5. Software Engineering best practices

Version Control: git

What is this?

A tool to allow you to track and revert changes, and collaborate with others (change management).

Why is this important?

- Allow versioning of the code and continuing functionality.
- Allow simultaneous changes to the same files.
- Fundamental for reproducing historic states in the line of development.
- Development follows a story and allows other users to build confidence in your work.

Version Control in practice: git

- Create a repo on GitHub
- Clone the repo to your local machine
- Checkout a branch and make changes
- `git add`, `git commit` and `git push`: IDEs such as VSCode make it easy for you!
- Observe how your repo changes on GitHub
- GitHub offers great learning labs: GitHub skills https://skills.github.com/
- roadmap.sh offers a git roadmap: https://roadmap.sh/git-github

Development workflows: GitHub-flow

What is this?

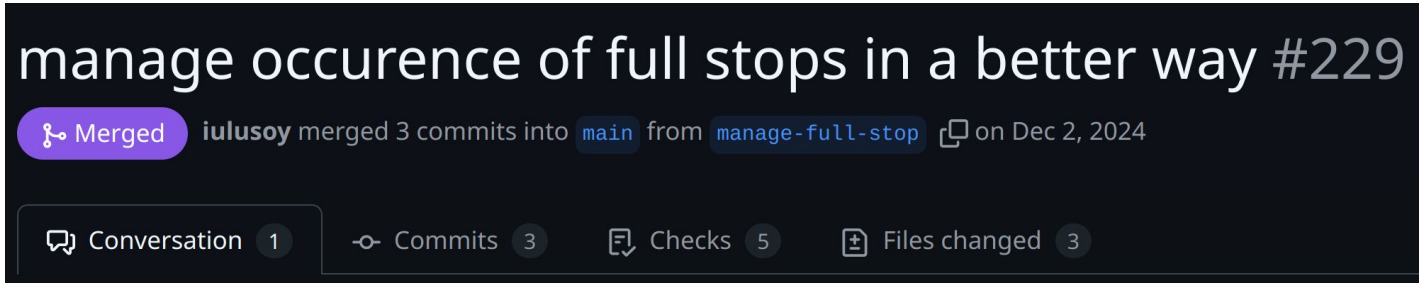
GitHub-flow is a lightweight workflow with creating branches, making changes, opening Pull Requests, running Continuous Integration, and requesting code review, together with Issues and Kanban project boards.

Why is this important?

- Manage how changes are incorporated in the software.
- Track progress in your project and highlight bottlenecks.
- Adhere to development guidelines, ensuring the implementation follows defined rules to ensure software quality.

Development workflows in practice: GitHub-flow

- After you made changes in your git branch, open a Pull Request on GitHub
- Observe how the PR highlights the changes in the line of development
- You can link issues, comment on the PR and run automated checks
- See GitHub skills https://skills.github.com/
- See roadmap.sh https://roadmap.sh/git-github



Requirements engineering and continuous delivery

What is this?

Requirements engineering is the process of translating stakeholder requirements on the research software into defined tasks. Early delivery and iteration over it allows refinement of the requirements and tasks.

Why is this important?

- Ensure that the software fulfills its purpose. In research software, requirements engineering is closely intertwined with the research process and subject to frequent changes.
- Understand the problem the software should solve and map this onto an efficient technically feasible solution considering all constraints.
- Allows prioritization of requirements/tasks and decision-making. Decisions should be documented together with the requirements.

