Research Software Engineering Policy Template

Version 1.1, March 2025

Introduction

Research software plays a pivotal role in advancing academic research across academic disciplines. Ensuring that research software is developed, maintained, and shared in alignment with best practices enhances reproducibility, reliability, and long-term impact. This template, prepared by the Oxford Research Software Engineering (OxRSE) Group (rse.ox.ac.uk), is intended for departments to adapt and adopt as a policy framework.

This document outlines key principles, responsibilities, and practices to guide research software development and management within an academic department. It is intended to be adapted to meet the unique needs of each academic unit while remaining aligned with institutional and funding body standards.

Purpose and scope

The purpose of this policy is to

- Promote the development of high-quality, sustainable, and reproducible research software.
- Establish clear responsibilities and expectations for researchers, research groups, and the department.

This policy applies to

- All research staff, students, and collaborators involved in developing, using, or maintaining research software within the department.
- Research software created during the course of research projects, including scripts, codebases, computational models, and software frameworks.
- All stages of the research software lifecycle, from planning to retirement.

Key principles

Sustainability

Research software should be designed and maintained to ensure usability beyond the lifetime of the project. Key aspects include documentation, modularity, and appropriate choice of technologies.

Reproducibility

Research software should enable reproducible and verifiable research outcomes. This includes clear documentation of methodologies, dependencies, and computational workflows.

Openness

Wherever possible, research software should adhere to open science principles, including open licensing and code sharing, while respecting intellectual property rights and confidentiality.

Investment

Attaining the outcomes promoted by these principles requires investment from individual researchers, supervisors, and departments. Investment in these outcomes means dedicating time and effort to acquiring and honing the skills required to do quality work, and to doing such work once trained. These investments carry an up-front cost that is repaid in increased maintainability of software and reliability of associated results.

Best practices

Version control: use version control systems (e.g., Git) for all research software projects; store version-controlled repositories on appropriate platforms (e.g., GitHub, GitLab); ideally ensure the repository is owned by an organization with multiple owners to reduce the risk of loss of access and ensure continuity of access after project end.

Documentation: include clear documentation for installation, usage, and maintenance; provide comments in code to improve readability and usability by others.

Testing and validation: implement testing to ensure software reliability.

Licensing: select an appropriate license early in the project lifecycle; document licensing terms clearly in the project repository.

Collaboration: foster collaboration by following coding standards, clear communication, and adopting tools for team development (e.g., code review platforms).

Archiving and preservation archive versions of software associated with research outputs (e.g. articles, theses, etc) in a long-term repository with appropriate metadata; provide clear instructions for future users, including dependencies and system requirements.

Responsibilities

Researchers

- Develop research software that adheres to best practices.
- Plan for software sustainability and reproducibility during the project design phase.
- Publish and share software outputs in line with open science principles.

Supervisors and Principal Investigators (PIs)

- Ensure that research projects include sufficient resources (time, funding, expertise) for research software development and maintenance.
- Support the recognition of research software as a key research output in grant applications and publications.

The department

- Encourage research staff, students and collaborators to attend workshops and training sessions on research software engineering tools and methodologies.
- Consider collaboration with OxRSE to organize tailored training for its research staff and students.
- Recognize and reward research software contributions in promotion and hiring processes.
- Collaborate with the OxRSE for guidance and expertise.

Oxford Research Software Engineering Group (OxRSE)

- Provide expertise, consultation, and training in research software engineering.
- Advocate for best practices and open research principles across the University.
- Serve as a central resource for guidance on policy implementation.
- Serve as a central pool of research software engineers to develop software.

Compliance checklist

1. Version control

☐ The project uses a version control system
  ☐ Git (strongly recommeneded)
  ☐ Mercurial
  ☐ Subversion (SVN)
  ☐ Not under version control
  ☐ Other – give details:

☐ The repository contains a README.md with project overview, usage instructions, and contact details.

☐ The code is stored on a collaborative platform
  ☐ GitHub
  ☐ GitLab
  ☐ Bitbucket
  ☐ Institutional Git service (e.g., University-hosted GitLab)
  ☐ Shared file storage (e.g., OneDrive, SharePoint – not recommended for active development)
  ☐ Not shared
  ☐ Other – give details:

☐ Repository owner
  ☐ Individual researcher
  ☐ Research group
  ☐ Supervisor / PI

- ☐ Department or faculty
- ☐ Oxford RSE Group (OxRSE)
- ☐ Other – give details:

☐ Repository access
- ☐ Owner only
- ☐ Project collaborators
- ☐ Supervisor / PI
- ☐ Research group members
- ☐ External collaborators
- ☐ Other – give details:

☐ Repository visibility
- ☐ Public
- ☐ Private
- ☐ Internal (e.g., within an institution)
- ☐ Other – give details:

☐ Repository administrator(s) or maintainer(s)
→ Name(s) and role(s):

2. Collaboration and development workflows

☐ Collaborative development workflows are in use.
- ☐ Feature branches
- ☐ Pull requests or merge requests
- ☐ Code reviews before merging
- ☐ Issue tracking (e.g., GitHub Issues, Jira, GitLab Issues)
- ☐ CI/CD pipelines for testing or deployment
- ☐ Other – give details:

☐ The project follows a consistent coding style or standard agreed within the team.
- ☐ Style guide documented in CONTRIBUTING.md or README.md
- ☐ Linting tools are used (e.g., flake8, ruff, eslint)
- ☐ Code formatting tools are used (e.g., ruff, black, prettier)
- ☐ Style adherence is checked automatically (e.g., in CI workflows)
- ☐ Agreed upon coding standards implemented through pre-commit configuration
- ☐ Other – give details:

