IS-ENES3 Deliverable D5.5

Style guide on coding standards

Reporting period: 01/07/2020 – 31/12/2021

Authors: Bouwe Andela (NLeSC), Niels Drost (NLeSC)
Reviewer(s): Carsten Ehbrecht (DKRZ), Valeriu Predoi (UREAD-NCAS), Javier Vegas-Regidor (BSC)
Release date: 20/12/2021

ABSTRACT

We developed a framework of coding standards, to improve code integration, code sharing between researchers, reusability, maintenance, and readability. These standards have been distributed to the community through dissemination activities and support material in the form of a style guide and contribution guidelines for ESMValCore and ESMValTool. The style guide is described in this report. These new standards have been implemented in the ESMValTool in WP9/JRA2. Applying the recommendations from this style guide to the ESMValTool has led to a dramatic increase in its popularity because the tool is now more user-friendly: it is easier to install and run, it runs faster and more reliably, and it has a more accessible community.

Revision table			
Version	**Date**	**Name**	**Comments**
1.0	19/11/2021	Bouwe Andela	Release for review
2.0	13/12/2021	Bouwe Andela	Version with review comments addressed

Dissemination Level		
PU	Public	X
CO	Confidential, only for the partners of the IS-ENES3 project	

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 824084

Table of contents

Executive Summary

When researchers employ transparency in their research - in other words, when they properly document and share the data and processes associated with their analyses - the broader research community is able to save valuable time when reproducing or building upon published results. Often, data or code from prior projects will be re-used by new researchers to verify old findings or develop new analyses.

To facilitate this process, a style guide has been developed specifically tailored for climate data analyses. The style guide is based on the extensive knowledge available in The Turing Way, an open-source, open collaboration, and community-driven handbook to reproducible, ethical, and collaborative data science.

Collaboratively developing ESMValTool version 2 has provided us with the experience required to tailor the generic advice from The Turing Way to the climate science community. Over the years, this has resulted in many discussions and a style guide for contributing to the ESMValTool that follows the advice presented in The Turing Way, as well as contributions of our experience to The Turing Way. Applying the recommendations from this style guide to the ESMValTool has led to a dramatic increase in its popularity because the tool is now more user-friendly: it is easier to install and run, it runs faster and more reliably, and it has a more accessible community.

1. Objectives of the style guide

1.1 Style guide for diagnostic development

To facilitate the development of climate data analysis or "diagnostic" code, it is useful to define a "style guide" with best practices for developing such code. Examples of topics that are covered by the style guide are code quality, documentation, and promoting collaboration. This will enable the re-use of existing analysis software, thereby avoiding duplicating effort and improving the quality of science by making the analyses more transparent to other scientists.

1.2 Implementation in the ESMValTool

To immediately apply the developed standard to existing diagnostics, several of the recommended best practices have been used to develop the ESMValTool software package. In addition to this, a style guide tailored to the ESMValTool diagnostics has been developed and is available in its documentation.

2. Methodology and Results

2.1 Style guide based on The Turing Way

"*The Turing Way handbook to reproducible, ethical and collaborative data science"* available at https://the-turing-way.netlify.app/welcome.html is a collaborative effort to develop a handbook with best practices for developing software for carrying out scientific analyses. This handbook formed the starting point for developing a set of best practices for developing climate data analysis scripts, as many points are generic to all scientific code, while other points are specific to the climate data analysis field. This resulted in a style guide containing a list of eleven recommendations for developing climate data analysis software. The style guide is available at https://iseneseval.github.io/diagstandards/.

The style guide refers the developer to existing community resources whenever possible (e.g. The Turing Way, PEP8), instead of creating new code formatting guidelines, documentation guidelines, etc. just for climate science. This is on purpose: it provides a readily available solution, and as these are existing standards that are kept up to date by the community, this makes the style guide more sustainable.

2.2 Implementation in the ESMValTool

Most of the 10 recommendations from the style guide have been applied to the ESMValTool software package as a whole, reducing the burden on diagnostic developers. Contribution guidelines have been developed for the ESMValTool and are available in the ESMValTool

documentation at https://docs.esmvaltool.org/en/latest/community/index.html. These guidelines contain specific recommendations that diagnostic developers can use to improve the quality of their code. Over the past four years, many of the recommendations from the style guide have been adopted by the ESMValTool community.

3. Conclusions and Recommendations

3.1 Increased popularity by following best practices

The ESMValTool community has adopted the recommendations from the style guide, leading to the increased popularity of the tool over the past few years. This increase in popularity is a consequence of the tool being more user-friendly: it is easier to install and run (it can be installed using popular package managers, instead of users needing to download the source code and copy it to an appropriate location), it runs faster (about twenty times faster for a representative use case) and more reliably (comprehensive automated tests that are run daily ensure reliability, instead of occasional human testing), and it has a more accessible community (communication takes place in public on GitHub as well as in publicly announced, open meetings, instead of via email between those concerned). This demonstrates that these best practices are useful recommendations.

