedback collection. Versioned releases are archived and cited in publications, with development continuing to address new research questions and improvements.

EVERSE pilot examples: The WfExS-backend from EOSC-Life demonstrates this pattern, evolving through multiple research questions about secure workflow orchestration while maintaining rigorous development practices.

**Research Software Infrastructure (Tier 3)** operates with software and research cycles that are no longer directly connected. Development teams may be large and distributed across organizations, with different members having varying objectives representing different communities. Large external user bases depend on the software without directly contributing to development. This tier requires the most advanced software engineering techniques, proper development planning, and community management to organize teams, collect feedback, and ensure regular releases. For mission-critical software requiring continuous operation, specialized methods like DevOps and CI/CD are essential, along with governance models and sustainable funding approaches.

EVERSE pilot examples: The ACTS Common Tracking Software exemplifies this tier, serving multiple particle physics experiments with complex governance and release management processes.

## Alternative RSQKit Lifecycle Model

An alternative view of the research software lifecycle, developed as part of the RSQKit to ease navigation of software best practices, emphasizes different stages and the connection between research and software aspects:

This model particularly highlights:

- The iterative nature of research software development
- Decision points where quality considerations become critical
- Integration points with research workflows
- Opportunities for tier progression

Funded by
the European Union

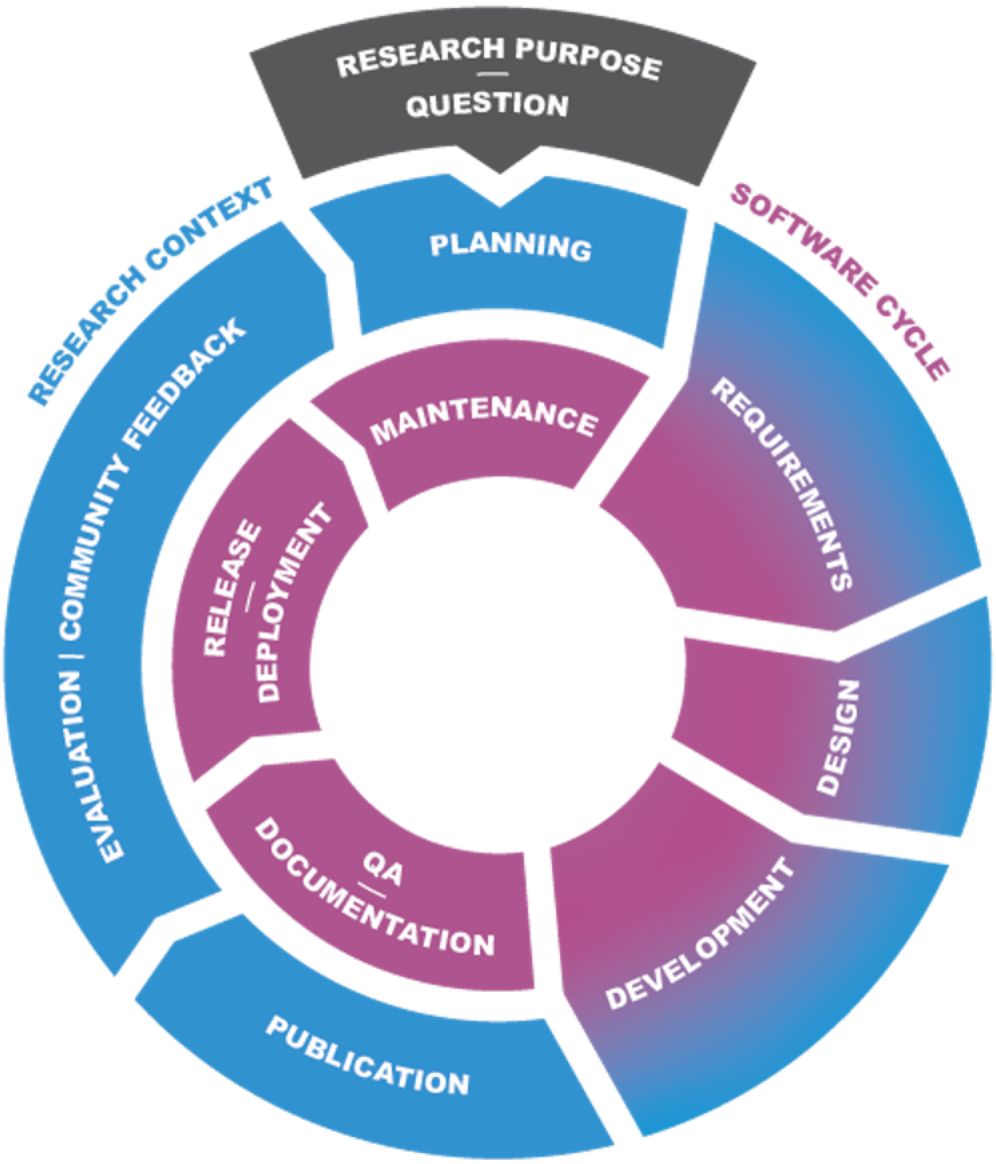


Figure 3: The Research Software Lifecycle developed for the Research Software Quality Kit (RSQKit) originating from the EVERSE project

Funded by
the European Union

### Quality Implications Across the Lifecycle

Different lifecycle stages emphasize different quality dimensions:

- **Planning/Design phases**: Focus on functional suitability and compatibility requirements
- **Development phases**: Emphasize maintainability, testability, and security practices
- **Release/Deployment phases**: Prioritize reliability, performance efficiency, and FAIR principles
- **Maintenance/Evolution phases**: Sustainability and community engagement become critical

### Lifecycle and Quality Assessment

Understanding where software sits in its lifecycle helps determine appropriate quality assessment approaches:

- Early-stage software may prioritize functionality and basic documentation
- Mature software requires comprehensive quality evaluation across all dimensions
- End-of-life software needs preservation and archival quality considerations

The lifecycle view thus provides essential context for applying the EVERSE quality framework, ensuring that quality expectations and assessment methods are appropriate to the software's current stage and intended trajectory.

