Improving software quality in bioinformatics through teamwork

Katalin Ferenc[1], Ieva Rauluseviciute[1], Ladislav Hovan[1], Vipin Kumar[1], Mariekie Kuijjer[1], and Anthony Mathelier[1,2,✉]

[1] Centre for Molecular Medicine Norway (NCMM), Nordic EMBL Partnership, University of Oslo, 0318 Oslo, Norway
[2] Department of Medical Genetics, Institute of Clinical Medicine, University of Oslo and Oslo University Hospital, Oslo, Norway

✉ Correspondence: Anthony Mathelier <anthony.mathelier@ncmm.uio.no>

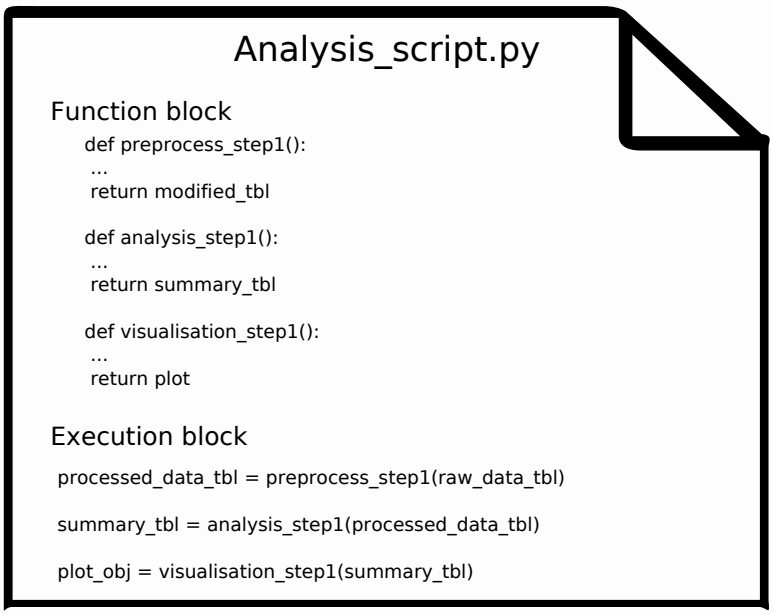
SUPPLEMENTARY FILE

SUPPLEMENTARY METHODS

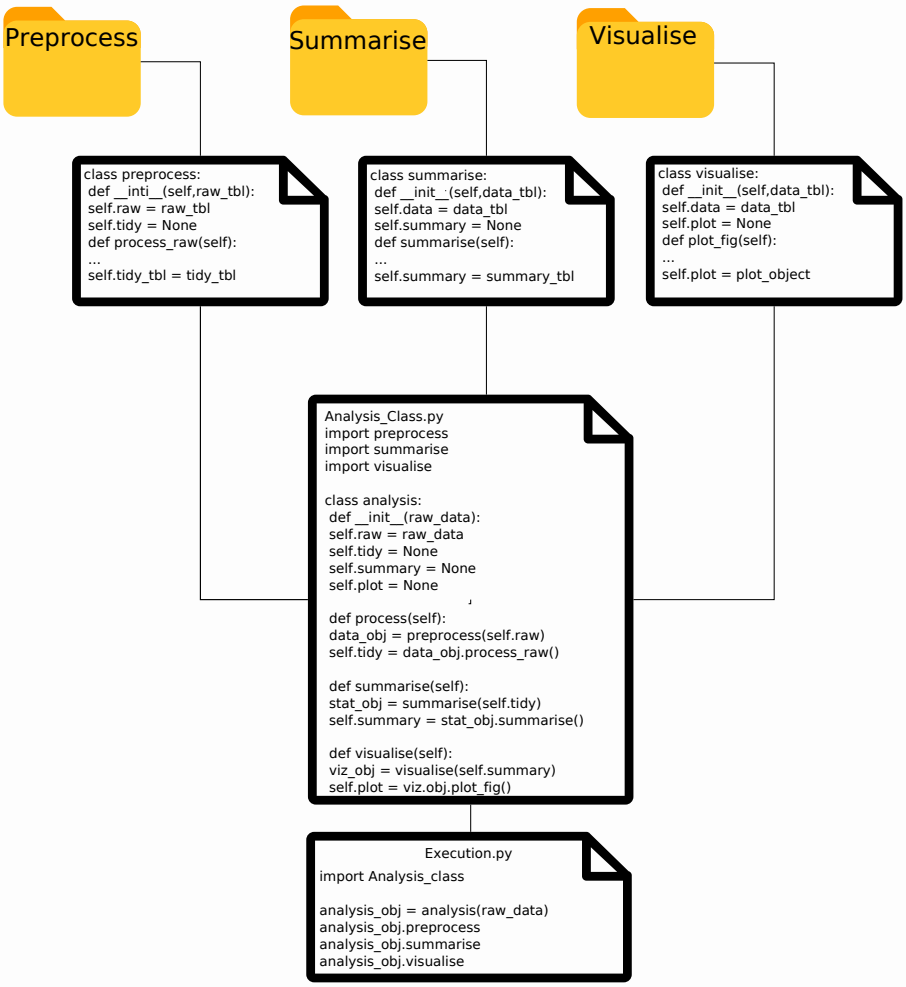
Following the standard methods of literature review, here we list the phrases and platforms of search. The literature search was performed in multiple iterations using Google (to include grey literature), PubMed and Google Scholar based on phrases “guidelines for bioinformatics software”, “rules for biologists learning bioinformatics”, “scientific software development”, “software engineering bioinformatics” and “bioinformatics software recommendations” throughout 2023. Additionally, relevant articles were selected based on the snowball effect from the references of the initial publications.

