A No-Nonsense Guide to Higher Code Quality for Researchers

Michaela Leštáková, Kevin T. Logan, Daniele Inturri

NFDI4ing Conference

TECHNISCHE UNIVERSITÄT DARMSTADT

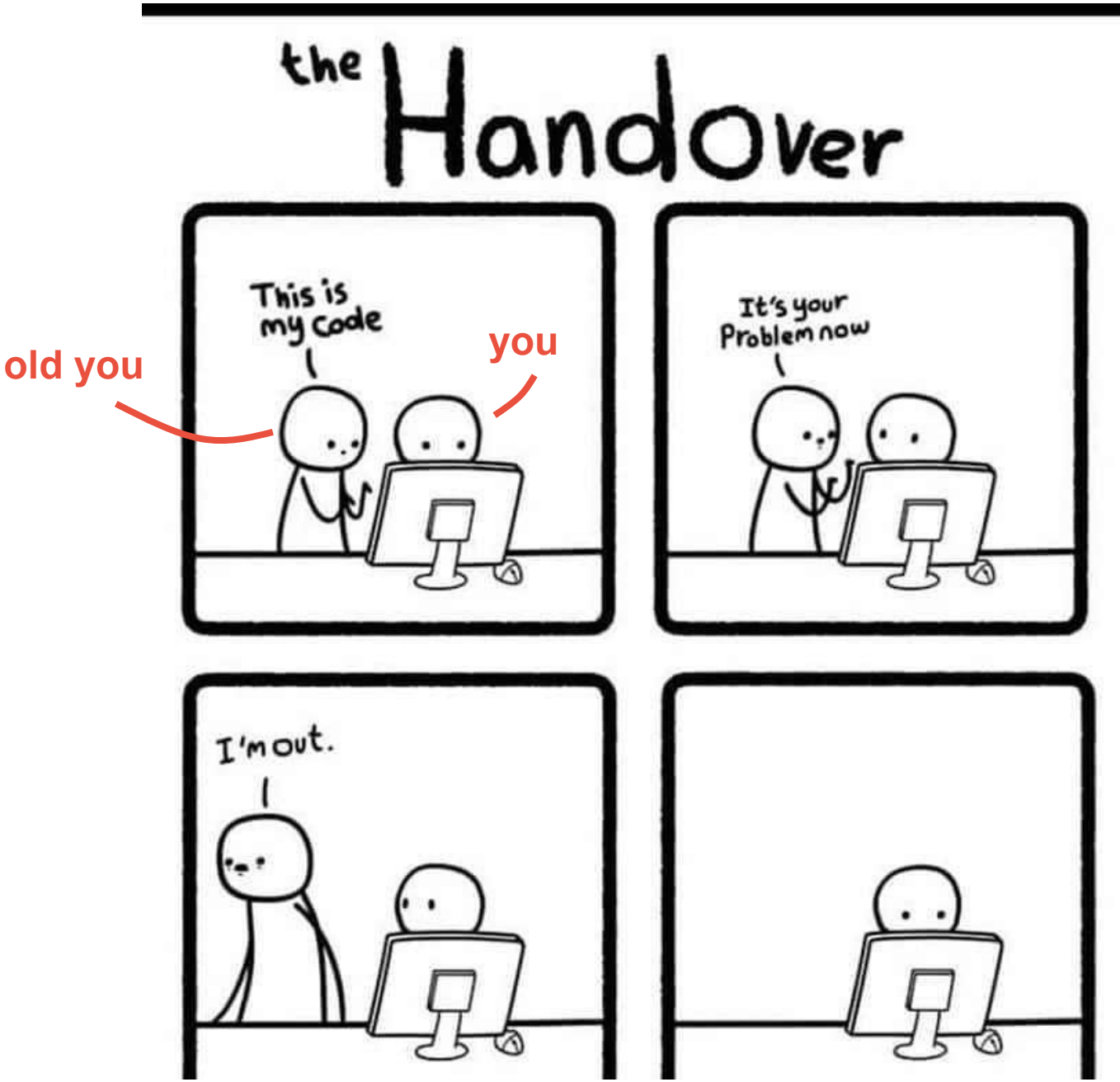
MASCHINENBAU
We engineer future

FLUIDSYSTEMTECHNIK
Prof. Dr.-Ing. Peter F. Pelz

Picture Source: https://devrant.com/rants/2948651/the-handover

Picture Source: https://devrant.com/rants/2948651/the-handover

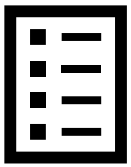
Picture Source: https://devrant.com/rants/2948651/the-handover



Picture Source: https://devrant.com/rants/2948651/the-handover

Today‘s Goal

TECHNISCHE UNIVERSITÄT DARMSTADT



determine an **easy** and **quick-to-implement** set of guidelines for higher code quality