Requirements engineering in practice

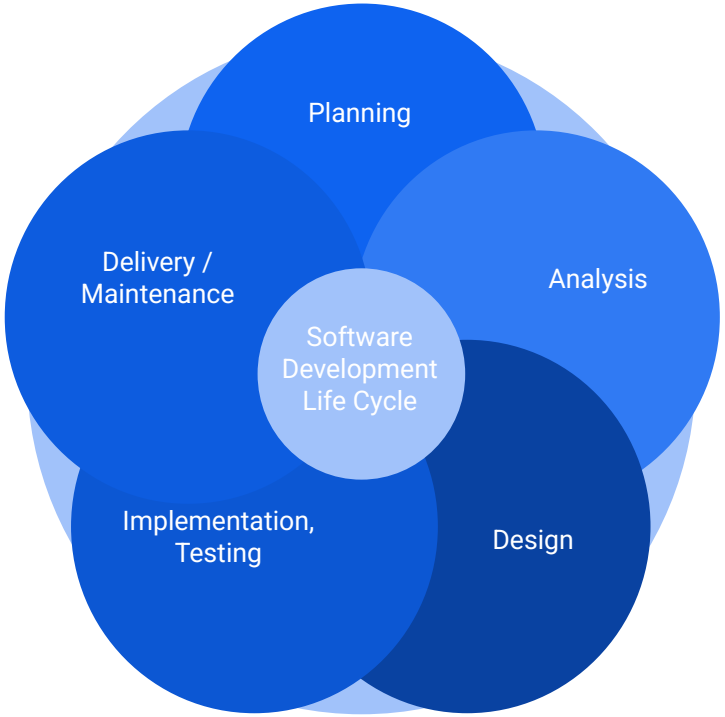
- Functional requirements (what the software should do: features)
- Non-functional requirements (how the software performs a task: ie performance, security)
- Domain requirements (specific to the domain: ie. Healthcare)
- Use tools such as draft.io or miro to gather requirements

As a <type of user>, I want <some goal> so that <some reason>.

As a <researcher>, I want <to obtain the probability of a class> so that <I can classify incoming images>.

Continuous delivery in practice

- Deliver early to find out if your software is fulfilling its purpose/moving in the right direction
- Use quality control to allow early delivery through the main branch in your GitHub-flow (*keep your main branch operational at all times*)
- Following agile / lean principles
- Use git to allow consistent usability of your software



Project management: Kanban boards

What is this?

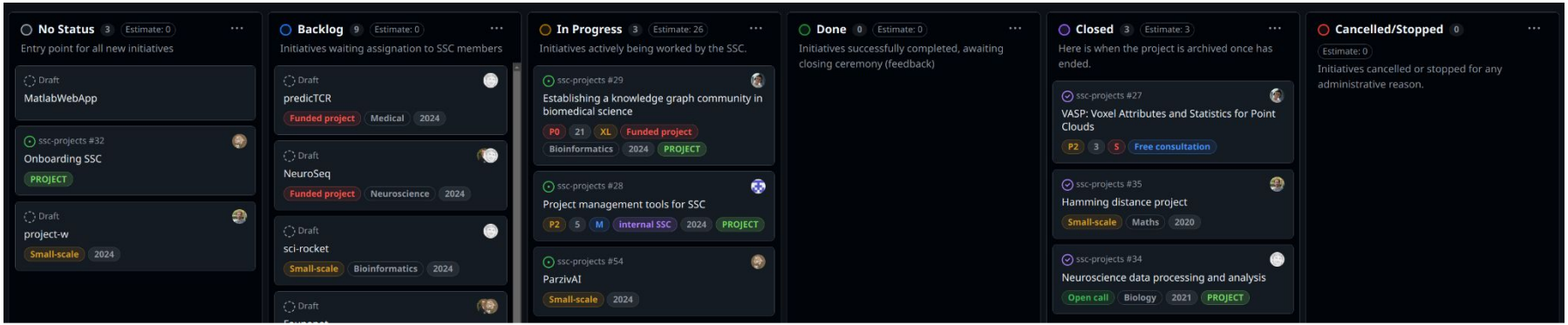
Project management is used to track progress, identify intertwined or dependent processes, and allows visual access to the project's status.

Why is this important?

- Ensure that the software development moves in the right direction.
- Increase the flow of ongoing work.
- Allow prioritization of tasks and understand interdependencies and bottlenecks in the development.

Project management in practice

- Use a Kanban board to organize tasks
- Separate tasks into backlog/todo, in progress, done
- Prioritize tasks and assign contributors/identify necessities
- For example, GitHub projects



Project planning: Architecture and design

What is this?

