Preferred Practices Through a Project Template

1st Peter F. Peterson
Computer Science and Mathematics Division
Oak Ridge National Laboratory
Oak Ridge, TN, USA
https://orcid.org/0000-0002-1353-0348
petersonpf@ornl.gov

2nd Chen Zhang
Computer Science and Mathematics Division
Oak Ridge National Laboratory
Oak Ridge, TN, USA
https://orcid.org/0000-0001-8374-4467
zhangc@ornl.gov (*Corresponding author*)

3rd Jose M. Borreguero-Calvo
Computer Science and Mathematics Division
Oak Ridge National Laboratory
Oak Ridge, TN, USA
https://orcid.org/0000-0002-0866-8158
borreguerojm@ornl.gov

4th Kevin A. Tactac
Computer Science and Mathematics Division
Oak Ridge National Laboratory
Oak Ridge, TN, USA
tactacka@ornl.gov

***Abstract*—In the realm of scientific software development, adherence to best practices is often advocated. However, implementing these can be challenging due to differing opinions. Certain aspects, such as software licenses and naming conventions, are typically left to the discretion of the development team. Our team has established a set of preferred practices, informed by, but not limited to, widely accepted best practices. These preferred practices are derived from our understanding of the specific contexts and user needs we cater to. To facilitate the dissemination of these practices among our team and foster standardization with collaborating domain scientists, we have created a project template for Python projects. This template serves as a platform for discussing the implementation of various decisions. This paper will succinctly delineate the components that constitute an effective project template and elucidate the advantages of consolidating preferred practices in such a manner.**

***Index Terms*—Python, best practices, template repository, research software engineer**

I. Introduction

In any software development team, both formal and informal rules coalesce based on shared values and practical needs. Documenting these rules is crucial for consensus building and serves as a valuable guide for newcomers, particularly as the team expands. At Oak Ridge National Laboratory (ORNL), the Neutron Data Project (NDP) team oversees a broad spectrum of Python software of various origins and frequently launches new, typically small, projects. To standardize project layouts, expedite new project initiation, and reduce the overhead incurred when maintaining a software portfolio, we have developed a comprehensive project template.

This paper outlines the components encompassed in our Python project template, ranging from opinion-based guidelines to practical best practices. This template is of particular relevance to research software engineers, as it can be adopted to establish their project standards, thereby optimizing efficiency and collaboration within the research software engineering community.

II. Template Contents

The neutrons python project template (on github) serves as a well-organized repository of essential configuration information and reusable examples. It not only promotes team consensus and efficient documentation but also provides an easily accessible source of information, proving particularly beneficial for new team members and collaborators. The included elements such as README contents, frequently used tools, libraries, and build configurations, all contribute to enhance clarity and foster reuse.

Although many development teams use templating tools like Cookiecutter [1] to initiate new projects, this approach was not adopted by the NDP team. Despite the convenience of Cookiecutter, it requires instantiation to fully unveil its features. Preference was given to a working example that directly illustrates the necessary modifications. This strategy ensures that projects on GitHub appear ready-to-use, thereby encouraging domain scientists to follow NDP's preferred practices. From the standpoint of a research software engineering team, this method contributes to enhancing software stability and usability, thus minimizing potential issues during collaboration.

A. Preferred license

Choosing the right open-source license is crucial for any project. Among the numerous open source licenses [2], our default choice is the MIT License due to its simplicity and

permissiveness. The MIT License only requires the preservation of copyright and license notices, making it easy to comply with organizational requirements while encouraging open collaboration. However, depending on project-specific needs or constraints, we remain flexible and open to adopting other licenses as necessary. This adaptability ensures that our licensing aligns with both our team's values and the broader requirements of our projects.

B. OpenSSF Best Practices Badge Program

The Open Source Security Foundation (OpenSSF) [3] is a community of developers and security experts dedicated to securing open source software. Supported by the Linux Foundation [4], they have developed the OpenSSF Best Practices Badge Program [5], which outlines guidelines for open source software development. These guidelines cover essential topics such as basics (e.g., website and license), change control (e.g., revision control and unique version numbers), reporting (e.g., bugs and vulnerabilities), quality (e.g., automated testing and compiler warnings), security (e.g., involving knowledgeable security personnel), and analysis (e.g., static and dynamic code analysis).

