doi:10.1017/S1431927619001429
Microsc. Microanal. 25 (Suppl 2), 2019
© Microscopy Society of America 2019

# Open Source Development Tools for Robust and Reproducible Electron Microscopy Data Analysis

Magnus Nord[1*] and Johan Verbeeck[1]

[1.] EMAT, University of Antwerp, Antwerp, Belgium
* Corresponding author: Magnus.Nord@uantwerpen.be

Much of today's material science research relies heavily on advanced data processing techniques. This is made possible by the continued exponential growth in computing power available in even modest desktop computers, and method advances in computer science. If we take the case of electron microscopy, this has allowed for things such as electron tomography reconstruction and spectroscopic processing. Relying on the same development trend in semiconductors, also detectors and instrumentation control have greatly improved. In electron microscopy this is e.g. obvious in recent developments of fast pixelated detectors enabling new techniques such as pixelated/4D scanning transmission electron microscopy [1], where individual raw datasets can easily exceed 100 GB, much larger than most desktop computer's available working memory.

Such increased datasets call for new and better analysis tools, which often requires experimentalists with little or no a background in software development to write complicated code [2]. Not surprisingly, this leads to many issues, as the used algorithms are often complicated numerical routines, the code expands beyond the capability of the single programmer and unexpected behaviour creeps in. The situation can get worse when several people are collaborating, where emailing code between each other quickly becomes unsustainable. In addition, with an increasing amount of collaborators, care must be taken so that the functionality doesn't unintentionally break in subtle ways: a change in the code in one part of the software can have unintended consequences elsewhere. This is a known issue in the software development community and a wide range of tools is used to stay ahead of the chaos. The open software movement played a key role in this evolution and the developments of the toolset, perhaps because out of necessity to deal with organically changing teams of volunteers, not unlike the academic world.

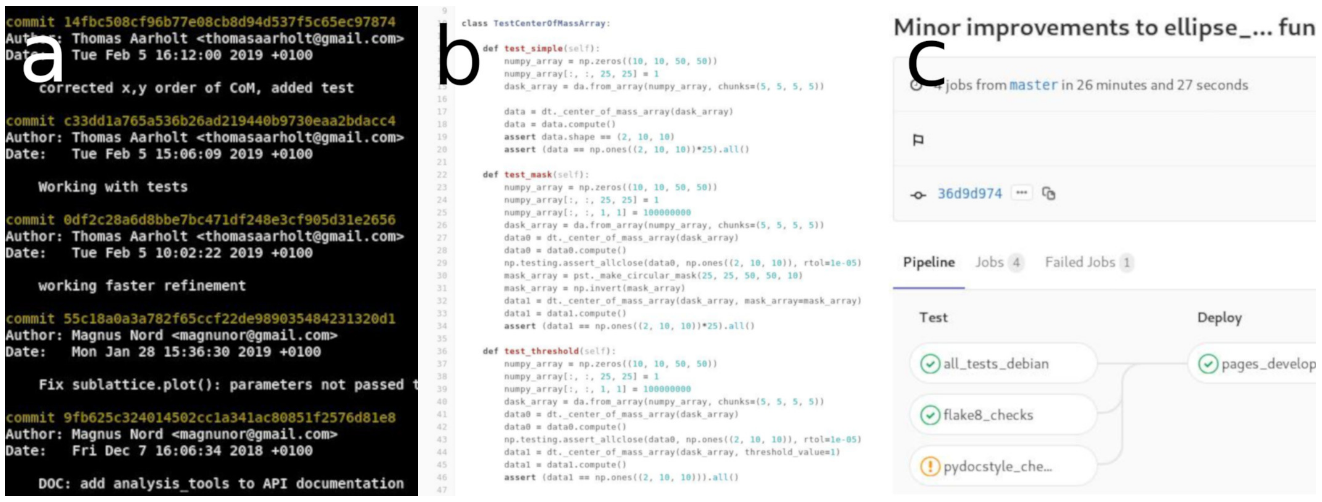
Another recent trend is the increased focus on open science, driven by the observation that the reproducibility of scientific experiments, the cornerstone of science itself, is in a bad state of affairs. This is also related to the ever increasing complexity and cost of high end instrumentation, data acquisition and analysis. The open science trend, following lessons learned from open software development, posits that both the result and the tools used in scientific research should be transparent, reproducible and publicly available. For the aforementioned analysis tools, this means the source code should be available under open source licenses, so that other researchers can both inspect, reuse, and even possibly contribute to the code. Especially the latter would not only enable more efficient use of researcher's time through less duplication of work, but also lead to less bugs due to more eyes being on code. Combining this with access to the raw data, it is possible to replicate all parts of the data processing.