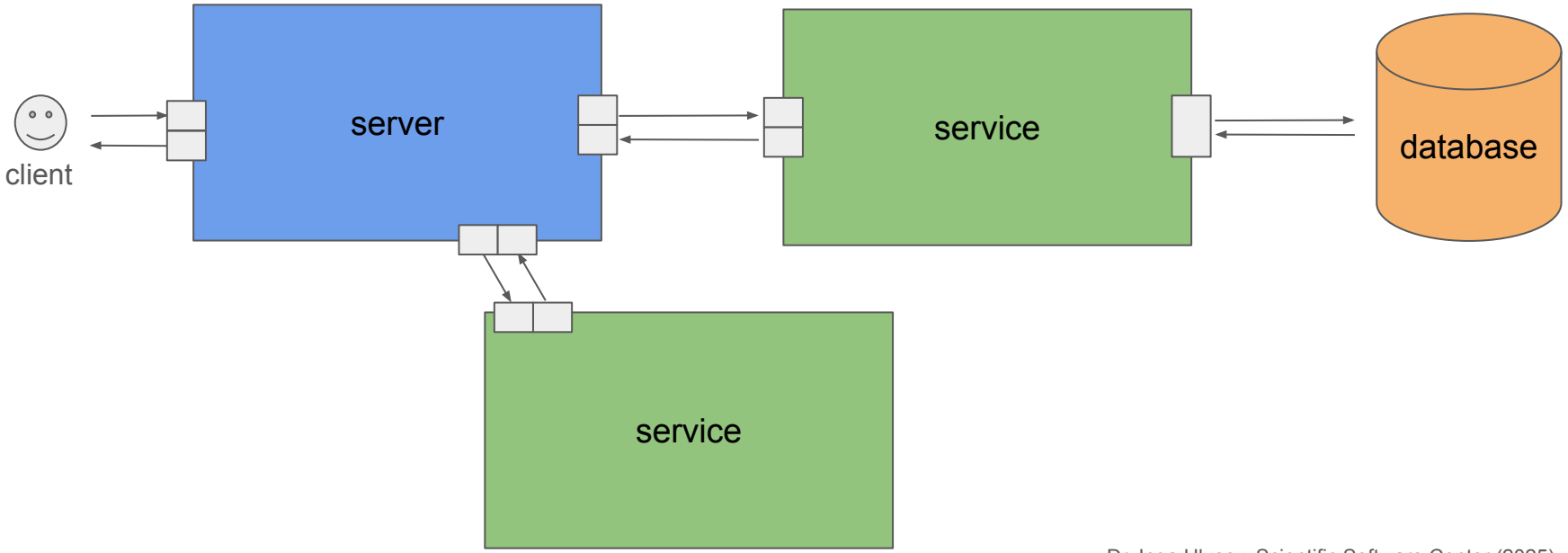
Software architecture describes how the system is composed of different pieces, and the interplay of the components. Design refers to the actual implementation of the requirements in the system as a whole and the different components.

Why is this important?

- Makes the software efficient and allow re-use of functionalities.
- Allows extensions and additions of features at a later stage without major refactoring.
- Makes the software maintainable.