SUPPLEMENTARY FIGURES

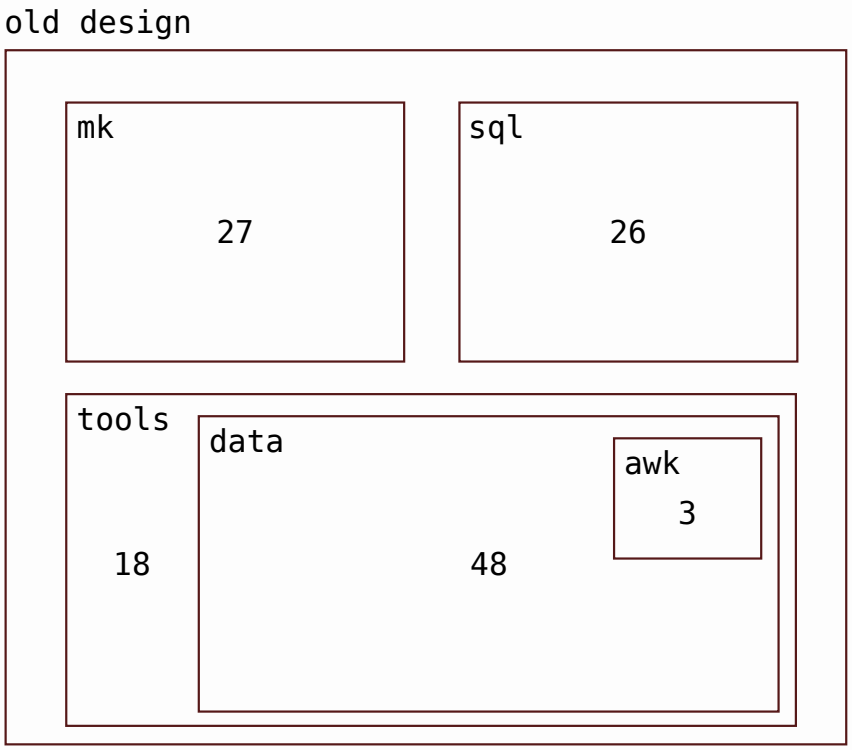
Modularization



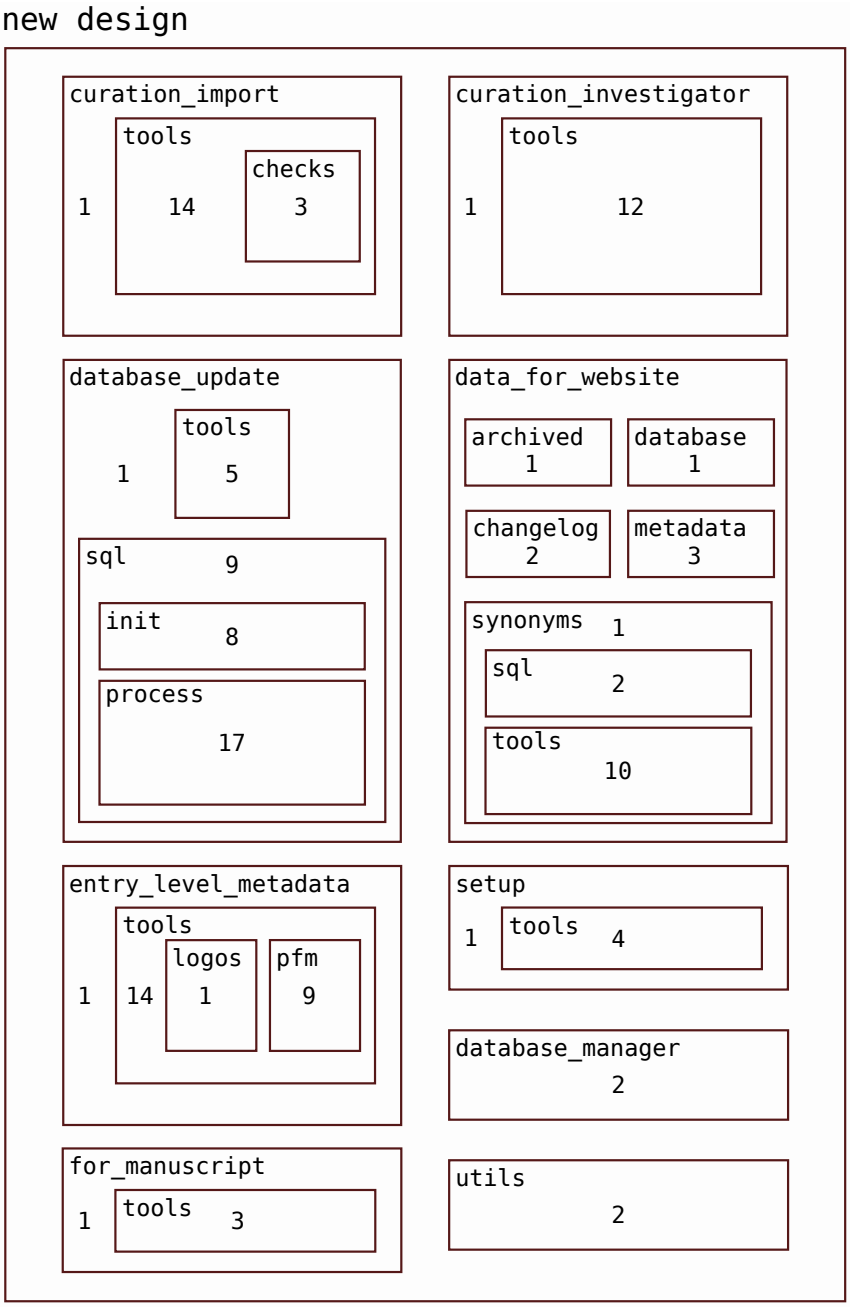
Supplementary Figure 1: **Improving the modularization of a small codebase: previous.** In the previous design a single script was used that processed data in one-off manner without consideration for extendability.



Supplementary Figure 2: **Improving the modularization of a small codebase: current.** In the current design we separate different aspects of the analysis into dedicated modules, which can be more robustly be extended.