zoom in onto unit testing

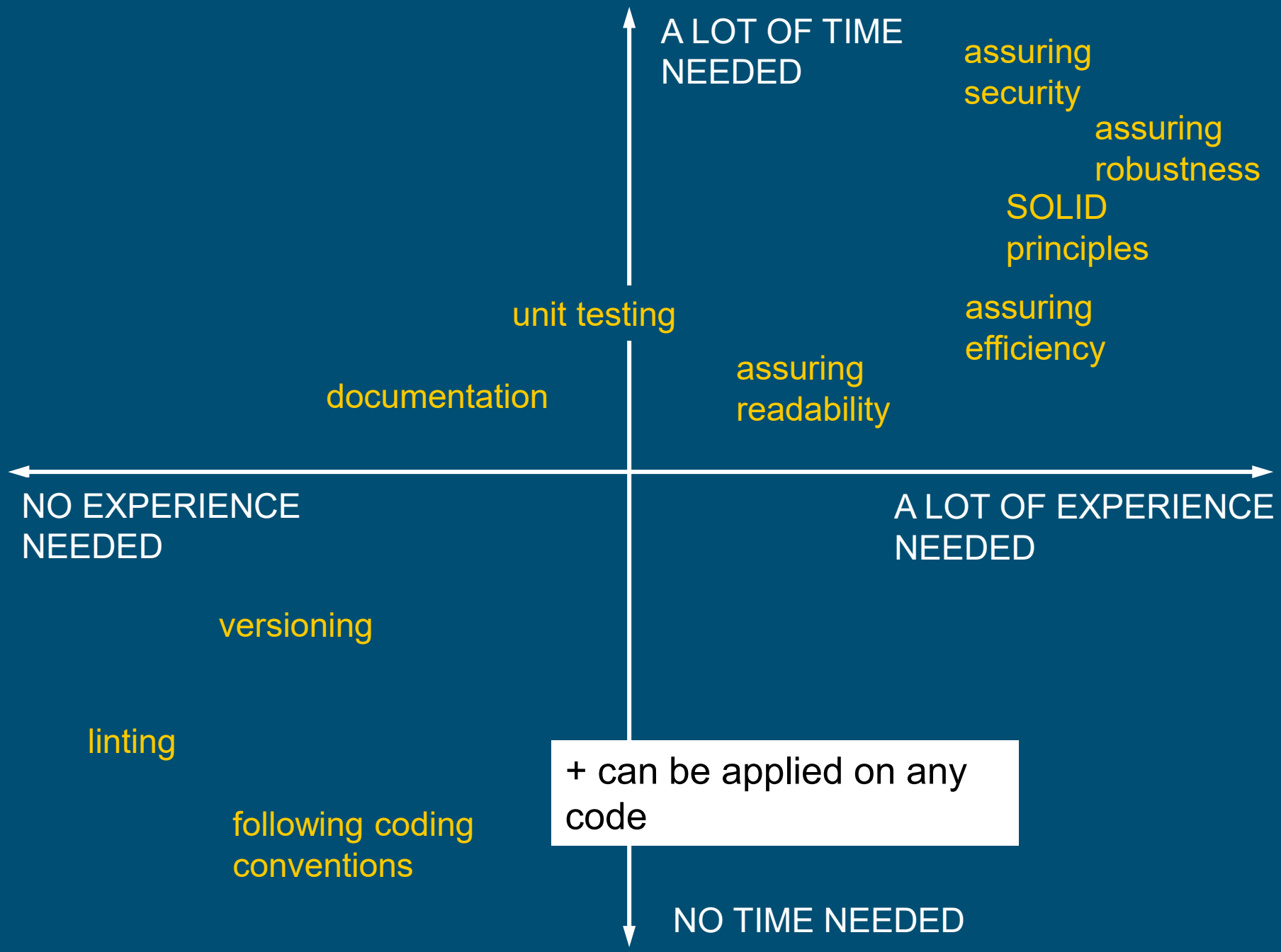
What we will not look into

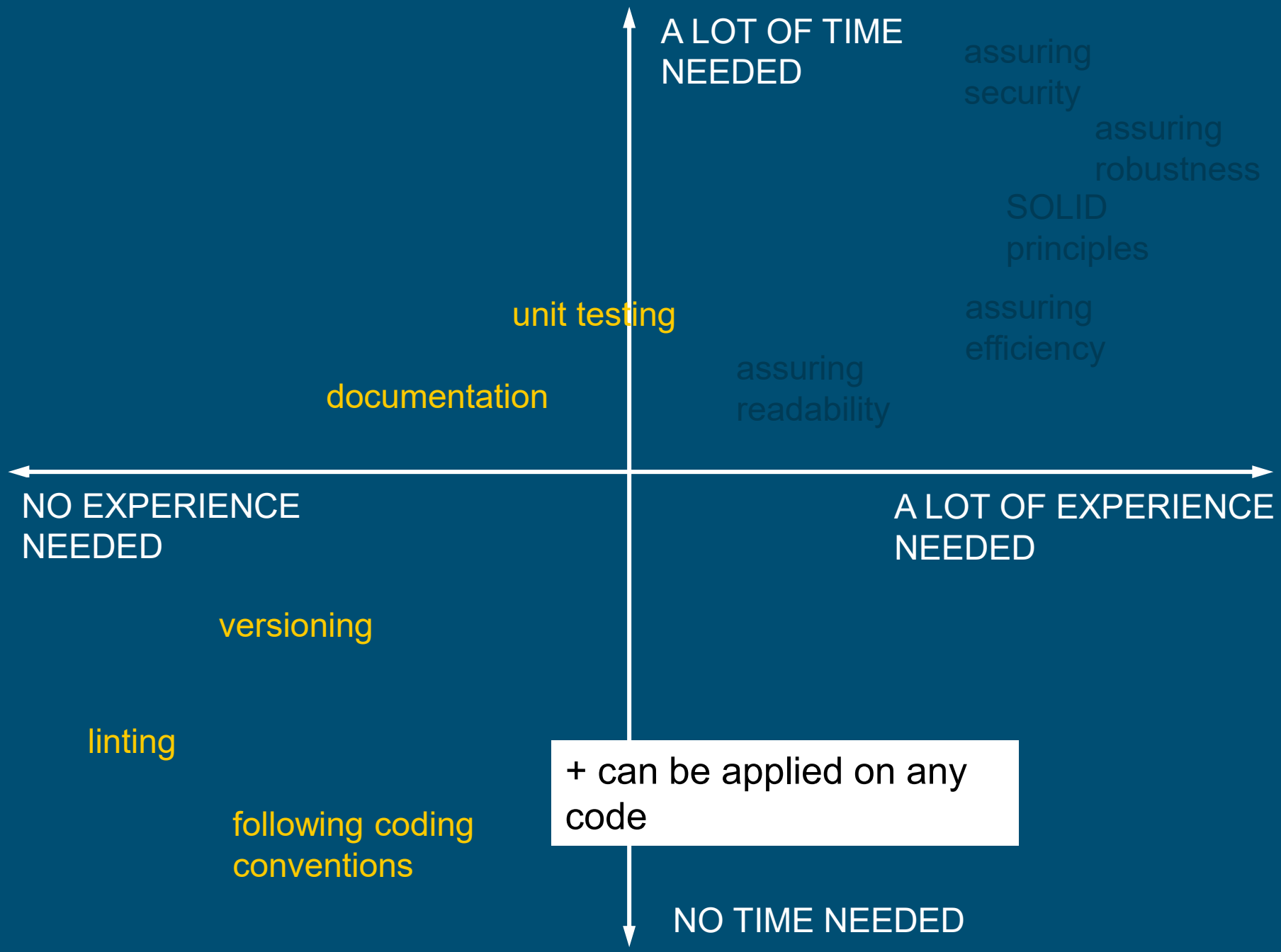
TECHNISCHE UNIVERSITÄT DARMSTADT

- quality of the content of the code
- how to write good object oriented code
- problem-specific questions

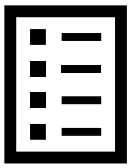
Code Quality

https://miro.com/app/board/uXjVPK1Bq20=/


A LOT OF TIME NEEDED
assuring security
assuring robustness
SOLID principles
assuring efficiency
unit testing
documentation
assuring readability
NO EXPERIENCE NEEDED
A LOT OF EXPERIENCE NEEDED
versioning
linting
following coding conventions
+ can be applied on any code
NO TIME NEEDED


A LOT OF TIME NEEDED
assuring security
assuring robustness
SOLID principles
assuring efficiency
assuring readability
unit testing
documentation
NO EXPERIENCE NEEDED
A LOT OF EXPERIENCE NEEDED
versioning
linting
following coding conventions
+ can be applied on any code
NO TIME NEEDED

Today‘s Goal



determine an easy and quick-to-implement set of guidelines for higher code quality