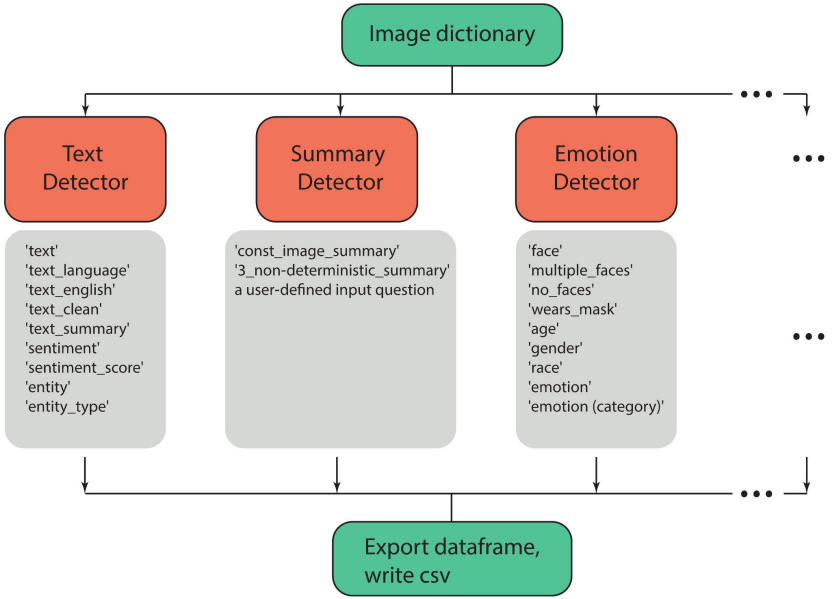
Architecture in practice

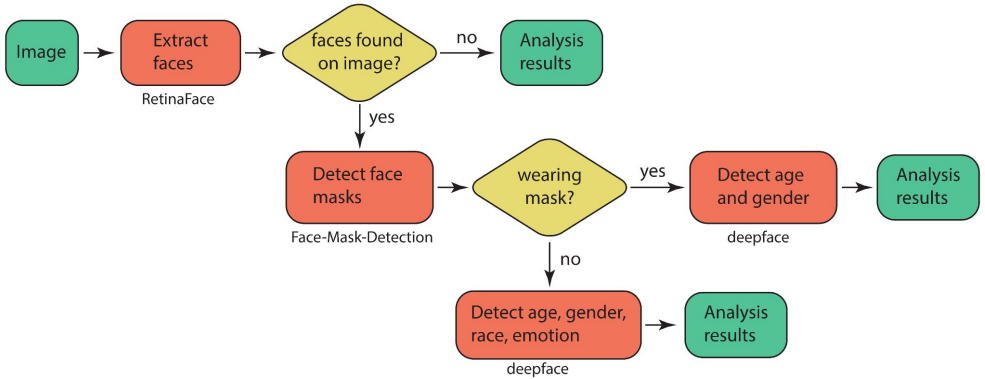
- Use (black/white) box diagrams to identify components and their interactions
- https://roadmap.sh/software-design-architecture / miro / draft.io / draw.io



Design in practice

- Map the input/output, data formats, transformations / logic / processes
- https://roadmap.sh/software-design-architecture / miro / draft.io / draw.io






```
image_summary_vqa_detector = ammico.SummaryDetector(image_dict,
    analysis_type="summary_and_questions",
    model_type="base")
for key in image_dict.keys():
    image_dict[key] = image_summary_vqa_detector.analyse_image(
        subdict=image_dict[key], analysis_type="summary_and_questions",
        list_of_questions = list_of_questions)
```

Quality management: Testing and continuous Integration

What is this?

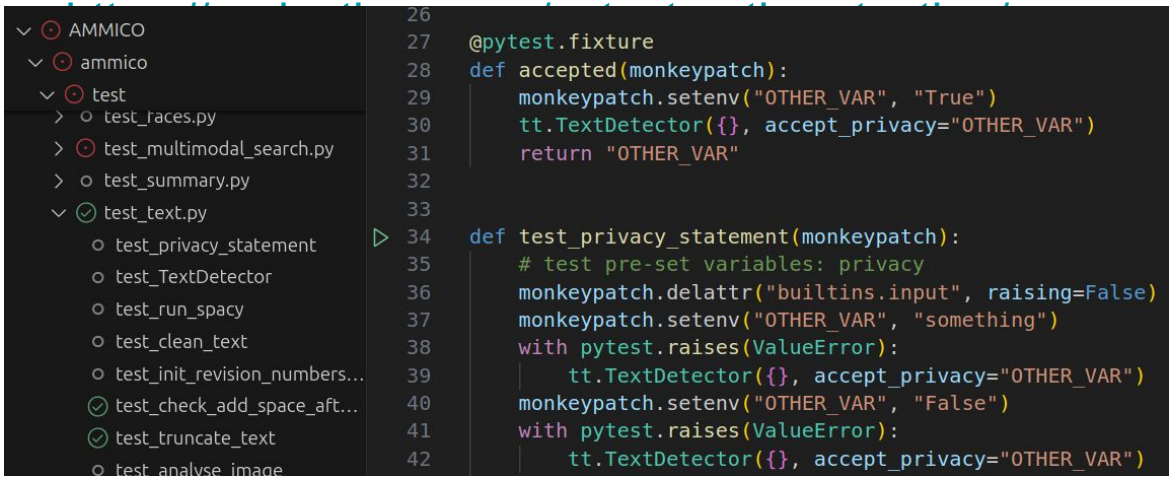
GitHub-flow is a lightweight workflow with creating branches, making changes, opening Pull Requests, running Continuous Integration, and requesting code review, together with Issues and Kanban project boards.

Why is this important?

- Manage how changes are incorporated in the software.
- Track progress in your project and highlight bottlenecks.
- Adhere to development guidelines, ensuring the implementation follows defined rules to ensure software quality.

