

# A No-Nonsense Guide to Higher Code Quality for Researchers



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

Michaela Leštáková, Kevin T. Logan, Daniele Inturri  
NFDI4ing Conference

```
def __init__(self):
    self.before = get_class(self) + get_class(self)
    self.after = get_class(self) + get_class(self)
    self.size = get_class(self) + get_class(self)

def apply(self):
    self.collection = get_class(self) + get_class(self)
    # ...
    # applies these parameters to a collection and returns a result
    # ...
    self = self.get_class(self)
    # TODO: This pattern after user before of both are set, but should be fine for
    # ...
    # self before to set size
    self["size"] = [self] + get_class(self)
    # self after to set size
    self["size"] = [self] + get_class(self)
    return collection.find(self["size"], self["size"], self["size"])
```

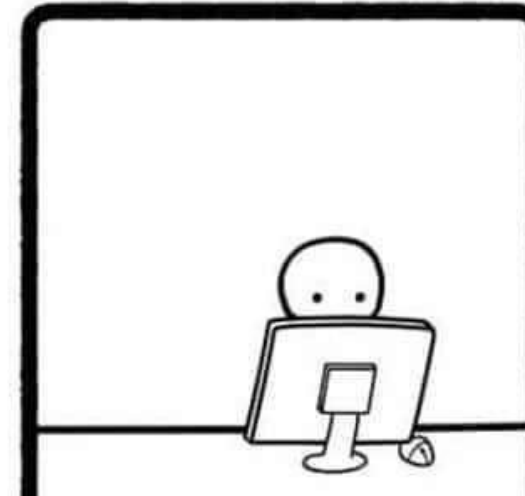
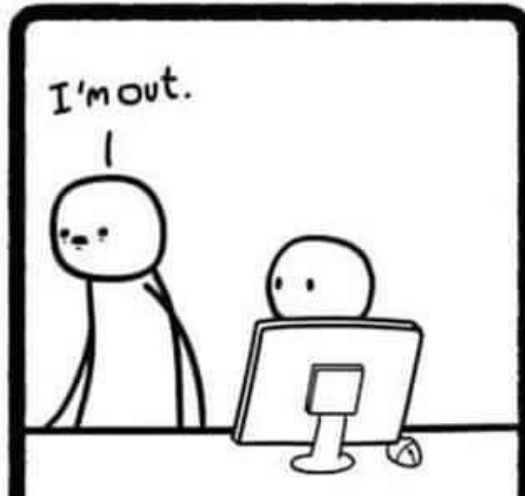
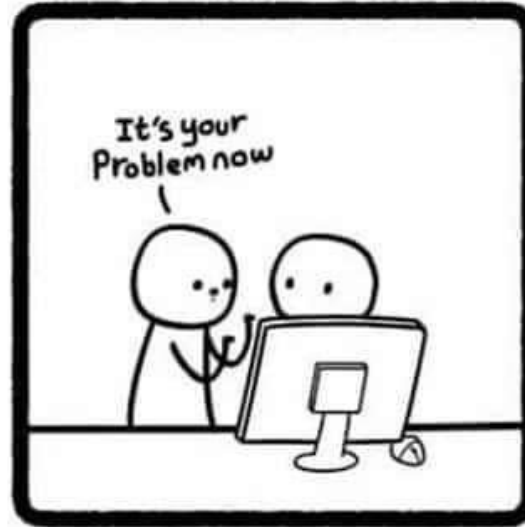
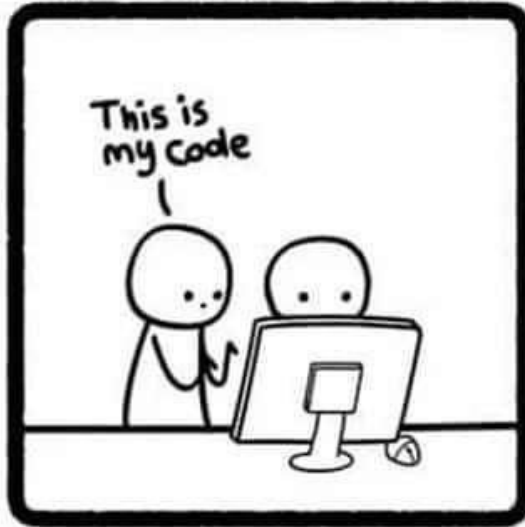
**MASCHINENBAU**