Styleguide

TECHNISCHE UNIVERSITÄT DARMSTADT

before coding

Common Infrastructure and Tools

during coding

Language Rules

- linting
- autoformatting
- type checking

Style Rules

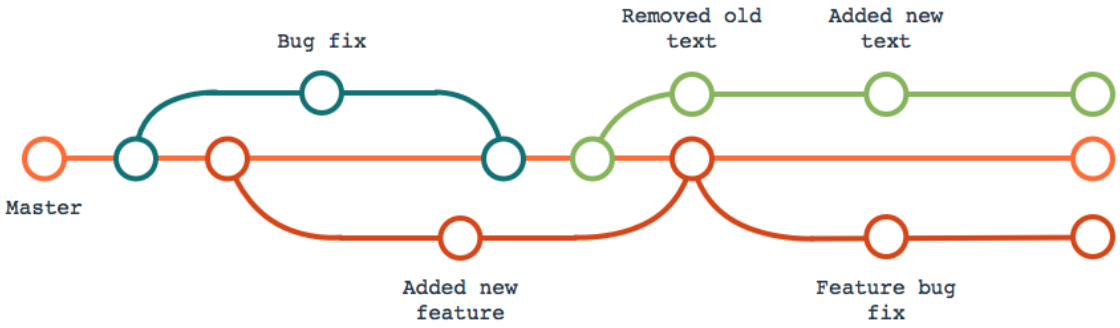
- documentation
- naming conventions
- code structure

Unit Testing

Common Infrastructure and Tools

Version Control

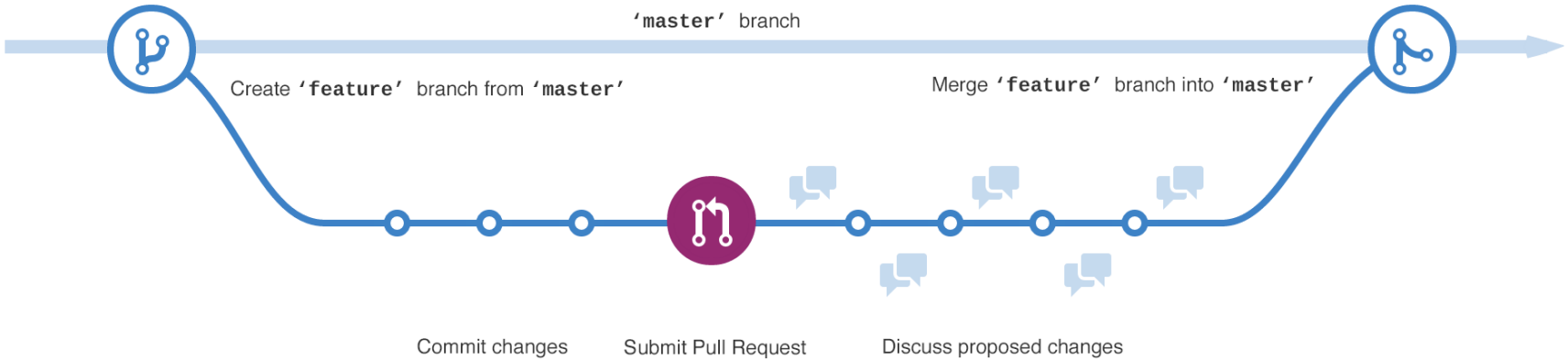
Common Infrastructure and Tools
Version Control



Picture Source: https://blog.cpanel.com/git-version-control-series-what-is-git/

- ✓ collaborative work becomes easier (researcher – student, researcher – researcher)
- ✓ possibility to reference different versions of the code

Common Infrastructure and Tools
Version Control



Picture Source: https://blog.codeinsider.fr/gerer-son-worklfow-avec-git/

Defining common workflows helps **prevent chaos**
and makes it **easier for new team members** to get on track.

Github Workflow
Gitlab Workflow (a little less straightforward)

Common Infrastructure and Tools
Version Control

TECHNISCHE UNIVERSITÄT DARMSTADT

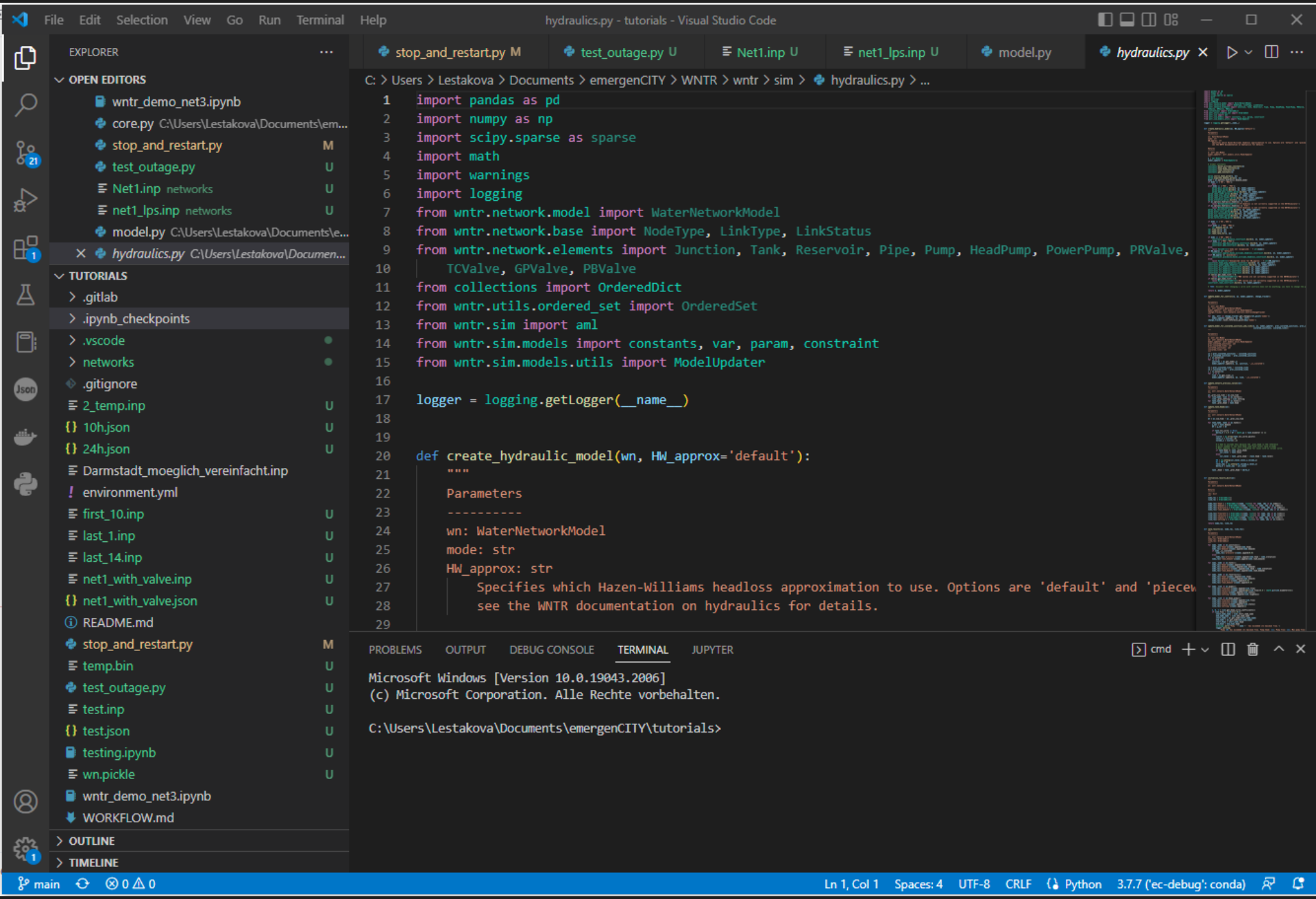
Agree on and use the a version control tool (git) and a platform for collaborative work, like GitLab.