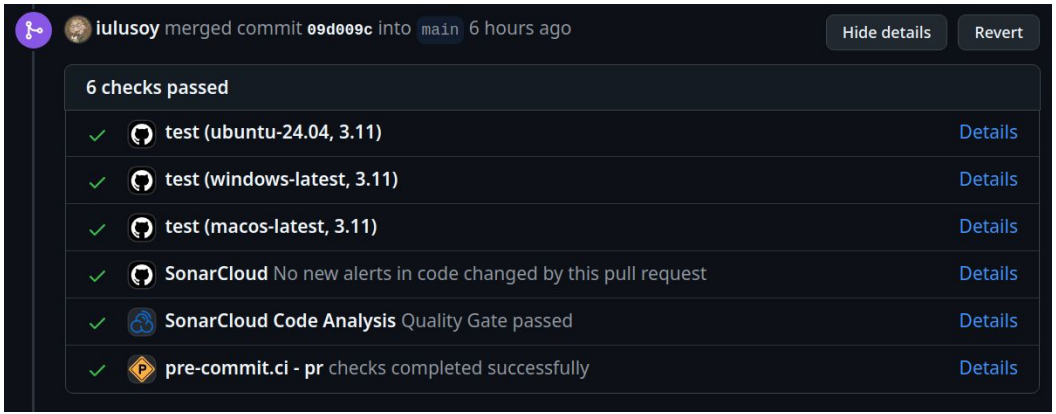
Testing in practice

- Use testing frameworks such as `pytest`
- Write tests in a `tests/` folder: unit tests, integration tests, system tests, compatibility tests, …
- To learn how to use pytest: https://docs.pytest.org/en/stable/,



Continuous integration in practice

- Set up your tests to be automatically run by GitHub actions
- Include code linter and quality control in your actions
- These should be set up to run automatically when you open a Pull Request
- GH actions, `codecov`, `sonarcloud`, `snyk`, `pre-commit`, code formatting (`black`), GitHub Guardian, `dependabot`



Software Management Plans

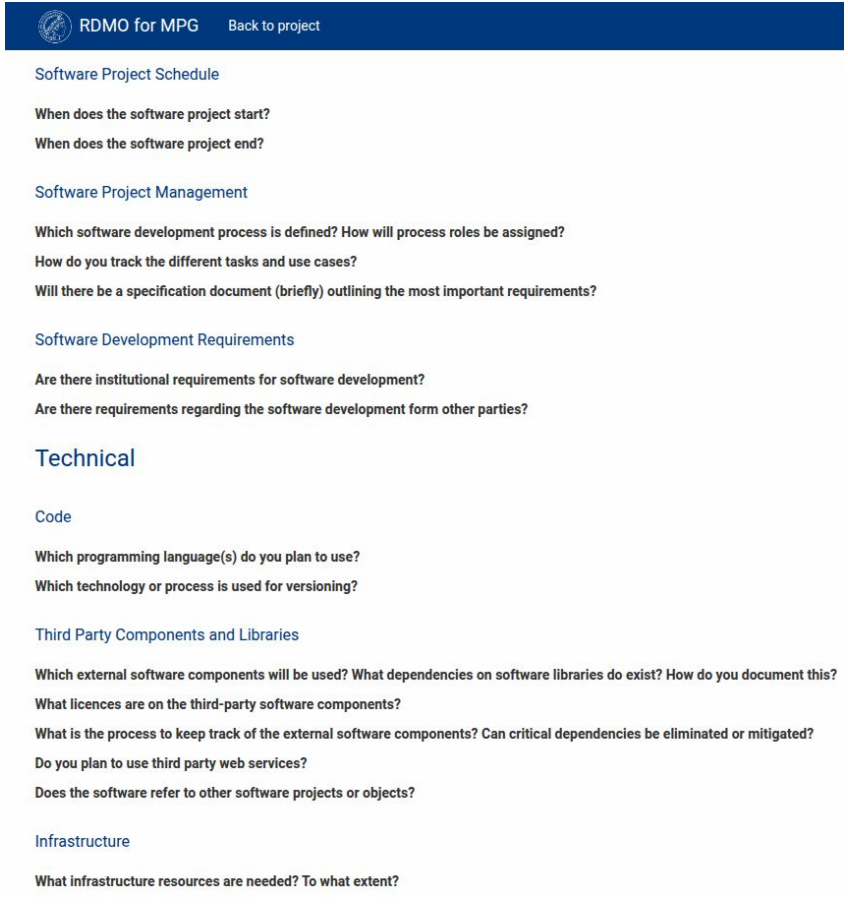
What is this?

