Foundations of Research Software Publication - Notes and Questions

> You can find further information in the workshop materials (https://gitlab.com/hifis/hifis-workshops/make-your-code-ready-for-publication/workshop-materials).

1) Put your code under version control

2) Make sure that your code is in a sharable state

How to deal with the risk of old/deprecated packages?

- Usually, it requires a steady minimal effort to maintain a software. Dealing with dependencies and updating them is one part of this work.
- With regard to reproducibility, you might want to take additional steps by "conserving" the runtime environment as well. E.g., by using containers (https://carpentries-incubator.github.io/docker-introduction/reproduciblity/index.html) or services such as Binder (https://mybinder.org/). But this depends on the level of reproducibility you want to achieve. In this regard, the Code Provenance chapter (https://merely-useful.tech/py-rse/provenance.html#provenance-code) of the book Research Software Engineering with Python (https://merely-useful.tech/py-rse/index.html) briefly discusses the topic "Reproducibility versus inspectability".

To avoid all the "reinventing of wheels" about that - Why not make the jump to an IDE (which takes care of environment etc.) right away?

- Integrated Development Environments (IDEs) create their own domain of issues because of their user interface. Usually, you want to use a solution which allows you the automation of certain tasks without manual user interaction.
- In the example, we only used a `requirements.txt` file to capture the directly required libraries. By running `pip install -f requirements.txt` you can automatically install all these required libraries. This is one of the simplest solutions to problem. There exist other solutions as well. For example, python-poetry (https://python-poetry.org/) which is more a pure Python solution for a package manager. Another option are Conda environment files (https://docs.conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html#create-env-file-manually).

Could you share a good resource for getting started w/ test automatization? Thanks!

- In HIFIS, we offer a workshop focusing on Test Automation with Python (https://gitlab.com/hifis/hifis-workshops/test-automation-with-python/workshop-materials). See also the workshop events here (https://events.hifis.net/category/4/).
- The book Research Software Engineering with Python (https://merely-useful.tech/py-rse/index.html) also offers a nice testing chapter (https://merely-useful.tech/py-rse/testing.html).

3) Add essential documentation

Would you put an explanation of algorithms used in the readme?

- If this is an important part of what a user needs to understand, I would add documentation about it. But maybe it is better to put it in a separate user guide that you can link to from the `README`. The user guide would also be part of your project. For example, you could write documentation using Markdown and use a documentation generator such as Sphinx (https://www.sphinx-doc.org/en/master/) to generate your documentation in PDF or HTML format.

4) Add a license file

Are you sure that legal departments of Helmholtz centers are prepared to answer software licensing questions?

- Not really sure but we have to improve in Helmholtz on this aspect, if we want to push more in the direction of Open Science / Open Source. Currently, I would recommend to contact the HIFIS Software Engineering Consulting (https://hifis.net/services/software/consulting.html) in case of questions.

5) Make your code citable

6) Release your code

This was a quite detailed description of all (many?) of the do's and dont's of SW publication. I wonder how that would be received by, say, a biologist considering switching from Excel to "real" programming, or just wishing to go away from that proprietary tool. Intimidating, I would think. Is there any way that IT support could provide them a preconfigured environment to simplify the transition (e.g. Jupyter plus a number of templates)?

- I think there are different aspects in this questions.
- Switching from one tool to the other, for example from Excel to Python, can be already an improvement for reproducibility in itself. For example, Python makes it much simpler to test/validate your results.

- The other aspect is publishing an artefact in a way to improve reproducibility of the overall scientific results. Here I would argue that one has to do many of the things discussed for publishing the Excel file as well. For example, writing documentation or adding a way to persistently link it from the scientific publication. Even copyright and licensing still play a role here.
- But in the end, we need to simplify the publishing process and available infrastructure (e.g., platforms such as GitLab, build pipelines) and ready-to-use project templates will definitely help here.
- With regard to templates, the tool cookiecutter (https://www.cookiecutter.io/) and already available templates (https://www.cookiecutter.io/templates) might be worth a look.