Supplementary Figure 3: **Improving the modularization of a large codebase: previous.** In the previous design the files were arranged by on their type. The numbers denote the number of files in each directory represented by the rectangle. mk: makefile



Supplementary Figure 4: **Improving the modularization of a large codebase: current.** In the current design the files are arranged by their function. The numbers denote the number of files in each directory represented by the rectangle. The number of files are different due to added features and changes beyond the organization. pfm: position frequency matrix

Testing

This is an example of testing, represented by a subset of test used by the SPONGE package. The unit tests check the correctness of individual functions. Some of the tests shown test the plogp function, which calculates the value of p * log2(p) while treating the zero case correctly. Selected content of `tests/test_helper_functions.py` is shown below.

```
import pytest

# Helper functions
import sponge.helper_functions

# parametrize allows testing multiple inputs without code
duplication
@pytest.mark.parametrize("input, expected_output", [
    (0, 0),
    (0.5, -0.5),
    (1, 0),
])

def test_plogp(input, expected_output):
    assert helper_functions.plogp(input) == expected_output
```

Also tested is the calculation of the information content for individual motifs in the calculate_ic function. The motifs used by the tests are defined in a separate file `test_motifs` and accessible as pytest fixtures.

```
import pytest
import pandas as pd
from Bio.motifs.jaspar import Motif
from pyjaspar import jaspardb

# A motif without any information
@pytest.fixture
def no_info_motif():
    no_info_row = [0.25] * 4
    no_info_counts = [no_info_row] * 6
    no_info_pwm = pd.DataFrame(no_info_counts, columns=['A', 'C',
```

```
'G', 'T'])
    no_info_motif = Motif(matrix_id='XXX', name='XXX',
counts=no_info_pwm)

    yield no_info_motif


# A motif with perfect information
@pytest.fixture
def all_A_motif():
    all_A_row = [1] + [0] * 3
    all_A_counts = [all_A_row] * 6
    all_A_pwm = pd.DataFrame(all_A_counts, columns=['A', 'C',
'G', 'T'])
    all_A_motif = Motif(matrix_id='XXX', name='XXX',
counts=all_A_pwm)

    yield all_A_motif


# A real motif for SOX2
@pytest.fixture
def SOX2_motif():
    jdb_obj = jaspardb(release='JASPAR2024')
    SOX2_motif = jdb_obj.fetch_motif_by_id('MA0143.1')

    yield SOX2_motif
```

Selected content of `tests/test_helper_functions.py` is shown below.

```
import pytest
from test_motifs import *
from sponge.helper_functions import calculate_ic

def test_calculate_ic_no_info(no_info_motif):
    assert calculate_ic(no_info_motif) == 0


def test_calculate_ic_all_the_same(all_A_motif):
    # Length of the test motif is 6, so expected value is 2 * 6 =
12
```

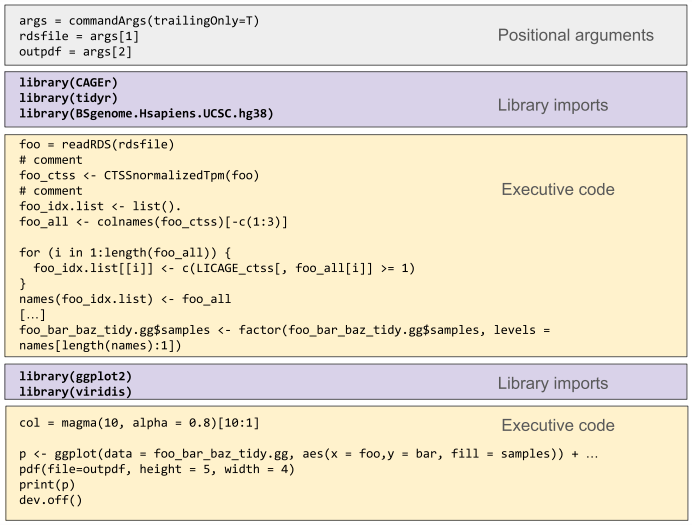
```
    assert calculate_ic(all_A_motif) == 12


def test_calculate_ic_SOX2(SOX2_motif):
    assert calculate_ic(SOX2_motif) == pytest.approx(12.95,
abs=0.01)
```

The integration tests check that the entire workflow produces the expected output, effectively checking that the components work well together. In this case, the full functionality of SPONGE with the default parameters is checked. Selected content of `tests/test_sponge.py` is shown below.