Clearly describe the common workflow.

(+) Create common templates for projects, merge requests, issues etc.

Common Infrastructure and Tools

Code Editor


File Edit Selection View Go Run Terminal Help
hydraulics.py - tutorials - Visual Studio Code
EXPLORER
OPEN EDITORS
wntr_demo_net3.ipynb
core.py C:\Users\Lestakova\Documents\em...
stop_and_restart.py M
test_outage.py U
Net1.inp networks U
net1_lps.inp networks U
model.py C:\Users\Lestakova\Documents\e...
hydraulics.py C:\Users\Lestakova\Documen...
TUTORIALS
.gitlab
.ipynb_checkpoints
.vscode
networks
.gitignore
2_temp.inp U
10h.json U
24h.json U
Darmstadt_moeglich_vereinfacht.inp
environment.yml
first_10.inp U
last_1.inp U
last_14.inp U
net1_with_valve.inp U
net1_with_valve.json U
README.md
stop_and_restart.py M
temp.bin U
test_outage.py U
test.inp U
test.json U
testing.ipynb U
wn.pickle U
wntr_demo_net3.ipynb
WORKFLOW.md
OUTLINE
TIMELINE
stop_and_restart.py M
test_outage.py U
Net1.inp U
net1_lps.inp U
model.py
hydraulics.py
C: > Users > Lestakova > Documents > emergenCITY > WNTR > wntr > sim > hydraulics.py > ...
import pandas as pd
import numpy as np
import scipy.sparse as sparse
import math
import warnings
import logging
from wntr.network.model import WaterNetworkModel
from wntr.network.base import NodeType, LinkType, LinkStatus
from wntr.network.elements import Junction, Tank, Reservoir, Pipe, Pump, HeadPump, PowerPump, PRValve,
TCValve, GPValve, PBValve
from collections import OrderedDict
from wntr.utils.ordered_set import OrderedSet
from wntr.sim import aml
from wntr.sim.models import constants, var, param, constraint
from wntr.sim.models.utils import ModelUpdater
logger = logging.getLogger(__name__)
def create_hydraulic_model(wn, HW_approx='default'):
"""
Parameters
----
wn: WaterNetworkModel
mode: str
HW_approx: str
Specifies which Hazen-Williams headloss approximation to use. Options are 'default' and 'piecew
see the WNTR documentation on hydraulics for details.
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER
cmd
Microsoft Windows [Version 10.0.19043.2006]
(c) Microsoft Corporation. Alle Rechte vorbehalten.
C:\Users\Lestakova\Documents\emergenCITY\tutorials>
main
0 0
Ln 1, Col 1 Spaces: 4 UTF-8 CRLF Python 3.7.7 ('ec-debug': conda)

Common Infrastructure and Tools
Code Editor

TECHNISCHE UNIVERSITÄT DARMSTADT

- using the same editor will make it easier for less experienced colleagues to focus on the code
- some editors allow you to save the editor setup along with the code:

```
{
    "python.formatting.provider": "black",
    "python.sortImports.args": [
        "--profile=black",
    ],
    "[python]": {
        "editor.rulers": [80],
        "editor.formatOnSave": true,
        "editor.codeActionsOnSave": {
            "source.organizeImports": true
        }
    },
    "files.exclude": {
        "**/__pycache__": true,
        "**/*.egg-info": true,
    }
}
```

Common Infrastructure and Tools
Code Editor

TECHNISCHE UNIVERSITÄT DARMSTADT

Agree on and use the same editor.

(+) Save/track editor settings in your projects.

Language Rules
Linting

Language Rules
Linting

TECHNISCHE UNIVERSITÄT DARMSTADT

```
48
49  #Visualisieren des Netwerks
50  pos = nx.spring_layout(G, pos = node_coordinates, fixed = fixed_positions)
51  plt.figure()
52  nx.draw(
53      G, pos, edge_color='black', width=1, linewidths=1,
54      node_size=15) #,labels={node: node for node in G.nodes()}) #Knotenbeschriftung
55
56  #nx.draw_networkx_edge_labels(G, pos, edge_labels=farness_centrality) #Kantenbeschriftung
57  plt.axis('off')
58  plt.title('Farness Centrality')
59
60
61  #M
62  so
63  #Anzahl der zu entfernenden Kanten
64  for i in range(len(farness_centrality)-100):
65      j = 100
66      del sorted_betw_cent[next(islice(sorted_betw_cent, j, None))]
67
68  marked_nodes = sorted_betw_cent
69  nx.draw_networkx_nodes(G, pos = node_coordinates, nodelist= marked_nodes, node_color= 'red', node_size=15)
70
71
72  ''' #Ausgabe in der Form: [[Kantenname, Wert][..., ...]]
73  pipe_name_farness_centrality = []
74  for i in farness_centrality:
75      for junction_ID, junction in wn.junctions():
```

→ code not consistent, chaotic, hard to read, buggy

Language Rules
Linting

TECHNISCHE UNIVERSITÄT DARMSTADT

- is a static code analysis for finding programming errors, bugs, stylistic errors and suspicious constructs that do not conform with a standard, such as pep8
- good news: there are tools for that!

Language Rules
Linting

TECHNISCHE UNIVERSITÄT DARMSTADT

```
48
49  #Visualisieren des Netwerk
50  pos = nx.spring_layout(G, pos = node_coordinates, fixed = fixed_positions)
51  plt.figure()
52  nx.draw(
53      G, pos, edge_color='black', width=1, linewidths=1,
54      node_size=15) #, labels={node: node for node in G.nodes()}) #Knotenbeschriftung
55
56  #nx.draw_networkx_edge_labels(G, pos, edge_labels=farness_centrality) #Kantenbeschriftung
57  plt.axis('off')
58  plt.title('Farness Centrality')
59
60
61  #Markieren und visualisieren der Kanten mit dem höchsten Farness Centrality Wert
62  sorted_betw_cent = {k: v for k, v in sorted(farness_centrality.items(), key=lambda item: item[1], reverse=False)}
63  #Anzahl der zu entfernenden Kanten
64  for i in range(len(farness_centrality)-100):
65      j = 100
66      del sorted_betw_cent[next(islice(sorted_betw_cent, j, None))]
67
68  marked_nodes = sorted_betw_cent
69  nx.draw_networkx_nodes(G, pos = node_coordinates, nodelist= marked_nodes, node_color= 'red', node_size=15)
70
71
72  ''' #Ausgabe in der Form: [[Kantenname, Wert][..., ...]]
73  pipe_name_farness_centrality = []
74  for i in farness_centrality:
75      for junction_ID, junction in wn.junctions():
```

Language Rules
Linting

TECHNISCHE UNIVERSITÄT DARMSTADT

Agree on a linter and lint on save.

(+) add linting into your CI/CD pipeline.

