Improving METplus Test Infrastructure and Code Quality

2024 DTC Visitor Report

John Sharples

Bureau of Meteorology, Australia

DTC Host: George McCabe

Background

When applications undergo continuous development for many years the complexity of the codebase greatly increases. This leads to a much greater chance of bugs, design issues, or redundant code, all of which introduces technical debt, making it harder for developers to understand, maintain, and extend the software in the future. The best guard against these types of issues is code review and a comprehensive test suite.

Unit testing ensures that each component behaves as expected, reducing the likelihood of introducing bugs, and guarding against unintended future changes. The DTC development team proactively works to improve unit testing throughout all their repositories, however due to historical factors, the age of the repositories, and the quantity of contributors, testing is an area that requires significant uplift.

This project aimed to broadly improve testing across the METplus Analysis Tools software suite, specifically METdataio, METplotpy, and METcalcpy. The project priority was to improve test coverage and testing infrastructure, with secondary aims to improve code quality and maintainability. The priority repository was METdataio, which was identified as having very poor test coverage. METplotpy and METcalcpy were identified as secondary priorities.

This project followed on from work completed in 2023, where the BoM made an in-kind contribution, delivering a 20% increase to testing code coverage in the main METplus Wrappers repository.

Project Outcomes

Below are listed the main outcomes from this project. Overall, the project was very successful in its primary goal of improving testing and maintainability in METdataio, and delivered some benefits to both METplotpy and METcalcpy. The project also delivered some general maintainability to all repositories. Lastly a number of issues were found to impact on the ability to test these repositories. These issues, and possible solutions are outlined as possible future work.

Aside from requirements gathering and some small review tasks, all deliverables were captured as GitHub pull requests. For specific details of each change refer to the relevant GitHub repository, and filter for commits from user John-Sharples.

Improvements to all repositories

Some changes were made across all repositories. These typically aim to improve how the packages are installed and how tests are run, delivering more reliable test coverage reporting.

- Implemented *pyproject.toml* in all repos. This avoids deprecated uses of *setup.py* for package installation and prevents repo clutter by implementing a single file for all tool configuration.
- Defined the source code that requires testing. For example, in some repos there are "contributed" modules which are not treated as core source code. In discussion with DTC developers, I developed rules for code coverage reporting and implemented these in the *pyproject.toml* for each repo.
- Implemented or extended the use of a *conftest.py* file for common test infrastructure.
- Conducted a review of the use of logging and error handling.
- Improved test coverage for all repos - see details below.
- Several bugfixes and documentation improvements, which are impractical to list here.
- Standardised many instances of logging.

METdataio

The priority for this project was the METdataio repository, especially the METdbLoad module, which had no reliable testing. Below are the major improvements made to this repo.

- Lift test coverage for METdataio from 0% to 85%.
- Implement a repeatable method for running tests on all modules in METdataio.
- Implement testing for METdbLoad, which previously had no consistent testing.
- Extend GitHub Actions Workflows to run tests automatically for PR and merges to METdataio.
- Setup infrastructure for implementing and managing a test database, both locally and in GitHub Actions.
- Expand the test data to cover outputs of type RHIST, VSDB, MTD, and MODE.

- Add test infrastructure to *conftest.py* which handles access to test data, and creation of temporary test files.
- Adopt the use of common logger functionality across all modules in METdbLoad
- Created testing documentation to guide future contributions. This details how to set up the test database for local development.

METplotpy

Improvements made to METplotpy focused on test infrastructure, facilitating future test development. There was limited increase in test coverage as the existing test coverage was quite good, and there were difficulties in wide scale test expansion (see Recommended Future Work).

- Make the use of *CompareImages* test functionality configurable via environment variable. While useful, this functionality has long been problematic as it does not work reliably across different operating systems. This work was ultimately rolled back due to undocumented downstream dependency issues (for more details see Recommended future work).
- Add test infrastructure to support comparison of *plotly* plots using json representations. This provides an alternative to *CompareImages*, but only applies to plots that use *plotly*.
- Add test infrastructure to create temporary netCDF files to use as test data.
- Add fixture to set working directory for tests, allowing simplification of repeated test setup code.
- Expanded test coverage to include currently untested code, raising the test coverage from 79% to 81%. Note these figures compare only coverage of the core METplotpy code and do not reflect changes to the reported coverage by, for example, excluding "contributed" modules.

METcalcpy

Improvement to METcalcpy mostly focused on expanding the test coverage to previously untested modules.

- Add example of test data management via *utils.py*
- Expanded test coverage to include currently untested code, raising the test coverage from 43% to 55%. Note these figures compare only coverage of the core METcalcpy code and do not reflect changes to the reported coverage by, for example, excluding "contributed" modules.

Recommended Future Work

In delivering this project, several issues were identified which relate to code maintainability, but fall outside the scope of this project. Some areas of further work are outlined below. These address issues that were encountered while completing this project.

1. **Maintain and enforce package dependencies**

   The issue with *CompareImages* described above had to be rolled back after it was discovered a dependency could not be imported by a downstream system at NOAA. In effect, there was an undocumented "shadow" requirement to use only certain python modules. This could be avoided by better maintenance of the listed Python requirements, and associated GitHub Actions to check imports.

2. **Reduce code with import issues**

   Some modules in METcalcpy (and possibly other repos) are untestable as they cannot be imported. For example, `metcalcpy/util/read_file.py` has a requirement to use METdataio, but METdataio is not installed in GitHub Actions, meaning no Pull Request can import this module and pass the pipelines. Similarly, `metcalcpy/pre_processing/waves.py` requires the package *xrft*, which is also not a listed requirement. This issue also relates to Python package dependencies and could be solved by adding these dependencies. Alternatively, these modules could be refactored or deprecated.

3. **Reduce untestable code**

   Some modules in METplotpy (and possibly other repos) are untestable because the code cannot be reached. Modules in `metplotpy/plots/tcmpr_plots/scatter/` are

explicitly excluded from the supported plot types (see `TcmprConfig`), and result in an error. While this could be overcome by mocking the supported plot types, it's unclear why these plots have been excluded, and if writing tests against their current implementation is useful. A better approach would be to undertake the work required to support these plot types, and write appropriate unit tests at that time. Otherwise, these modules could be deprecated and removed from the codebase.

4. **Deprecate unused code**

   In writing tests for METcalcpy, it became apparent that some code while testable is almost certainly not used. For example, `metcalcpy/util/mode_3d_volrat_statistics.py` had a bug in the function *rename_column* (see here). As every other function in this module called *rename_column*, all of them return None instead of a numeric result. The git history suggests this bug has existed since the code was added, 5 years ago. As such it seems improbable that anyone is using this module.

   This bug has been fixed, and tests added for this module in line with the goals of this project. However, if DTC were to adopt a policy of deprecating obviously unused code, this module could be safely deprecated and removed. This has the added advantage of improving the long term maintainability of the codebase.

Acknowledgments

Special thanks to George McCabe for hosting my visit and filling out my social calendar. Special thanks also to Minna Win-Gildenmeister for all the code reviews and hiking recommendations. Thanks also for the friendly support from John Halley Gotway, Julie Prestopnik, Molly Smith, Michelle Harrold, Jenny Bolton, Louisa Nance, Weiwei Li, and everyone at the Foothills Lab. I deeply appreciate the generosity and effort from everyone involved in the DTC Visitor Program for allowing me this opportunity and experience.