## 3.3 Personas View

Research software quality considerations vary significantly depending on one's role in the research ecosystem. The Personas View recognizes that different stakeholders have distinct responsibilities, priorities, and levels of engagement with software quality practices. Understanding these differences enables more targeted guidance and ensures that quality frameworks serve the diverse needs of the research software community.

### Core Personas in Research Software

The RSQKit identifies five primary personas within the research landscape, each with specific software quality interests and responsibilities:

**Researcher who codes**: Scientists and researchers who develop software as part of their research activities, ranging from simple analysis scripts to more complex tools. Their primary focus is often on functionality and getting research results, though they benefit from guidance on maintainability and reproducibility practices.

Funded by
the European Union

**Research Software Engineer (RSE)**: Professional software developers working in research contexts who bring software engineering expertise to research projects. They typically prioritize technical quality dimensions like maintainability, testability, and performance, while also understanding research-specific requirements.

**Principal Investigator**: Research leaders responsible for project oversight, funding, and strategic direction. Their software quality interests often center on sustainability, compliance, risk management, and ensuring software supports research objectives and reproducibility.

**Policy Maker**: Individuals involved in developing guidelines, standards, and policies for research software at institutional, national, or international levels. They focus on frameworks for assessment, recognition, funding criteria, and alignment with broader research integrity goals.

**Trainer**: Educators and training coordinators who teach software development skills to researchers. They need to understand quality practices across all levels and translate complex concepts into accessible learning materials appropriate for diverse audiences.

## Persona-Specific Quality Priorities

Different personas emphasize different aspects of the EVERSE quality framework:

**Technical Dimensions**: RSEs typically lead on technical quality implementation, while researchers who code need guidance on achieving adequate technical quality without extensive software engineering background.

**FAIR Principles**: All personas value FAIR software, but researchers and PIs may focus more on discoverability and reusability, while RSEs implement the technical infrastructure to achieve FAIRness.

**Open Source Practices**: Policy makers and PIs often drive open source decisions at strategic levels, while RSEs and researchers who code handle implementation details like licensing and contribution workflows.

**Sustainability**: PIs and policy makers focus on long-term planning and funding models, while RSEs and trainers work on technical and educational approaches to sustainability.

## Role Specialization and Software Tiers

As research software progresses through the three-tier model, the likelihood of role specialization tends to increase:

**Analysis Code (Tier 1)**: Often developed by researchers who code working independently, with minimal role differentiation.