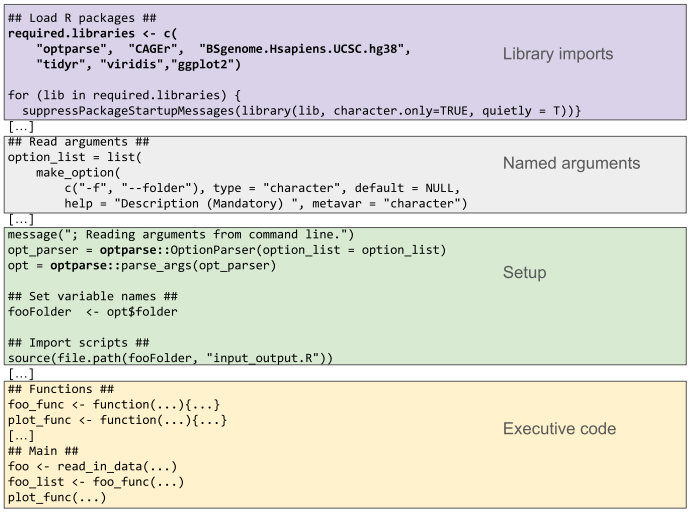
```
### Integration tests ###
import os
import pytest

from sponge.sponge import Sponge

# The test is marked as slow because the download of the bigbed
file takes
# a lot of time and the filtering is also time consuming unless
parallelised
@pytest.mark.slow
def test_full_default_workflow(tmp_path):
    # Tests the full SPONGE workflow with default values

    ppi_output = os.path.join(tmp_path, 'ppi_prior.tsv')
    motif_output = os.path.join(tmp_path, 'motif_prior.tsv')

    sponge_obj = Sponge(
        run_default=True,
        temp_folder=tmp_path,
        ppi_outfile=ppi_output,
        motif_outfile=motif_output,
    )

    assert os.path.exists(ppi_output)
    assert os.path.exists(motif_output)
```

Dependency management

There are two angles of dependency management we give example to here. First, we share a previous and current version of a code where the placing of the package imports are improved. This code also can be seen as an example for modularization with the rearrangement of the linear script to setup and functions. Furthermore, we improved the documentation and usability with using named arguments instead of positional ones.



Supplementary Figure 5: **An example for dependency management within the code: previous**



Supplementary Figure 6: **An example for dependency management within the code: current**

Second, we share an example of documenting the requirements where the responsibility of installing the software is moved from the user to the developer. README-based solution: the user is required to install the dependencies, version and source might be given but compatibility following updates is not ensured.

```
## Installation


- R (version >= 3.6.1)
- CAGEr (version >= 2.6.1) (for installation follow the
instructions here [https://bioconductor.org/packages/release/bioc/
vignettes/CAGEr/inst/doc/CAGEexp.html#normalization])
- BSgenome.Hsapiens.UCSC.hg38
- tidyr
- viridis
- ggplot2
```

Container-based solution: the user can either use the publicly available container that includes a snapshot of all necessary requirements, or build their own environment.

```
## Installation


Container available at https://hub.docker.com/r/cbgr/cager261
For details, refer to requirements.R
```

The content of `requirements.R` is shown below.

```
## Container folder structure
.libPaths( c( "/opt/software" , .libPaths() ) )
## CRAN packages:
packages_cran = c(
  "optparse",         ## Read in data
  "tidyr",                ## Data formatting and manipulation
  "ggplot2",           ## Plotting
  [...])
message(
"; Installing these R packages from CRAN repository:",
packages_cran)
install.packages(
packages_cran, repos="https://cran.uib.no/", lib="/opt/software")
```

```
## Bioconductor packages:
packages_bioconductor <- c(
"BSgenome.Hsapiens.UCSC.hg38")

message(
"; Installing these R Bioconductor packages: ",
packages_bioconductor)
BiocManager::install(packages_bioconductor, lib="/opt/software")
```