Language Rules
Autoformatting

Language Rules
Autoformatting

TECHNISCHE UNIVERSITÄT DARMSTADT

- autoformatted code will look the same
  - regardless of who wrote it
  - regardless of the project
- you can focus more on the content!
- Python autoformatters: black, autopep8, yapf

Language Rules
Autoformatting

TECHNISCHE UNIVERSITÄT DARMSTADT

```
for zwitch in switches:
        model = {"id": "101", "type": "Switch", "bus": "Bus_123"}

for bus, location in zip(buses, bus_locations):
        model = {"id": "Bus_123", "type": "Bus",
            "location": { "type": "Point", "coordinates": [location["x"], location["y"]]}}
```

```
for zwitch in switches:
        model = {
            "id": "101",
            "type": "Switch",
            "bus": "Bus_123",
        }

for bus, location in zip(buses, bus_locations):
        model = {
            "id": "Bus_123",
            "type": "Bus",
            "location": {
                "type": "Point",
                "coordinates": [location["x"], location["y"]],
            },
        }
```

Language Rules
Autoformatting

TECHNISCHE UNIVERSITÄT DARMSTADT

Agree on an autoformatter and autoformat on save/on type.

(+) add autoformatting into your CI/CD pipeline.

Language Rules
Type Checking

Language Rules
Type Checking

TECHNISCHE UNIVERSITÄT DARMSTADT

- **static type checking**: the type of variable must be known at compile time (C++, Java, …)
- **dynamic type checking**: type gets verified at runtime (Python, Ruby, …)
- Python allows to use an optional static type checker, e.g. mypy

```python
def calculate_distance_geometric(point_1, point_2):
    d_x = point_2[0] - point_1[0]
    d_y = point_2[1] - point_1[1]
    d = sqrt(d_x ** 2 + d_y ** 2)
    return d
```

```python
def calculate_distance_geometric(
    point_1: tuple[float, float], point_2: tuple[float, float]
) -> float:
    d_x = point_2[0] - point_1[0]
    d_y = point_2[1] - point_1[1]
    d = sqrt(d_x ** 2 + d_y ** 2)
    return d
```

Language Rules
Type Checking

TECHNISCHE UNIVERSITÄT DARMSTADT

Agree on and use a type checker.

(+) add type checking into your CI/CD pipeline.

Style Rules
Documentation

Style Rules
Documentation

TECHNISCHE UNIVERSITÄT DARMSTADT

docstring

comment

```python
def create_hydraulic_model(wn, HW_approx='default'):
    """
    Parameters
    ----------
    wn: WaterNetworkModel
    mode: str
    HW_approx: str
        Specifies which Hazen-Williams headloss approximation to use. Options are 'default' and 'piecewise'. Please
        see the WNTR documentation on hydraulics for details.

    Returns
    -------
    m: wntr.aml.Model
    model_updater: wntr.models.utils.ModelUpdater
    """
    m = aml.Model()
    model_updater = ModelUpdater()

    # Global constants
    constants.hazen_williams_constants(m)
    constants.head_pump_constants(m)
    constants.leak_constants(m)
    constants.pdd_constants(m)

    param.source_head_param(m, wn)
    param.expected_demand_param(m, wn)
    mode = wn.options.hydraulic.demand_model
    if mode in ['DD', 'DDA']:
        pass
```

Style Rules
Documentation

TECHNISCHE UNIVERSITÄT DARMSTADT

- docstrings use different styles (google/sphinx/numpy)
- docstrings can be automatically generated (e.g. VS code extension „Docstring Generator“)

```python
def calculate_distance_geometric(point_1, point_2):
    """_summary_

    Args:
        point_1 (_type_): _description_
        point_2 (_type_): _description_

    Returns:
        _type_: _description_
    """
    d_x = point_2[0] - point_1[0]
    d_y = point_2[1] - point_1[1]
    d = sqrt(d_x ** 2 + d_y ** 2)
    return d
```

```python
def calculate_distance_geometric(point_1, point_2):
    """Calculates geometric distance between two points.

    Args:
        point_1 (tuple): coordinates of the first point
        point_2 (tuple): coordinates of the second point

    Returns:
        float: geometric distance between the two points
    """
    d_x = point_2[0] - point_1[0]
    d_y = point_2[1] - point_1[1]
    d = sqrt(d_x ** 2 + d_y ** 2)
    return d
```

Style Rules
Documentation

TECHNISCHE UNIVERSITÄT DARMSTADT

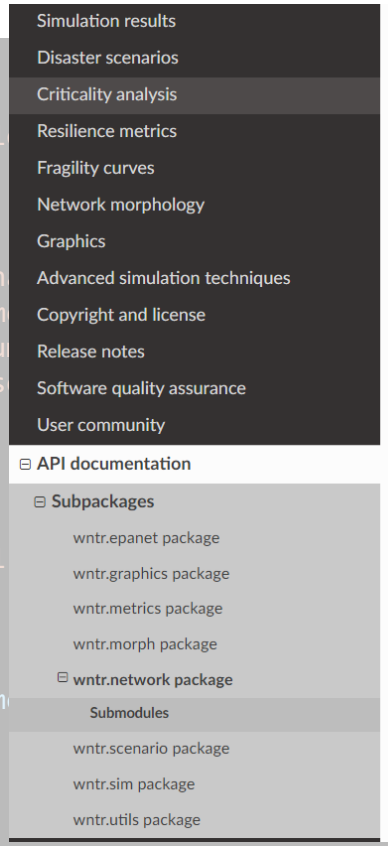
Interactive documentation using Sphinx

```
def get_volume(self, level=None):
    """
    Returns tank volume at a given l

    Parameters
    ----------
    level: float or NoneType (option
        The level at which the volum
        If level=None, then the volu
        at the current tank level (s

    Returns
    -------
    vol: float
        Tank volume at a given level
    """

    if self.vol_curve is None:
        A = (np.pi / 4.0 * self.diam
        if level is None:
            level = self.level
        vol = A * level
    . . .
```



get_volume (*level=None*) [source]

Returns tank volume at a given level

Parameters

level (*float or NoneType (optional)*) – The level at which the volume is to be calculated. If level=None, then the volume is calculated at the current tank level (self.level)

Returns

vol (*float*) – Tank volume at a given level

add_leak(*wn*, *area*, *discharge_coeff=0.75*, *start_time=None*, *end_time=None*) [source]

Add a leak to a tank.

Leaks are modeled by:

Q = discharge_coeff*area*sqrt(2*g*h)

where:

Q is the volumetric flow rate of water out of the leak g is the acceleration due to gravity h is the gauge head at the bottom of the tank, P_g/(rho*g); Note that this is not the hydraulic head (P_g + elevation)

Note that WNTR assumes the leak is at the bottom of the tank.

Parameters

- **wn** (`WaterNetworkModel`) – The WaterNetworkModel object containing the tank with the leak. This information is needed because the WaterNetworkModel object stores all controls, including when the leak starts and stops.

Citing WNTR — WNTR 0.5.0 documentation

Style Rules
Documentation

TECHNISCHE UNIVERSITÄT DARMSTADT

Agree on docstring style and use automatic docstring generator.

All functions, methods and classes must have docstrings.

Create a workflow for generating code documentation.