**Prototype Tools (Tier 2)**: May involve collaboration between researchers who code and RSEs, with PIs providing oversight and direction.

Funded by
the European Union

**Research Software Infrastructure (Tier 3)**: Typically requires teams with specialized roles, including dedicated RSEs, formal project management, policy oversight, and structured training programs.

## Persona Interactions and Collaboration

Effective research software quality often emerges from collaboration between personas rather than isolated individual efforts:

- Researchers who code and RSEs collaborate on balancing research needs with technical best practices
- PIs and Policy makers align project goals with institutional and community standards
- Trainers work across all personas to build capacity and share knowledge
- RSEs often serve as bridges between technical implementation and research requirements

## Practical Applications

The Personas View helps in:

**Tailoring Quality Guidance**: Different personas need different types of information, tools, and support to contribute effectively to software quality.

**Resource Allocation**: Understanding persona-specific needs helps prioritize development of tools, training materials, and support services.

**Communication Strategies**: Quality initiatives can be framed in ways that resonate with each persona's priorities and constraints.

**Career Development**: Recognizing distinct personas supports career path development and professional recognition for different types of contributions to research software quality.

## Integration with Other Views

The Personas View intersects with other EVERSE framework views:

- Different science clusters may emphasize certain personas or have additional domain-specific roles
- Software lifecycle stages may require leadership from different personas
- Three-tier progression often involves expanding the range of personas involved in software development and maintenance

Funded by the European Union

This view ensures that the EVERSE framework serves the real people working with research software, acknowledging their diverse backgrounds, responsibilities, and contributions to software quality and excellence.

## 3.4 Science Clusters View

In EVERSE, we consider representative software use cases (pilots) from the five EOSC Science Clusters, to be used in a feedback loop between EVERSE and the scientific communities. These use cases are:

- **SSHOC**: UDPipe, a multi-language textual analysis pipeline of tools (local or as a service)
- **ESCAPE**: software for ML-enabled data compression (Baler), reconstruction (ACTS) and data analysis software built for Open Science (xAODAnaHelpers)
- **Life sciences RI**: software for secure and federated workflow orchestration and movement of encrypted data (The Workflow Execution Service backend)
- **PANOSC**: software used to enable heterogeneous computing (ALPACA) and for scaling to high performance computing (PIConGPU)
- **ENVRI Community**: ENVRI-HUB knowledge base and Virtual Research Environment

While software quality, as well as its dimensions and indicators (and therefore RSQKit personas and tasks) are meant to be general and cross-field, there are cluster-specific aspects due to both technical and cultural/work environment differences. These differences mean that the personas above will be supplemented by other roles in some clusters, and that certain dimensions will be considered more relevant by some of the personas in certain clusters.

### Additional Cluster-Specific Roles

In the ESCAPE Cluster, the role of RSE is only emerging recently; software developers are historically physicists who know the physics challenges that the software must solve very well but may not have received formal training in software development. For this reason, EVERSE can help with the recognition of (and with retaining) professional figures that focus on software development, and building career paths that include software-centered roles.

Given that many scientists in ESCAPE and PANOSC work in large, often distributed collaborations, there are additional roles that are interested in software quality, namely: host laboratories, experiment coordinators, data/open data managers and central/distributed computing site managers (the sites are where the software runs).

In the life sciences (LIFE-RI), there is a large community of users who do not code but who are still interested in software quality – and they are not necessarily principal investigators, although they could be considered within that category.

Funded by
the European Union

### Quality Dimension Prioritization

Different clusters prioritize quality dimensions based on their specific needs and contexts. While all dimensions remain relevant, emphasis varies across the science clusters depending on their domain requirements, technological constraints, and community practices.

*Note: Visual representation of cluster-specific prioritization will be developed for future versions of this framework.*

### Integration with EVERSE Framework

We use directly the EOSC Science Cluster definitions, with each cluster having sub-views based on the individual research infrastructures or other well-defined science communities within them. This ensures that domain-specific requirements are captured while maintaining alignment with the overall EVERSE quality framework.

## 4 Conclusions

*This section will be further developed in the next versions of the RF.*

Funded by
the European Union