The content of the `Dockerfile` is shown below.

```
# Docker install R 4.3, Bioconductor 3.17
FROM bioconductor/bioconductor_docker:3.17

# Set up folder structure
WORKDIR /opt/software

# Install CAGEr 2.6.1
RUN R -e 'BiocManager::install("CAGEr")'

# Install other R dependencies
COPY requirements.R /opt/software/requirements.R
RUN Rscript requirements.R
ENV R_LIBS=${R_LIBS}:/opt/software
```

SUPPLEMENTARY TABLES

Supplementary Table 1: Software quality attributes and their description

Below is a high level overview of the software quality attributes of the ISO 25010 standard.

Attribute	Description
Functional Suitability	Whether the software functions as expected.
Performance Efficiency	Characterizes how well does the software product utilizes computational resources.

Attribute	Description
Compatibility	Describes how well does the software can work together with other components, e.g. other tools.
Usability	The quality of the software from the perspective of the experience of the user.
Reliability	A software product to behave as expected under pressure, tolerate failures, and recover quickly.
Security	A software product to protect information and ensure authorized access only.
Maintainability	The ease at which new features can be added and bugs can be fixed.
Portability	The ease of moving the system onto a different environment, such as installing to a new device.

Supplementary Table 2: **Examples of software quality meeting topics** This table contains examples of the topics of past software quality meetings. It has been organise to follow the same categories as **Table 1**.

Category	Title	Description
Software development 101	Big O notation, Stack and Queue	Review of Data Structures and Algorithms
	Union Find and Hash Table	Review of Data Structures and Algorithms
	Heaps and Binary Search Trees	Review of Data Structures and Algorithms

Category	Title	Description
	Graphs, graph algorithms	Review of Data Structures and Algorithms
	Dynamic programming and Sorting	Review of Data Structures and Algorithms
	Databases	Review of Database solutions
	Introduction to sed	Understanding command line tricks used in bioinformatics
	Introduction to awk	Understanding command line tricks used in bioinformatics
	S4 R objects	Understand class objects in R
	Snakemake	How to build pipelines using Snakemake workflow manager
	Plotting	How to improve figures
Advanced software development	Design patterns	Review the concept of design patterns through a set of common examples
	Class diagrams	Unified Modeling Language and its usage for software analysis and design
	The Pragmatic Programmer	Book review
	Inheritance and decorators	Review of miscellaneous concepts in object oriented programming in Python

Category	Title	Description
Software development process	Introduction to software architecture	How to handle medium to large sized code bases
	Parallelization in R	Review of methods to parallel processing in R
	Parallelization in Python	Review of methods to parallel processing in Python
	Error handling and defensive programming	Introduction to error handling and expecting the unexpected
	Code review	Reviewing a github repository and discussing how to make your code understandable for others
	Git and GitFlow	Understand the version controlling with Git. Branching and merging
	R package development	How to create an R package to share with the community
	Technical debt	Understand the concept of technical debt and discuss the impact on our work
	A project management example: JASPAR database update codebase	Software development practices in a team settings using requirement elicitation, JIRA, docstring, code reviews, testing and more
Testing and validation	Debugging and testing	Basic introduction about tools and methods for catching bugs

Category	Title	Description
Reproducibility	Testing workshop	Try out refactoring and test writing of our tools
	Docker & conda	Discuss platforms and tools for dependency management
	Docker & continuous integration	Workshop on how to use docker and GitHub Actions
Documentation	RMarkdown	Basic aspects of writing a report in markdown with text, tables and plots.
	GitHub and Bitbucket pages	Extended online documentation of tools and research results
Community effort	Sustainable computation in bioinformatics	Get to know our computational footprint. How to choose “green” software.