(+) Create code documentation in a CI/CD pipeline

Style Rules

Naming Conventions

Style Rules
Naming Conventions

TECHNISCHE UNIVERSITÄT DARMSTADT

- follow a language-specific naming convention
- for better readability, use meaningful names instead of hard-to-understand short names
- use a spell checker

Type	Public	Internal
Packages	`lower_with_under`	
Modules	`lower_with_under`	`_lower_with_under`
Classes	`CapWords`	`_CapWords`
Exceptions	`CapWords`	
Functions	`lower_with_under()`	`_lower_with_under()`
Global/Class Constants	`CAPS_WITH_UNDER`	`_CAPS_WITH_UNDER`
Global/Class Variables	`lower_with_under`	`_lower_with_under`
Instance Variables	`lower_with_under`	`_lower_with_under` (protected)
Method Names	`lower_with_under()`	`_lower_with_under()` (protected)
Function/Method Parameters	`lower_with_under`	
Local Variables	`lower_with_under`	

PEP8 summarized in
styleguide | Style guides for Google-originated open-source projects

Style Rules

Code Structure

Style Rules
Code Structure

TECHNISCHE UNIVERSITÄT DARMSTADT

Set a maximum line length (80 or 120 characters).

Automatically delete whitespaces.

Improve readability by limiting the number of nested statements.

Preferrably write short functions, and functions that can be tested.

Discussion

Do you use other tools related to Code Quality you can‘t do without anymore?

https://miro.com/app/board/uXjVPK1Bq20=/

Unit Testing in Python

Michaela Leštáková, Kevin T. Logan, Daniele Inturri

NFDI4ing Conference

TECHNISCHE UNIVERSITÄT DARMSTADT

MASCHINENBAU
We engineer future

FLUIDSYSTEMTECHNIK
Prof. Dr.-Ing. Peter F. Pelz

What to expect

TECHNISCHE UNIVERSITÄT DARMSTADT

- general introduction to automated testing
- introduction to various tools for testing in python
- metrics for completeness/quality of tests → code coverage
- advanced testing strategies → property-based-testing
- tips for writing testable code

Why write tests?

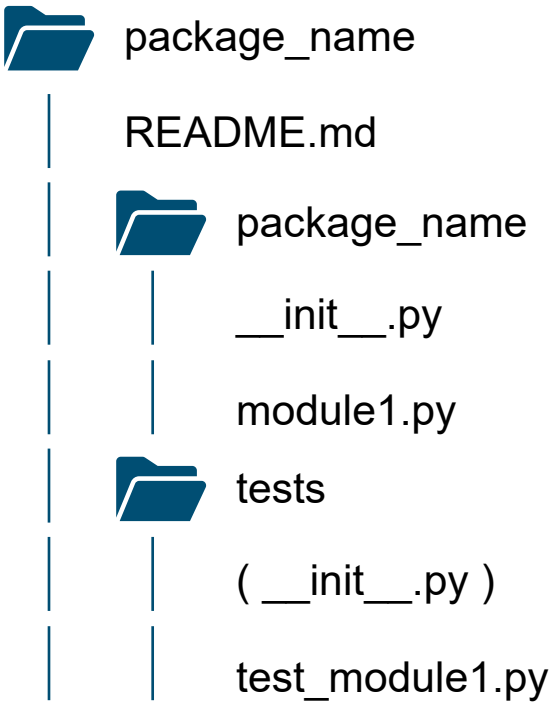
TECHNISCHE UNIVERSITÄT DARMSTADT

- higher confidence code works as expected
- safety net when refactoring code
- more time efficient than testing manually
- improves code quality → easy to test code = good code
- simpler debugging process
- enhances working in a team
- ...

Setting up a testing framework

TECHNISCHE UNIVERSITÄT DARMSTADT

project structure



tools

- unittest
  - part of the standard library
  - test functions in classes
- pytest
  - third party package
  - test functions can be defined outside classes

→ pytest simpler and requires less boilerplate code

Anatomy of a test - AAA

TECHNISCHE UNIVERSITÄT DARMSTADT

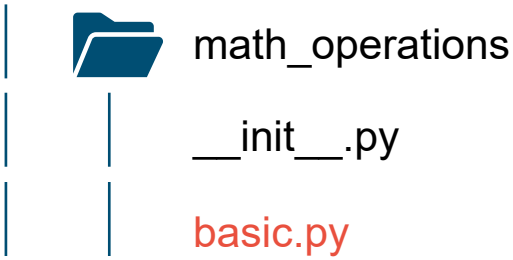
1. Arrange → prepare test
2. Act → call the function to be tested
3. Assert → compare output with expected output
4. (Cleanup)

https://docs.pytest.org/en/7.1.x/explanation/anatomy.html

Writing a module

Example function

TECHNISCHE UNIVERSITÄT DARMSTADT




```python
def add(num1: Union[int, float], num2: Union[int, float]) -> Union[int, float]:
    return num1 + num2
```

Writing tests

The first test

TECHNISCHE UNIVERSITÄT DARMSTADT


```python
from math_operations.basic import add
def test_add() -> None:
    assert add(2,2) == 4
```

naming convention: test_*

```
\math_operations$ pytest
```

Writing tests

TECHNISCHE UNIVERSITÄT DARMSTADT

Demo

Writing a module

Example function

TECHNISCHE UNIVERSITÄT DARMSTADT

math_operations

__init__.py

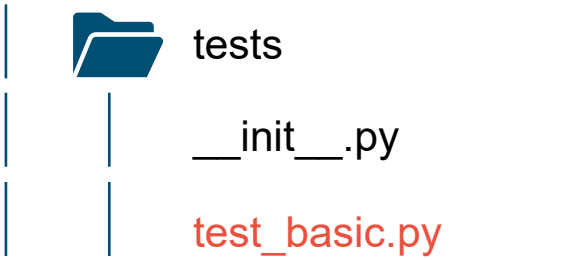
basic.py

```python
def add(num1: Union[int, float], num2: Union[int, float]) -> Union[int, float]:
    if not isinstance(num1, (int, float)):
        raise TypeError(f"'num1' is {type(num1)}; expected float, int")
    if not isinstance(num2, (int, float)):
        raise TypeError(f"'num2' is {type(num2)}; expected float, int")
    return num1 + num2
```

Writing tests

Adding a second test

TECHNISCHE UNIVERSITÄT DARMSTADT




```python
import pytest

def test_add_raises_type_error() -> None:
    with pytest.raises(TypeError):
        add("a", [1])
```

Writing tests

TECHNISCHE UNIVERSITÄT DARMSTADT

Demo

Code Coverage

TECHNISCHE UNIVERSITÄT DARMSTADT

- metric for the amount of code tested by the unit tests

Statement Coverage	$\frac{\text{number of executed statements}}{\text{total number of statements}}$
Branch Coverage	$\frac{\text{number of executed branches}}{\text{total number of branches}}$
Condition Coverage	percentage of conditions that affected independently the outcome of a conditional statement

Code Coverage

example

TECHNISCHE UNIVERSITÄT DARMSTADT

```python
def foo(x: int, y: int) -> int:

    if x > 0 and y > 0:
        z = x + y
    else:
        z = 10
    return z
```

- 4 statements
- 2 branches
- 3 independent conditions

  → (True, True), (True, False), (False, False/True)

Code Coverage

example

TECHNISCHE UNIVERSITÄT DARMSTADT

Test 1: (x = 1, y = 1)