Our project template incorporates several of these principles by default. By utilizing GitHub [6], including a license, and pre-configuring static analysis tools, projects that adopt our template are well-positioned to meet the criteria for a passing badge. While achieving a passing badge does not guarantee high-quality software, it establishes a solid foundation for open source projects, promoting best practices and effective communication with the community.

C. Dependency management

A topic that needs to be decided early in a python project is how dependencies will be managed and how the software will be distributed. Managing dependencies when creating and distributing Python packages can be complex. Many of the more popular options for dependency management involve virtualization via things like conda [7], virtualenv [8], and poetry [9]. The choice of dependency management system is heavily influenced by how the software will be packaged and distributed, however, selecting one does not necessarily dictate the other. Some of our software is python with much written in C++, which has many C++ dependencies [10] that are not readily available via PyPi [11]. Based on this and the need to install python software into a central location on behalf of users, our group selected conda for both packaging and dependency management.

Anaconda [7] simplifies dependency management by storing package configurations in `conda.recipe/meta.yaml`, promoting reproducible environments. While publishing a package directly via PyPI [11] is straightforward, it lacks extensive dependency resolution, leading to potential issues in production. To circumvent this, we use Anaconda to mitigate dependency concerns, installing any packages exclusive to PyPI last to ensure maximum compatibility. Anaconda circumvents having to rely on resources installed on the host machine, such as compilers, database client/servers, as well as system, numerical, and GU libraries. Different software packages or even different versions of the same package will require different version of some of these resources. Encapsulating these resources in anaconda environments allows us to consistently deploy more than 30 products to over 100 systems every two weeks. Despite its efficiency, this method may still require additional configuration for projects with complicated dependencies or those involving multiple languages.

D. Static analysis configuration

Static analysis tools are essential for ensuring code quality, catching numerous common issues even though they may occasionally produce false positives. The Python-based pre-commit framework [12] simplifies the integration of various static analysis tools, ensuring they run before code is committed, thus conserving automated build resources for testing. Pre-commit's extensibility supports a wide range of tools, and its service, pre-commit.ci [13], runs these tools automatically and commits any changes. This acts as a safety net for developers who might not configure pre-commit locally or use web-based integrated development environments (IDEs). Additionally, pre-commit.ci proactively creates pull requests for updates to itself and the tools used.

Our team configures pre-commit with built-in tools to block large files, check YAML syntax, enforce consistent end-of-line characters, and remove trailing spaces. For Python, we have adopted the recently released ruff and ruff-format tools [14], which are faster replacements for flake8, pylint, and black [15]–[17]. Ruff also resolves configuration discrepancies among these tools, simplifying setup by requiring only a single configuration at a centralized location.

E. Package layout

```
project_root/
├── src/
│   └── package_name/
│       └── __init__.py
├── tests/
├── notebooks/
│   └── example_notebook.ipynb
├── docs/
│   └── conf.py
├── scripts/
│   └── example_script.py
├── conda.recipe/
│   └── meta.yaml
├── environment.yml
└── .github/
    └── workflows/
        └── ci.yml
```

Fig. 1. Preferred Python Package Layout

A well-structured package layout (Figure 1) is vital for maintaining organization and efficiency in Python projects.

Our preferred layout supports development, documentation, and deployment needs, ensuring clarity and ease of use. The key components of the repository are listed as follows:

- Source Files (`src/package_name/`): This directory stores all Python source files, ensuring a clean namespace and reducing import errors.
- Tests (`tests/`): This directory hosts all unit test and integration test files, ensuring thorough testing and validation of the project.
- Demonstration Notebooks (`notebooks/`): This optional folder contains Jupyter notebooks [18] for demonstrating the package's functionality, useful for tutorials and examples.
- Documentation (`docs/`): This directory stores all project documentation, which is published using Read the Docs [19], ensuring consistent and accessible documentation.
- Scripts (`scripts/`): For packages with application features requiring standalone command-line applications, this optional folder stores the necessary starting scripts.
- Anaconda Packaging (`conda.recipe/`): This directory contains configuration files for creating conda packages, facilitating distribution and installation.
- Development Environment (`environment.yml`): A single YAML file provides the configuration for the development environment, ensuring consistency across setups.
- GitHub Workflows (`.github/workflows/`): This folder contains templated GitHub Action definitions for CI/CD processes, automating testing and deployment.