Software Management Plans (SMPs) help to identify goals and the means required to pursue the goals in practice.

Why is this important?

- Identify criticality and required maturity of your software.
- Identify which measures are needed to ensure compliance of your software with the intended goals.
- Quantify milestones and tools for the intended purpose.

SMPs in practice



- Use the SMPs provided by the Max Planck digital library
- Helps you with your requirements and project management
- https://rdmo.mpdl.mpg.de/

Documentation

What is this?

Documentation can be comments, docstrings, readme's, tutorials, demonstration notebooks, and contains technical and domain-specific / application-specific descriptions of the software.

Why is this important?

- Document what the software can and cannot do, and parameter ranges.
- Allow others to install and use your software (or yourself, at a later time).
- Allow others to contribute to your software.

Documentation in practice

text module

class **text.PostprocessText**(*mydict: dict | None = None, use_csv: bool = False, csv_path: str | None = None, analyze_text: str = 'text_english'*)

Bases: `object`

analyse_topic(*return_topics: int = 3*) → tuple

Performs topic analysis using BERTopic.

Parameters: **return_topics** (*int, optional*) – Number of topics to return. Defaults to 3.
Returns: **tuple** – A tuple containing the topic model, topic dataframe, and most frequent topics.

get_text_df(*analyze_text: str*) → list

Extracts text from the provided dataframe.

Parameters: **analyze_text** (*str*) – Column name for the text field to analyze.
Returns: **list** – A list of text extracted from the dataframe.

get_text_dict(*analyze_text: str*) → list

Extracts text from the provided dictionary.

Parameters: **analyze_text** (*str*) – Key for the text field to analyze.
Returns: **list** – A list of text extracted from the dictionary.

- Use tools like `sphinx` and `mkdocs` to render docstrings and markdown at minimal effort
- Include jupyter notebooks that showcase use of your software - these can be run on google colab
- Document dependencies in a requirements file and provide installation instructions

Deployment: Runtime environment / containerisation

What is this?

Deployment information such as runtime environments or containers allow easy adaption as they provide direct access to running the software without installation and dependency conflicts.

Why is this important?

- A big step towards reproducibility and transferability of your approach.
- The software ecosystem changes quickly, and this allows to preserve a snapshot that can be shared and run easily.

Containerisation in practice

- Use docker to provide build instructions for containers, and possibly deploy the containers on Dockerhub for anyone to download and use
- Docker roadmap https://roadmap.sh/docker, official tutorial https://docs.docker.com/

Dockerfile > FROM

```
1  FROM jupyter/base-notebook
2
3  # Install system dependencies for computer vision packages
4  USER root
5  RUN apt update && apt install -y build-essential libgl1 libglib2.0-0 libsm6 libxrender1 libxext6
6  USER $NB_USER
7
8  # Copy the repository into the container
9  COPY --chown=${NB_UID} . /opt/ammico
10
11 # Install the Python package
12 RUN python -m pip install /opt/ammico
13
14 # Make JupyterLab the default for this application
15 ENV JUPYTER_ENABLE_LAB=yes
16
17 # Export where the data is located
18 ENV XDG_DATA_HOME=/opt/ammico/data
```

Software Licensing

What is this?

A software license states the terms of use, re-use and distribution, among others, without violating copyrights, and defines responsibilities.

Why is this important?

- So that others may use your code, and to prevent misuse.
- So that others may contribute to your code.
- So that the responsibilities for how the software is used are clear.
- Establishes the rights of all parties involved with the software.

Software licensing in practice

- Use the provided templates from GitHub: Either at repository creation or when adding a new file called LICENSE
- Permissive open-source license: BSD 2-Clause, MIT, Apache License 2.0
- Copyleft open-source license: GNU version 3, LGPL
- Proprietary licenses: Do not only keep your project close-source and potentially less visible, but also carry responsibilities for contract fulfillment
- https://opensource.org/licenses, https://choosealicense.com/
- **When using/incorporating third-party software**: ie. use of open-source libraries - is the third-party code distributed with your software? If so, compatibility needs to be confirmed!