- ☐ Continuity of repository access after project end
  - ☐ Ownership and access will remain with a permanent institutional account (e.g., department or PI account)
  - ☐ Repository will be transferred to a long-term maintainer (e.g., OxRSE, successor project)
  - ☐ Repository will be archived in a long-term public repository (e.g. Zenodo)
  - ☐ Other – give details:

2. Documentation

- ☐ Installation instructions are included
  - ☐ pyproject.toml (recommended) or requirements.txt (Python)
  - ☐ environment.yml (Python/R)
  - ☐ DESCRIPTION file (R package)
  - ☐ Makefile, CMakeLists.txt or build script (C/C++)
  - ☐ Dockerfile or container specification
  - ☐ Manual installation steps documented in README.md
  - ☐ Other – give details:

- ☐ User documentation is provided and includes
  - ☐ Input/output data formats
  - ☐ Command-line or GUI usage examples
  - ☐ Configuration options or parameters
  - ☐ Troubleshooting or FAQ section
  - ☐ Other – give details:

- ☐ Developer documentation is provided and includes
  - ☐ Code structure or module overview
  - ☐ Instructions for contributing (e.g., CONTRIBUTING.md)
  - ☐ Build and test instructions
  - ☐ API or function-level documentation
  - ☐ Other – give details:

- ☐ Inline comments are present in the code and explain:
  - ☐ Non-obvious logic
  - ☐ Algorithmic steps or mathematical operations
  - ☐ Function/class purpose and assumptions
  - ☐ Other – give details:

3. Testing and validation

- ☐ Automated tests are implemented using a testing framework.
  - ☐ pytest (Python)
  - ☐ unittest (Python)
  - ☐ nose (Python)
  - ☐ doctest (Python)
  - ☐ testthat (R)
  - ☐ Jupyter notebook-based tests (Python)
  - ☐ Custom test suite or scripts (C/C++)
  - ☐ Other – give details:

- ☐ Tests cover core functionality and are runnable with clear pass/fail results.
  - ☐ Unit tests
  - ☐ Integration tests
  - ☐ System or end-to-end tests
  - ☐ Regression tests
  - ☐ Test coverage is measured (e.g., with coverage.py)
  - ☐ Tests are automatically run (e.g., via CI pipeline)
  - ☐ Other – give details:

- ☐ Validation of the software has been performed and documented.
  - ☐ Against real or published datasets
  - ☐ Using synthetic or test-case data
  - ☐ Against expected mathematical or statistical outcomes
  - ☐ Results are described in a validation.md, notebook, or publication
  - ☐ Other – give details:

4. Licensing and Reuse

- ☐ A LICENSE file is present in the repository.
  - ☐ MIT
  - ☐ Apache 2.0
  - ☐ GPLv3
  - ☐ BSD 3-Clause
  - ☐ Proprietary / restricted
  - ☐ Other – give details:

- ☐ The license is appropriate for the intended reuse and clearly stated in:
    - ☐ LICENSE file
    - ☐ README.md
    - ☐ Project metadata (e.g., pyproject.toml)
    - ☐ Other – give details:

5. Reproducibility

- ☐ The computational environment is documented using reproducible formats.
    - ☐ pyproject.toml (recommended) or requirements.txt (Python)
    - ☐ environment.yml (Python or R)
    - ☐ DESCRIPTION and renv.lock (R)
    - ☐ Makefile, CMakeLists.txt, or pkg-config (C/C++)
    - ☐ Dockerfile or Singularity definition
    - ☐ Other – give details:

- ☐ All external dependencies are explicitly listed and versioned.
    - ☐ Python packages, via requirements.txt or lock files such as poetry.lock and uv.lock
    - ☐ R packages or CRAN/Bioconductor dependencies
    - ☐ System-level dependencies (e.g., OS packages, compilers, shared libraries)
    - ☐ C/C++ libraries or headers (e.g., via pkg-config, vcpkg, or manual build instructions)
    - ☐ Hardware or platform-specific dependencies
    - ☐ Other – give details:

- ☐ Computational workflows are reproducible using:
    - ☐ Shell scripts or Makefiles (all languages)
    - ☐ Jupyter or R Markdown notebooks
    - ☐ Workflow engines (e.g., Snakemake, Nextflow, CWL)
    - ☐ Build and test automation for compiled languages (e.g., CMake + CTest)
    - ☐ Other – give details:

6. Archiving and Citation

- ☐ Versions of the software linked to research output (e.g. articles, theses, etc) are archived in a long-term repository.
    - ☐ Zenodo
    - ☐ Figshare
    - ☐ Software Heritage
    - ☐ Institutional repository
    - ☐ Other – give details:

☐ A Digital Object Identifier (DOI) has been generated and recorded.
→ DOI:

☐ Metadata for the archived version includes:
- ☐ Title
- ☐ Author(s)
- ☐ Version number
- ☐ License
- ☐ Date of release
- ☐ Other – give details:

☐ The software is formally cited in related publications.
- ☐ Citation using DOI in publication references
- ☐ Citation via CITATION.cff or codemeta.json
- ☐ Mentioned in acknowledgements or methods sections
- ☐ Other – give details:

☐ Author names and contribution roles are documented.
- ☐ AUTHORS.md or equivalent
- ☐ CRediT taxonomy used
- ☐ Contribution descriptions included in metadata or documentation
- ☐ Other – give details:

This policy template is provided as a framework for departments to adapt to their specific needs.

For further guidance, please contact the Oxford Research Software Engineering Group