This layout promotes organization, clarity, and ease of use by separating different aspects of the project. It makes management and collaboration easier and ensures that new team members can quickly understand the project structure. The consistent structure also aids in automated processes and helps maintain high-quality standards, reducing the workload for the devops engineers by using a consistent CI/CD structure.

F. Specifying metadata for development tools

Proper configuration of development tools is essential for efficient software development and publishing. We use a variety of tools, including Python's native tools for packaging, `versioningit` for versioning, `Anaconda` for packaging and publishing, and GitHub Actions for CI/CD automation. Each of these tools needs to be configured correctly to ensure smooth and efficient workflows. As these tools evolve in time, the template project is updated as necessity arises. These updates have the additional advantage that any fixes to the template can be readily ported to any child project. For instance, troubleshooting efforts are considerably reduced when the CI stops working because a update in any of the third-party GitHub actions cause the workflow to fail.

Storing Python package metadata correctly is essential for package management and distribution. We follow `PEP 621` [20] and the latest packaging specifications [21] to store metadata in `pyproject.toml`, which centralizes configuration and improves interoperability. This standardized approach ensures compatibility with various tools, though it may require updates as specifications evolve.[1]

Versioning software reliably is crucial for tracking changes and ensuring consistency. Managing version numbers manually can lead to errors, so we use the `versioningit` [22] tool to automate versioning based on Git tags. This ensures accuracy and reduces manual intervention, providing consistent and traceable versioning, though it requires adherence to a specific workflow, which is documented in the on-boarding guide for NDP team members.

Automating testing and deployment with GitHub Actions addresses common challenges such as ensuring code quality and reducing manual deployment errors. We configure GitHub Actions to run unit tests for every pull request, integration tests for every change in the protected branches, and automated package building and publishing for every major releases. This setup ensures continuous integration and delivery, enhancing reliability and efficiency, and the template configuration files in `.github/workflows` take care of most of the work, allowing developers to focusing on feature implementation.

In addition to the tools described above, we also use additional services such as Read the Docs and CodeCov to improve overall software quality. The guidelines on how to install CodeCov access token are detailed in the README, which enables automatic uploading of the coverage reports generated during execution of the GitHub actions. These tools offer straightforward integration with GitHub, and their configuration files can often be reused across projects. By including them in the template repository, new projects can benefit from these services right from the start, enhancing documentation accessibility and code coverage analysis.

G. GUI application with MVC pattern

Developing GUI applications differs from creating libraries and command-line tools, necessitating extra care. User interface design, event handling, and state management all add complexities that require careful planning to ensure a user-friendly and responsive application. There are various design patterns available for GUI development, but we've chosen to use the Model-View-Controller (MVC) pattern for all our GUI applications. This choice simplifies the development process and allows team members to easily understand and contribute to any project within our organization.

Listing 1. home_view.py

```
from qtpy.QtWidgets import (
    QHBoxLayout,
    QWidget,
)

class Home(QWidget):
    """Main widget"""
    def __init__(self, parent=None):
        super().__init__(parent)
```

[1]At the time of writing, `setup.py` is still required to build C-extensions for a Python package, which is a common requirement for large projects.

```
        layout = QHBoxLayout()
        self.setLayout(layout)
```

Listing 2. home_presenter.py

```
class HomePresenter:
    """Main presenter"""
    def __init__(self, view, model):
        self._view = view
        self._model = model

    @property
    def view(self):
        """Return the view for this presenter"""
        return self._view

    @property
    def model(self):
        """Return the model for this presenter"""
        return self._model
```

Listing 3. home_model.py

```
from mantid.kernel import Logger

logger = Logger("PACKAGENAME")

class HomeModel:
    """Main model"""

    def __init__(self):
        return
```

The MVC pattern separates concerns into three interconnected components: Model, View, and Controller. This division allows for each component to be developed, tested, and maintained independently, thus improving the overall quality and scalability of our applications. To facilitate quick start and maintain consistent coding standards, we provide template files in our project template repository. These templates offer a structured starting point for the main components of an MVC application.