Thus, today's complex data analysis tools needs to have clean code, to make it easy to both understand and contribute to it. Robustness, so the functionality doesn't break. Be well documented, so people can easily understand the functionality. However, as few microscopists have backgrounds in software development, the knowledge on how to do such thing are fairly low in the microscopy community.

In this talk we want to provide an overview of these development tools and how they could be of inspiration for experimentalists that want to develop or contribute code. It will focus on the examples Atomap [3] and pixStem [4], which are both Python libraries based on HyperSpy [5]. It will cover tools such as version control, tests, documentation tools and lastly how to tie all of this together using automatic testing systems. The end result being better and low-maintenance software projects. We will comment on how to work with large distributed teams and show examples how such codes also help to make open science a reality.

References:

[1] D. McGrouther et al., Microscopy and Microanalysis (2015), p. 1595.
[2] G. Wilson et al., PLOS Computational Biology (2017), p. 1.
[3] M. Nord et al., Advanced Structural and Chemical Imaging **3** (2017), p. 9.
[4] pixStem, https://pixstem.org (accessed February 21, 2019).
[5] F. de la Peña et al., HyperSpy (2018), DOI: 10.5281/zenodo.1469364



```
commit 14fbc508cf96b77e08cb8d94d537f5c65ec97874
Aut   : Thomas Aarholt <thomasaarholt@gmail.com>
Dat      Tue Feb 5 16:12:00 2019 +0100

    corrected x,y order of CoM, added test

commit c33dd1a765a536b26ad219440b9730eaa2bdacc4
Author: Thomas Aarholt <thomasaarholt@gmail.com>
Date:   Tue Feb 5 15:06:09 2019 +0100

    Working with tests

commit 0df2c28a6d8bbe7bc471df248e3cf905d31e2656
Author: Thomas Aarholt <thomasaarholt@gmail.com>
Date:   Tue Feb 5 10:02:22 2019 +0100

    working faster refinement

commit 55c18a0a3a782f65ccf22de989035484231320d1
Author: Magnus Nord <magnunor@gmail.com>
Date:   Mon Jan 28 15:36:30 2019 +0100

    Fix sublattice.plot(): parameters not passed t

commit 9fb625c324014502cc1a341ac80851f2576d81e8
Author: Magnus Nord <magnunor@gmail.com>
Date:   Fri Dec 7 16:06:34 2018 +0100

    DOC: add analysis_tools to API documentation
```

```python
class TestCenterOfMassArray:

    def test_simple(self):
        numpy_array = np.zeros((10, 10, 50, 50))
        numpy_array[:, :, 25, 25] = 1
        dask_array = da.from_array(numpy_array, chunks=(5, 5, 5, 5))

        data = dt._center_of_mass_array(dask_array)
        data = data.compute()
        assert data.shape == (2, 10, 10)
        assert (data == np.ones((2, 10, 10))*25).all()

    def test_mask(self):
        numpy_array = np.zeros((10, 10, 50, 50))
        numpy_array[:, :, 25, 25] = 1
        numpy_array[:, :, 1, 1] = 100000000
        dask_array = da.from_array(numpy_array, chunks=(5, 5, 5, 5))
        data0 = dt._center_of_mass_array(dask_array)
        data0 = data0.compute()
        np.testing.assert_allclose(data0, np.ones((2, 10, 10)), rtol=1e-05)
        mask_array = pst._make_circular_mask(25, 25, 50, 50, 10)
        mask_array = np.invert(mask_array)
        data1 = dt._center_of_mass_array(dask_array, mask_array=mask_array)
        data1 = data1.compute()
        assert (data1 == np.ones((2, 10, 10))*25).all()

    def test_threshold(self):
        numpy_array = np.zeros((10, 10, 50, 50))
        numpy_array[:, :, 25, 25] = 1
        numpy_array[:, :, 1, 1] = 100000000
        dask_array = da.from_array(numpy_array, chunks=(5, 5, 5, 5))
        data0 = dt._center_of_mass_array(dask_array)
        data0 = data0.compute()
        np.testing.assert_allclose(data0, np.ones((2, 10, 10)), rtol=1e-05)
        data1 = dt._center_of_mass_array(dask_array, threshold_value=1)
        data1 = data1.compute()
        assert (data1 == np.ones((2, 10, 10))).all()
```

Minor improvements to ellipse_... fun

4 jobs from master in 26 minutes and 27 seconds

36d9d974

**Figure 1**. (a) Example of version control software, showing individual code commits from Atomap. (b) Unit tests from pixStem, which is used to make sure the implementation of the center of mass function works correctly. (c) Automatic running of the unit tests and deployment of documentation using GitLab's continuous integration system.