We engineer future

**FLUIDSYSTEMTECHNIK**

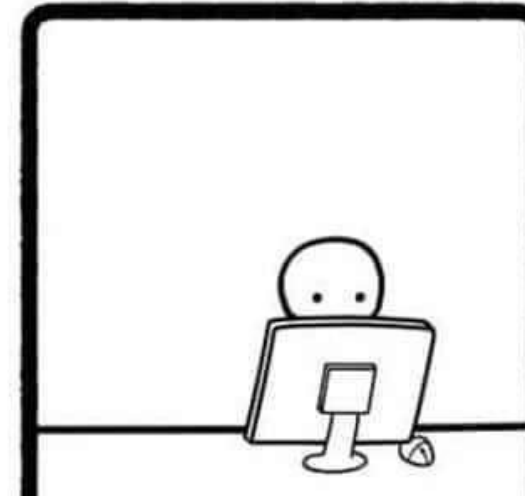
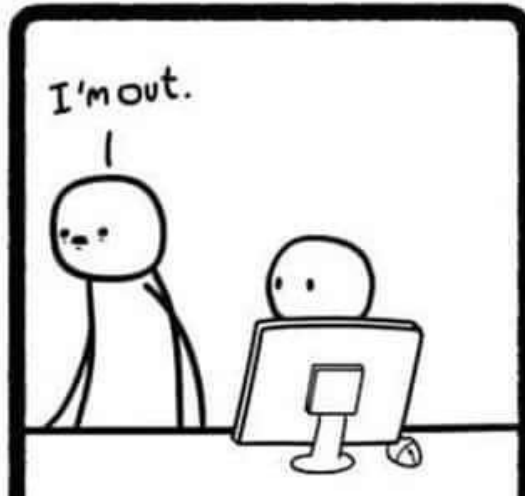
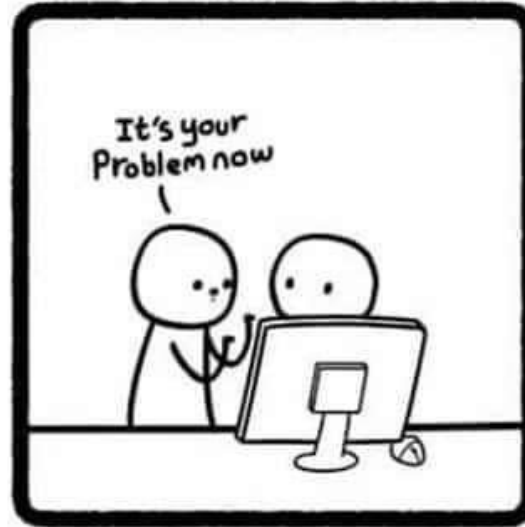
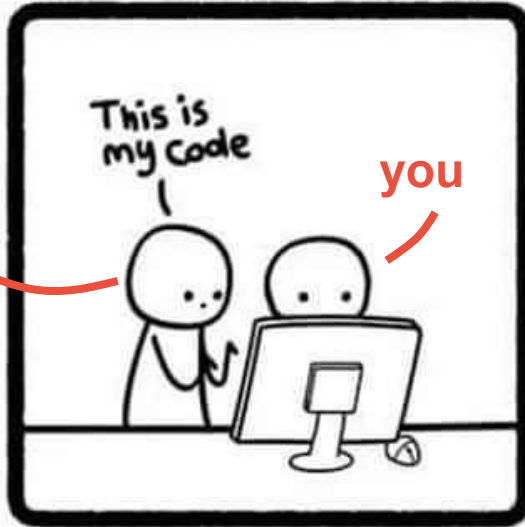
Prof. Dr.-Ing. Peter F. Pelz