3.2 Find the right balance between engineering and science

The style guide tries to strike the right balance between software engineering best practices and performing the scientific analysis process. Creating reliable, re-usable analysis software takes a lot more than doing the bare minimum to obtain the results needed to publish a paper. This balance has been found through the many discussions that we have had in the ESMValTool community on this topic. We have found that a modular design helps in making this tradeoff decision: parts of the software that are used and/or modified by many people benefit from rigorous engineering practices, while parts that are very specific to a certain analysis and only used by a few people benefit from giving scientists the freedom to quickly and easily make changes. In the ESMValTool community, we have therefore chosen to use extensive documentation and unit testing for the ESMValCore framework that runs the diagnostics provided by the ESMValTool, while for the diagnostic scripts we ask for just a few lines of documentation and we use peer review and regression tests for validation. For future evolution of the style guide, it is recommended to keep in mind that there is a balance to be found between rigor and quick results and that it does not need to be the same for all parts of the software.

4. Appendix: The style guide

To facilitate code integration, code sharing between developers, reusability, easy maintenance, and readability, it is recommended that these recommendations are followed when developing new diagnostic scripts.

It is recommended that you contribute your diagnostic script to an existing tool, such as the ESMValTool, as this makes it easier to find users and collaborators, thus increasing the chances that your software will survive long after you have moved on to something new and exciting. However, if you find that contributing to existing software does not work for you, you can start a new package. A convenient way to do so is by using a cookiecutter template that already implements many of the items recommended here, for example this python-template. Regardless of whether you are contributing to an existing project or starting a new project, it is recommended that you follow the guidelines in this document.

1 Make your code open-source

It is recommended that you make your code open-source from the start, to facilitate reproducible science and encourage potential collaboration. If you have reservations about making your code open source before publication, consider developing it in a private git repository and making it public once you have submitted your paper.

2 Add a license

It is critical that you share your code with a license because others will not legally be allowed to use it if it does not have a license. Make sure that any dependencies you use or plan to use are compatible with the license you choose. To make it easier for others to use your code, consider adding a license that is commonly used in the community that you are developing your code in and/or compatible with any tools that you may want to integrate it with. For example, when developing in R, you may want to choose an (L)GPL license, while the Apache 2.0 license may be a suitable choice for Python code. See choosealicense.com for an overview of the available options.

3 Use git for version control

It is recommended that you make use of a version control tool. In particular, it is recommended that you use git, because of all available version control systems it best facilitates (future) collaboration.

4 Make your code citable

In order to get scientific credit for your work, you may wish to make it easy to cite your code. Many papers ask for a DOI for your code these days. A convenient way to obtain that is by linking your GitHub repository with Zenodo as described in Make your code citable.

5 Choose a suitable programming language

Most mature programming languages come with a set of standards, tools, and services for developing and distributing software in that language. Some programming languages come with an extensive ecosystem of libraries that facilitate the analysis of climate data. It is recommended that you choose a programming language for your diagnostic script that has both of the above-mentioned features. At the moment, Python seems to be the most suitable language for analyzing climate data in general, mostly because of the availability of suitable libraries, but bear in mind that some specific communities can be heavily invested in other languages (like the predictions verification community is in R). You may want to do a library search for your specific problem to discover if your community is one of those.

6 Use libraries

Software that has been developed by a large and thriving community is usually more reliable and faster than developing your own solution. Therefore it is highly recommended that you make use of existing libraries and contribute to existing software instead of developing your own software from scratch.

Programming languages come with a large collection of built-in functions and libraries, called a standard library. It is recommended that you make use of functions from the standard library. An example of such a library is the Python standard library.

Climate data is typically stored in NetCDF format using the CF-Conventions. Usually, climate data is larger than the amount of memory available in a typical computer. A programming language that is suitable for analyzing climate data has libraries that support

- reading and writing NetCDF files or an equivalent storage format supporting the CF-Conventions
- understand the CF-Conventions
- computing statistics on arrays that are larger than memory
- creating figures

A good example of a programming language that meets all these requirements is Python, where the xarray or iris libraries facilitate the processing of large climate data, while matplotlib or altair can be used for visualization.

7 Follow code quality standards

It is recommended that you make use of the style guide that is available for the programming language of your choice. Examples of style guides:

- PEP8 and PEP257 for Python
- The Tidyverse style guide for R

If there is no style guide available for the programming language of your choice, you may want to reconsider that choice because the language may not be mature enough. If there are compelling reasons to choose a language without a style guide, it is recommended that you at least

- organize your code in multiple files of no more than a few hundred lines each
- organize your code in functions that fit your screen without scrolling (typically no more than 50 lines)
- use easy to understand variable names
- insert documentation or comments in your code that explain what is going on

This kind of organization increases the readability of your code, thereby reducing the risk of mistakes and making your code accessible to others.