We use qtpy to abstract out lower-level dependencies. Qtpy is a wrapper that allows us to use either PyQt or PySide libraries without changing the codebase. This abstraction ensures our code remains flexible, adaptable to different underlying libraries, and simplifies dependency management while enhancing portability.

By adhering to the MVC pattern and using our provided templates, we ensure all GUI applications within our organization follow a consistent structure and coding standard. This approach not only streamlines development but also promotes better collaboration and code quality across different projects.

III. Using the Project Template

This section demonstrates how the template repository can be applied to various scenarios within the Neutron Data Project (NDP), including starting new Python projects (both non-GUI and GUI applications), modernizing existing projects, and encouraging domain scientists to refine their prototypes. These examples highlight the repository's ability to streamline development and promote best practices.

A. For starting new projects

The template repository offers a robust foundation for new Python projects. For non-GUI applications, the provided structure ensures consistency and organization, enabling developers to focus on functionality rather than setup. Instructions on how to remove unnecessary files (for example, non-GUI apps do not need MVC template files) are provided in the README, allowing developers to modify the repository once the new project is created from the template. The standardized layout helps maintain a high standard of code quality and facilitates easier onboarding for new team members. For GUI applications, the included MVC pattern templates guide developers in creating clean, maintainable designs from the outset. This structured approach minimizes setup time and helps maintain a high standard of code quality.

B. For modernizing existing projects

Modernizing an existing project can be challenging, but the template repository serves as a valuable reference for incorporating best practices. By adopting the structured layout and modern tooling provided by the template, legacy projects can be updated to enhance maintainability, scalability, and performance. This modernization process includes integrating CI/CD pipelines, static analysis tools, automated testing, package publication, code coverage, and documentation publication. All these additions dramatically improve the overall development workflow and ensure a more robust and reliable codebase.

C. Encouraging Domain Scientists

Domain scientists often develop prototype software with significant differences in structure and standardization. Presenting a ready-to-use template repository motivates these scientists to adopt preferred practices, making their prototypes more robust and easier to integrate into larger projects. This approach reduces barriers to collaboration, as the prototype software is already aligned with established development standards, facilitating a smoother and more productive teamwork.

The template repository is a living document that evolves over time to incorporate new best practices and tools. Its version tracking and branching capabilities ensure versatility and applicability to a wide range of software development projects. This adaptability allows the template to meet the diverse needs of the Neutron Data Project (NDP) and support ongoing innovation in software development.

IV. Maintenance and future improvements

Maintaining and improving the project template is an ongoing process guided by practical development needs and team consensus. Our approach to sustaining this template is structured yet flexible, allowing for both incremental updates and significant changes when necessary.

These strategies ensure that our project template remains relevant, adaptable, and aligned with both evolving best practices and the specific needs of our software development environment. By balancing flexibility with centralized oversight,

we can continuously improve our template while minimizing disruption to ongoing projects. The following strategies outline how our team addresses maintenance and future improvements.

A. Incremental Updates via Pull Requests

Individual developers are encouraged to explore and adopt new tools or practices during their project development. When a developer identifies a beneficial improvement that does not require a drastic overhaul of the template, they can submit a pull request. The proposed change is then reviewed by one of the designated gatekeepers. If the change is accepted, it becomes the new standard for future projects. While existing projects are encouraged to adopt the updated practices, they are not required to do so.

B. Collaborative Decision-Making for Drastic Changes

In cases where a proposed change involves a substantial deviation from the existing template structure or introduces a significant new tool, the development team organizes a dedicated meeting. These discussions, held either in person or online, allow the team to evaluate the potential impact and reach a consensus on whether the new tool or practice should be adopted as the new standard. This ensures that any major updates are thoughtfully considered and align with the broader goals of the team.