# the Handover

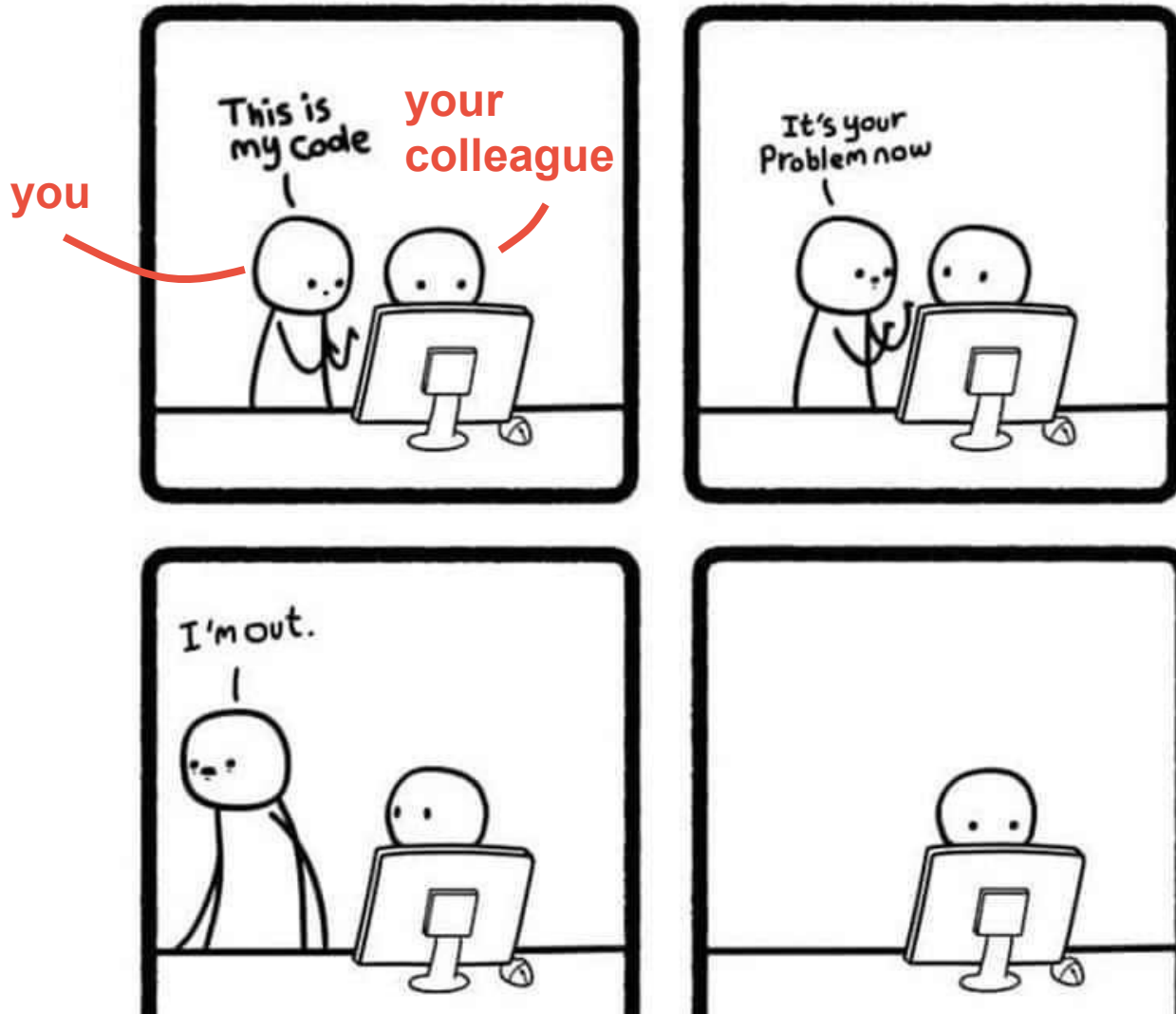


# the Handover

your  
colleague



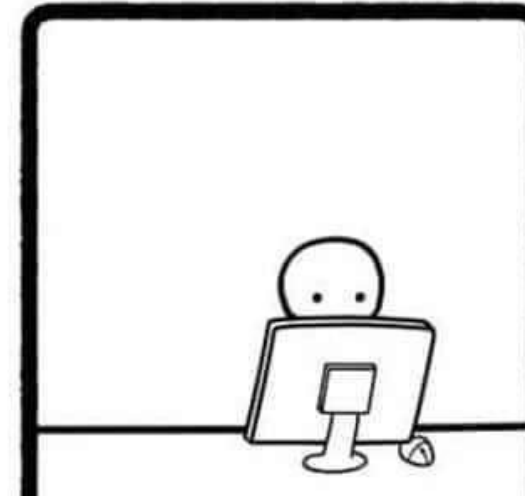