Style guides typically come with automatic formatters, that can automatically format your code according to the chosen style. In addition to automatic formatters, tools for checking adherence to a style guide as well as for common programming mistakes are usually also available. These tools are called linters. It is recommended that you make use of an automatic formatter and linter. For example, a popular automatic formatter for Python is black, while a good linter for Python is pylint. It is recommended that you use an online service to host your code, so you can automate checking the quality of your code and even formatting it, should you wish to do so. Working with an automatic linter and formatter can be tedious and frustrating at the start. After a few weeks that goes off and the improvements in readability and reliability will be huge, especially if you are developing code with more colleagues. Please, be patient, and then you will see the results by yourself.

8 Keep your code reliable and maintainable

If you write large pieces of code, it is recommended that you implement some form of automated testing to ensure that you do not lose more and more time testing that your code still produces the

correct results as it grows. This can range from unit tests, that test each function in your code individually, to regression tests that run larger portions of code and just check that the changes you make to your code do not change previously computed results that are not supposed to change. There is software available that can keep track of which parts of your code are actually tested by the tests you have written, this is called coverage. This is helpful to see if it is safe to change existing code. For example, for Python there is Coverage.py.

If you have or plan on having examples in your documentation, you will want to test that those examples actually work right from the start of the project. It is important to start automated testing as soon as you write the first example because you will never find the time to start doing this later once a lot of (broken) examples have accumulated. For example, you can test example Jupyter notebooks, that are used to generate your documentation with nbmake and other Python code examples with doctest.

For most programming languages, tools are available to run all tests with a single command. A good choice for running Python tests is pytest.

If there are other things that you commonly check about your code or repository, such as code style, naming conventions, etc, keep in mind that it is often possible to save yourself a lot of time by automating these checks using the same test framework.

9 Write documentation

It is recommended that you write documentation for your code. For a small piece of software, adding a README.md file with some installation and usage instructions might be sufficient. For a larger software package, having a webpage with, for example, usage instructions, examples, tutorials, contribution guidelines, and a code of conduct is recommended. This could for example be set up by using mkdocs.

For software packages that can be used as a library, it is recommended that you use a tool that can build a documentation webpage from documentation included in the code. For Python projects, sphinx is a common choice. If there is guidance available for writing documentation in the programming language of your choice, it is recommended that you make use of that. For example, Python has Docstring Conventions as well as various styles for formatting documentation, such as numpy style. If there is no specific guidance for the programming language of your choice, it is recommended that you look at the documentation of some of the libraries that you enjoy using in that language and follow their example.

It is recommended that you host your documentation on a website.

10 Do code reviews

If at some point you find yourself working on code with multiple people, it is highly recommended that you do code reviews to improve the quality and reliability of your code. As a bonus, you will see that the review process will greatly improve your abilities as a coder.

11 Use online services

A large number of services are available online to make developing and distributing software easier. These services are usually free for open-source projects.

- Host your code on a website with an issue tracker

It is recommended that you develop your diagnostic script in public on a modern code development platform like GitHub or GitLab. This facilitates collaboration and has the added benefit that you can make use of services for automatically hosting documentation, automatic quality control, automatic testing, and automatic publication.

Hold all conversations about developments of your code in the issue tracker, to make your work accessible to users and potential collaborators. If you have a conversation where decisions are made outside the issue tracker, make sure to summarize it in an issue.

- Host your documentation on a website

The most convenient way to publish your documentation is through a webpage. It is recommended that you use the service for documentation that is the default for the programming languages you are using. For Python, readthedocs is a common choice.

- Use a package registry

It is recommended that you only use libraries that are available in the default package registry for the programming language of your choice, to ensure that others are able to easily install the dependencies required to run your software. Examples of package registries are the Python Package Index and the Comprehensive R Archive Network for Python and R packages respectively.

Likewise, it is recommended that you distribute your diagnostic code as (part of) a package in the default package registry.

- Use a code quality checker

Various services are available that can automatically find common mistakes in your source code. An example of such a service is Codacy.

- Automate testing

Many online services are available for running tests automatically, for example whenever you make changes or every night to see if your software still works as the libraries that your software depends on are updated. This is called continuous integration. Examples of such services are GitHub Actions, GitLab CI, or CircleCI.

It is recommended that you use a code coverage reporting service, such as Codecov or Coveralls to keep track of code coverage. By making the code coverage visible, everyone working on the code will feel encouraged to improve code coverage, keeping your code reliable and maintainable.