C. Centralized Documentation of Issues and Solutions

A common challenge in maintaining multiple software projects is staying updated with the latest versions of tools and plugins, especially in continuous integration workflows like GitHub Actions. Often, build failures arise due to outdated plugin versions that are no longer maintained or have changed their interfaces. The template repository serves as a central information hub, where solutions to these problems are documented through pull requests. Software project leads can contribute their findings and, in turn, benefit from existing solutions recorded in the repository. This shared knowledge base reduces redundant troubleshooting efforts and helps ensure that all projects are using stable and supported configurations.

V. Summary

The comprehensive project template developed by the Neutron Data Project (NDP) at Oak Ridge National Laboratory provides a robust foundation for Python software development. By incorporating best practices and standardized configurations, the template streamlines the initiation of new projects, facilitates the modernization of existing ones, and encourages domain scientists to adopt structured development approaches. This template not only enhances collaboration and code quality but also supports ongoing innovation by evolving over time to integrate new tools and practices, ensuring it remains up to date, versatile, and effective for a wide range of scientific software development needs.

For the research software engineering community, this template repository offers significant benefits. It promotes consistent and efficient development practices, enhancing collaboration between software engineers and domain scientists. By providing a structured and standardized approach, the template reduces barriers to the setup of the ecosystem supporting a software package, making it easier for domain scientists to adopt best practices and contribute effectively. This fosters a more integrated and productive research environment. We recommend other research software engineering groups create their own template repository and share it, thereby promoting research software engineering practices to a broader audience and enhancing the overall quality of research software.

References

[1] A. R. Greenfeld, D. R. Greenfeld, and R. Pierzina. (2019) Cookiecutter: Better project templates. [Online]. Available: https://cookiecutter.readthedocs.io/en/2.0.2/

[2] (2024). [Online]. Available: https://choosealicense.com/

[3] (2024). [Online]. Available: https://openssf.org/

[4] (2024). [Online]. Available: https://www.linuxfoundation.org/

[5] (2024). [Online]. Available: https://www.bestpractices.dev/

[6] (2024). [Online]. Available: https://www.github.com/

[7] Anaconda, Inc. (2024) Anaconda. [Online]. Available: https://www.anaconda.com/download/

[8] (2024) venv — creation of virtual environments. [Online]. Available: https://docs.python.org/3/library/venv.html

[9] (2024) Poetry. [Online]. Available: https://python-poetry.org/

[10] O. Arnold, J. Bilheux, J. Borreguero, A. Buts, S. Campbell, L. Chapon, M. Doucet, N. Draper, R. F. Leal, M. Gigg, V. Lynch, A. Markvardsen, D. Mikkelson, R. Mikkelson, R. Miller, K. Palmen, P. Parker, G. Passos, T. Perring, P. Peterson, S. Ren, M. Reuter, A. Savici, J. Taylor, R. Taylor, R. Tolchenov, W. Zhou, and J. Zikovsky, "Mantid—data analysis and visualization package for neutron scattering and μ sr experiments," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 764, pp. 156 – 166, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0168900214008729

[11] (2024) Python package index - pypi. [Online]. Available: https://pypi.org/

[12] (2024). [Online]. Available: https://pre-commit.com/

[13] (2024). [Online]. Available: https://pre-commit.ci/

[14] (2024). [Online]. Available: https://docs.astral.sh/ruff/

[15] (2024). [Online]. Available: https://flake8.pycqa.org/

[16] (2024). [Online]. Available: https://pylint.pycqa.org/

[17] [Online]. Available: https://github.com/psf/black

[18] (2024). [Online]. Available: https://jupyter.org/

[19] (2024). [Online]. Available: https://about.readthedocs.com/

[20] (2024). [Online]. Available: https://peps.python.org/pep-0621/

[21] (2024). [Online]. Available: https://packaging.python.org/en/latest/specifications/

[22] (2024). [Online]. Available: https://versioningit.readthedocs.io/