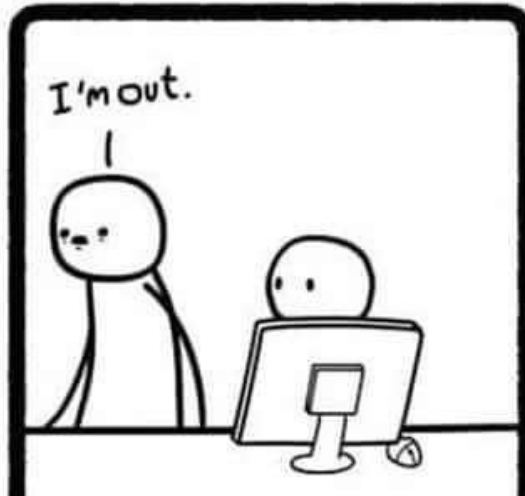
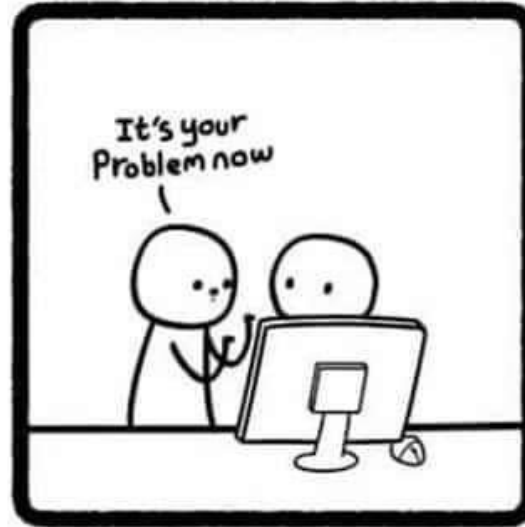
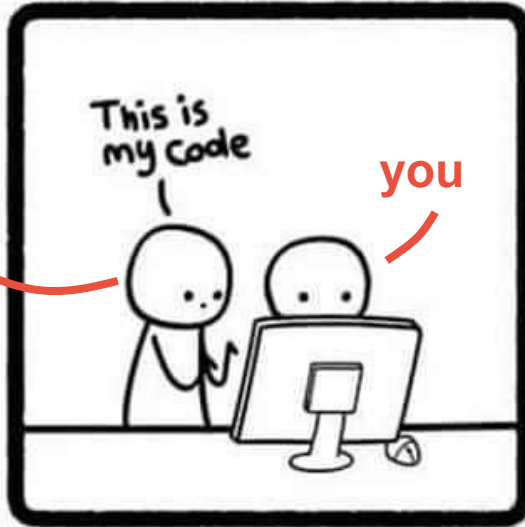
# the Handover