- statement coverage = 3/4 = 75%
- branch coverage = 1/2 = 50%
- condition coverage = 1/3 = 33% (True, True)

Code Coverage

example

TECHNISCHE UNIVERSITÄT DARMSTADT

```
def foo(x: int, y: int) -> int:

    else:
```

Test 1: (x = 1, y = 1); (x = 0, y = 1)

- statement coverage = 4/4 = 100%
- branch coverage = 2/2 = 100%
- condition coverage = 2/3 = 67% (True, True), (False, True)

Code Coverage

example

TECHNISCHE UNIVERSITÄT DARMSTADT

```python
def foo(x: int, y: int) -> int:

    else:
```

Test 1: (x = 1, y = 1); (x = 0, y = 1); (x = 1, y = 0)

- statement coverage = 4/4 = 100%
- branch coverage = 2/2 = 100%
- condition coverage = 3/3 = 100% (True, True), (False, True), (True, False)

Code Coverage

pytest-cov

TECHNISCHE UNIVERSITÄT DARMSTADT

- pytest-cov: package for generating coverage reports
- reports for statement & branch coverage can be generated but not for condition coverage

```
pytest --cov=math_operations tests/ --cov-report=html --cov-branch
```

Code Coverage

TECHNISCHE UNIVERSITÄT DARMSTADT

Demo

Code Coverage

pytest-cov

TECHNISCHE UNIVERSITÄT DARMSTADT

Coverage report: 82%

Module	statements ↑	missing	excluded	branches	partial	coverage
project__init__.py	0	0	0	0	0	100%
project\math_op.py	7	1	0	4	1	82%
Total	**7**	**1**	**0**	**4**	**1**	**82%**

```
1  from typing import Union
2
3  def add(num1: Union[int, float], num2: Union[int, float]) -> Union[int, float]:
4      if not isinstance(num1, (int, float)):
5          raise TypeError(f"'num1' is {type(num1)}; expected float, int")
6      if not isinstance(num2, (int, float)):                          6 →/7
7          raise TypeError(f"'num2' is {type(num2)}; expected float, int")
8      return num1 + num2
```

Writing tests

customize tests

TECHNISCHE UNIVERSITÄT DARMSTADT

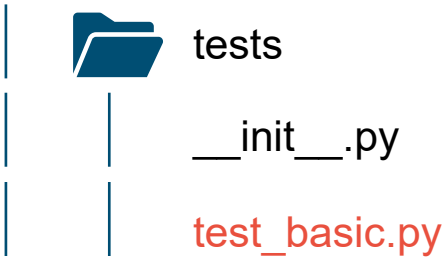

```python
def test_add_raises_type_error() -> None:
    with pytest.raises(TypeError):
        add("a", [1])
    with pytest.raises(TypeError):
        add(1, [1])
```

code duplication, inefficient

Writing tests

customize tests

TECHNISCHE UNIVERSITÄT DARMSTADT




```python
@pytest.mark.parametrize(„num1", [1, "1"])
@pytest.mark.parametrize(„num2", [[]])
def test_add_raises_type_error(num1, num2) -> None:
    with pytest.raises(TypeError):
        add(num1, num2)
```

100 % code coverage achieved = error free code?

Writing tests

customize tests

TECHNISCHE UNIVERSITÄT DARMSTADT

tests

__init__.py

test_basic.py

```python
@pytest.mark.parametrize("test_input,expected", [((2,2), 4),((0.1, -10), -9.9),
((0.1,0.2),0.3)])
def test_add(test_input, expected) -> None:
    assert add(test_input[0], test_input[1]) == expected
```

```
FAILED tests/test_math_op.py::test_add[test_input3-0.3] - assert 0.30000000000000004 == 0.3
```

Floats should never be checked for exact equality!

Writing tests

customize tests

TECHNISCHE UNIVERSITÄT DARMSTADT


```python
@pytest.mark.parametrize("test_input,expected", [((2,2), 4),((0.1, -10), -9.9),
((0.1,0.2),0.3)])
def test_add(test_input, expected) -> None:
    assert add(test_input[0], test_input[1]) == pytest.approx(expected)
```

Problem:

- How many different inputs need to be tested?

property-based testing

TECHNISCHE UNIVERSITÄT DARMSTADT

1. test with many inputs matching some specification→ Fuzzing
2. perform operations on the inputs
3. assert the result has some property

```python
def test_foo(a: str, b: int) -> None:
    output = foo(a, b)
    assert property(output)
    assert another_property(output)
```

- does not replace unit tests, but extends them → more efficient and robust for some test cases

https://hypothesis.readthedocs.io/en/latest/

property-based testing

example

TECHNISCHE UNIVERSITÄT DARMSTADT

tests

__init__.py

test_basic.py

```python
from hypothesis import given
import hypothesis.strategies as st

@given(num1 = st.integers(), num2 = st.integers())
def test_add(num1, num2):
    assert add(num1, num2) == num1 + num2
```

property-based testing

example

TECHNISCHE UNIVERSITÄT DARMSTADT

tests

__init__.py

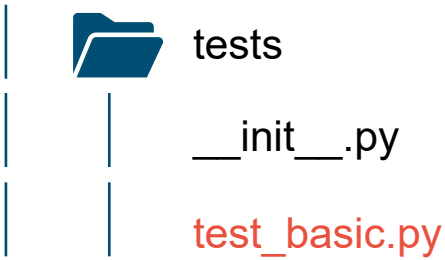
test_basic.py

```python
@st.composite
def int_or_float(draw):
    return draw(
        st.one_of(
            st.floats(allow_infinity=False, allow_nan=False), st.integers()
        )
    )
```

property-based testing

example

TECHNISCHE UNIVERSITÄT DARMSTADT




```python
@given(num1 = int_or_float(), num2 = int_or_float())
def test_add(num1, num2):
    assert add(num1, num2) == num1 + num2
```

property-based testing

TECHNISCHE UNIVERSITÄT DARMSTADT

Demo

property-based testing

example – generated inputs

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Trying example: test_add(
    num1=0, num2=0,
)
Trying example: test_add(
    num1=0.0, num2=0,
)
Trying example: test_add(
    num1=0.0, num2=0.0,
)
Trying example: test_add(
    num1=0, num2=0.0,
)
Trying example: test_add(
    num1=0, num2=0.0,
)
Trying example: test_add(
    num1=27819, num2=10000000.0,
)
```

```
Trying example: test_add(
    num1=0.0, num2=0.0,
)
Trying example: test_add(
    num1=0, num2=0.0,
)
Trying example: test_add(
    num1=-534380772, num2=-5.960464477539063e-08,
)
Trying example: test_add(
    num1=0.0, num2=0.0,
)
Trying example: test_add(
    num1=-0.0, num2=0.0,
)
Trying example: test_add(
    num1=-5.6185205645328266e+243, num2=3.402823466e+38,
)
```

property-based testing

example

TECHNISCHE UNIVERSITÄT DARMSTADT

```python
@given(st.lists(int_or_float()))
def test_sort(arr: list[Union[int, float]]) -> None:
    sorted_arr = bubble_sort(arr)
    assert isinstance(sorted_arr, list)
    assert Counter(arr) == Counter(sorted_arr)
    assert all(
        x <= y for x,y in zip(sorted_arr, sorted_arr[1:])
    )
```

https://speakerdeck.com/pycon2016/matt-bachmann-better-testing-with-less-code-property-based-testing-with-python?slide=15

Comments

How to write easy to test code?

TECHNISCHE UNIVERSITÄT DARMSTADT

- functions should perform only one task
- functions should be „pure functions“

„pure functions“:

- the function has identical outputs for identical inputs
  - no global variables inside the function
- no side effects
  - no print statements or other I/O actions
- „pure functions“ should be separated from „impure functions“

Comments

How to write easy to test code?

TECHNISCHE UNIVERSITÄT DARMSTADT

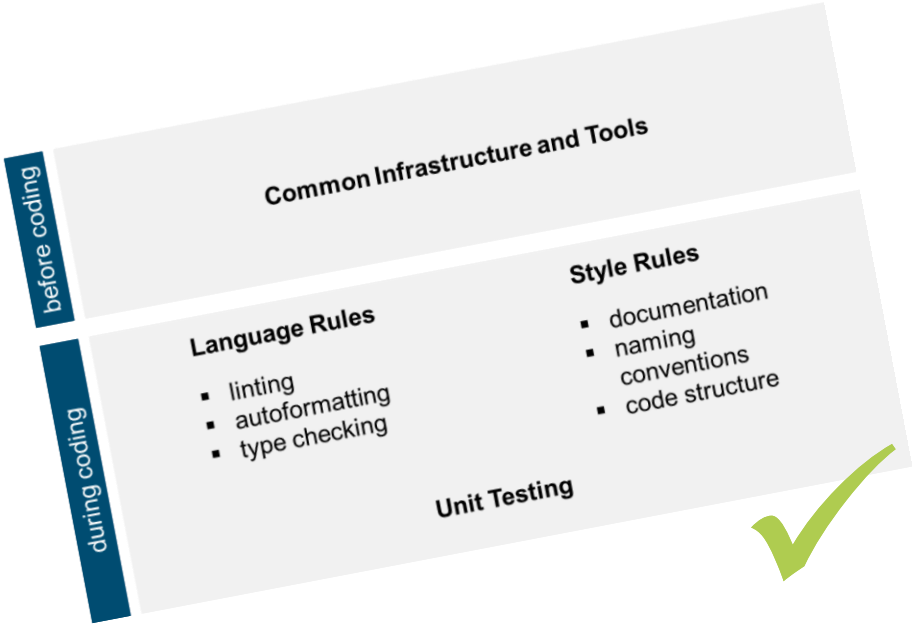
```python
def display_user_info(name: str, age: int) -> None:
    print(f"Name: {name}\nAge: {age}")
```

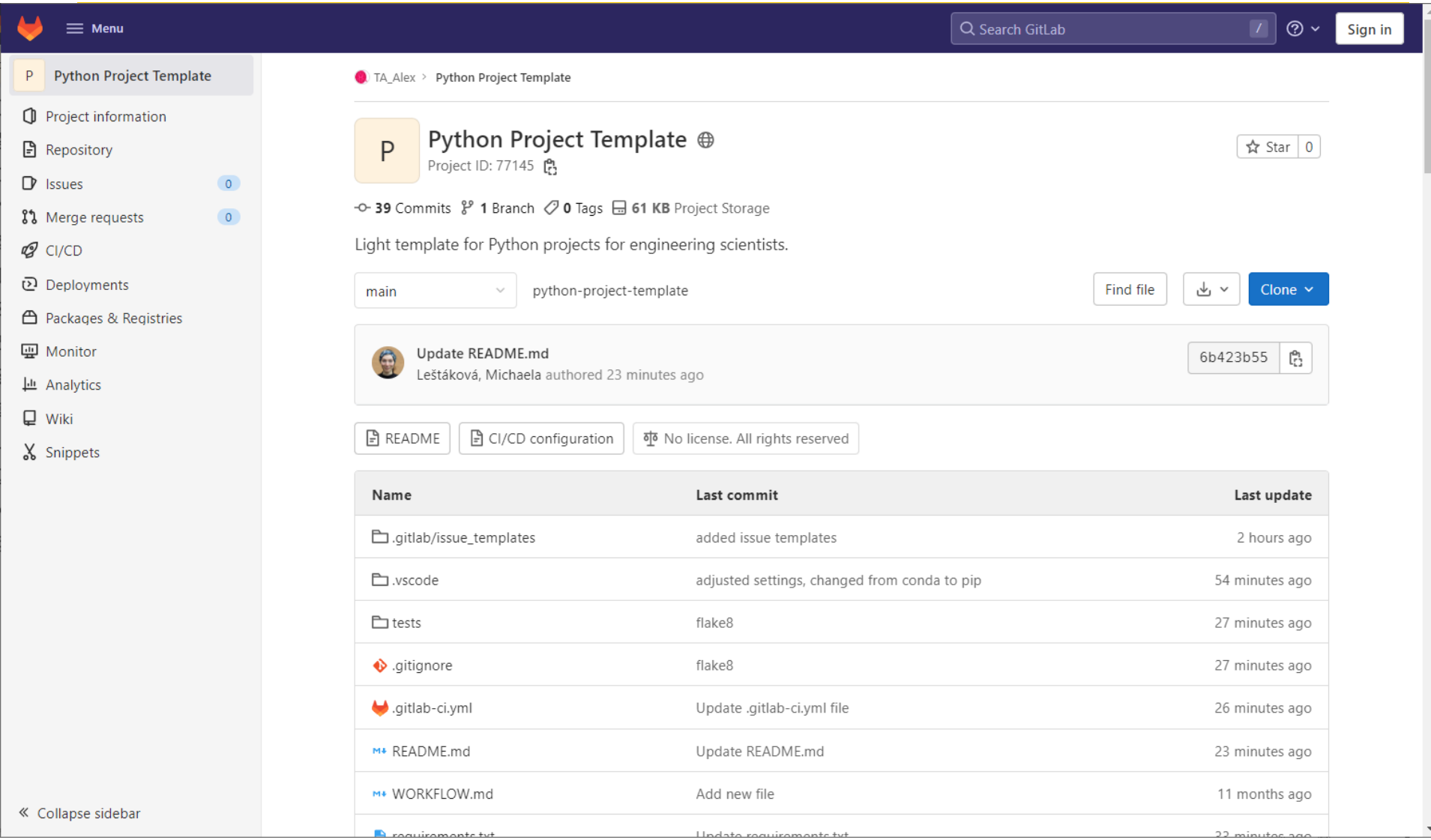
```python
def format_user_info(name: str, age: int) -> str:
    return f"Name: {name}\nAge: {age}"

def display_user_info(name: str, age: int) -> None:
    print(format_user_info(name, age))
```

Wrap-Up

TECHNISCHE UNIVERSITÄT DARMSTADT





https://git.rwth-aachen.de/ta_alex/python-project-template

Acknowledgements

Michaela Leštáková and Kevin T. Logan would like to thank the Federal Government and the Heads of Government of the Länder, as well as the Joint Science Conference (GWK), for their funding and support within the framework of the NFDI4Ing consortium. Funded by the German Research Foundation (DFG) - project number 442146713.