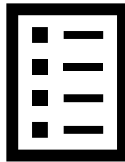
# the Handover

old you

you



# Today's Goal



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



zoom in onto unit testing

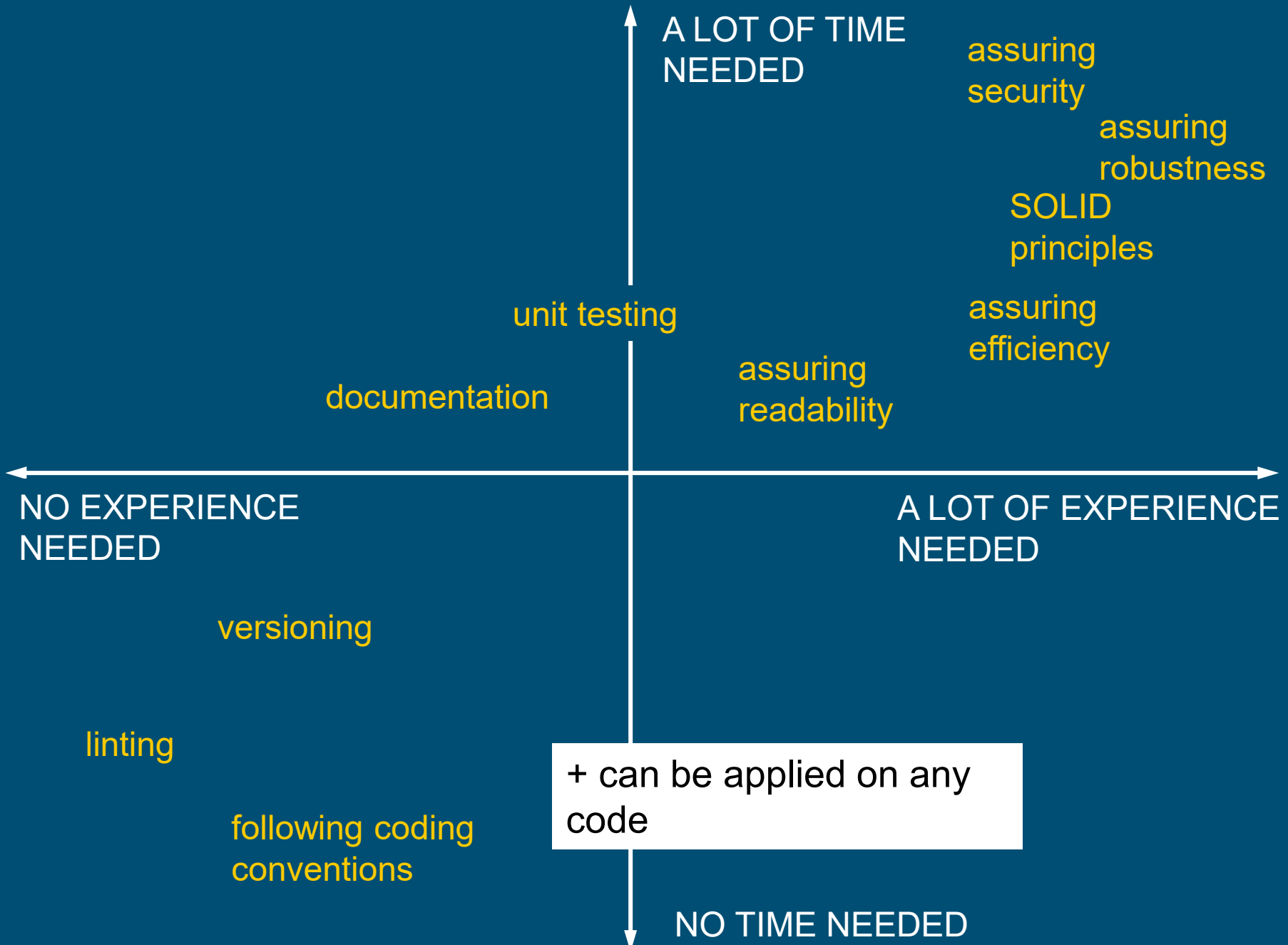
# What we will not look into

- 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

unit testing

assuring efficiency

documentation

assuring readability

NO EXPERIENCE NEEDED

A LOT OF EXPERIENCE NEEDED

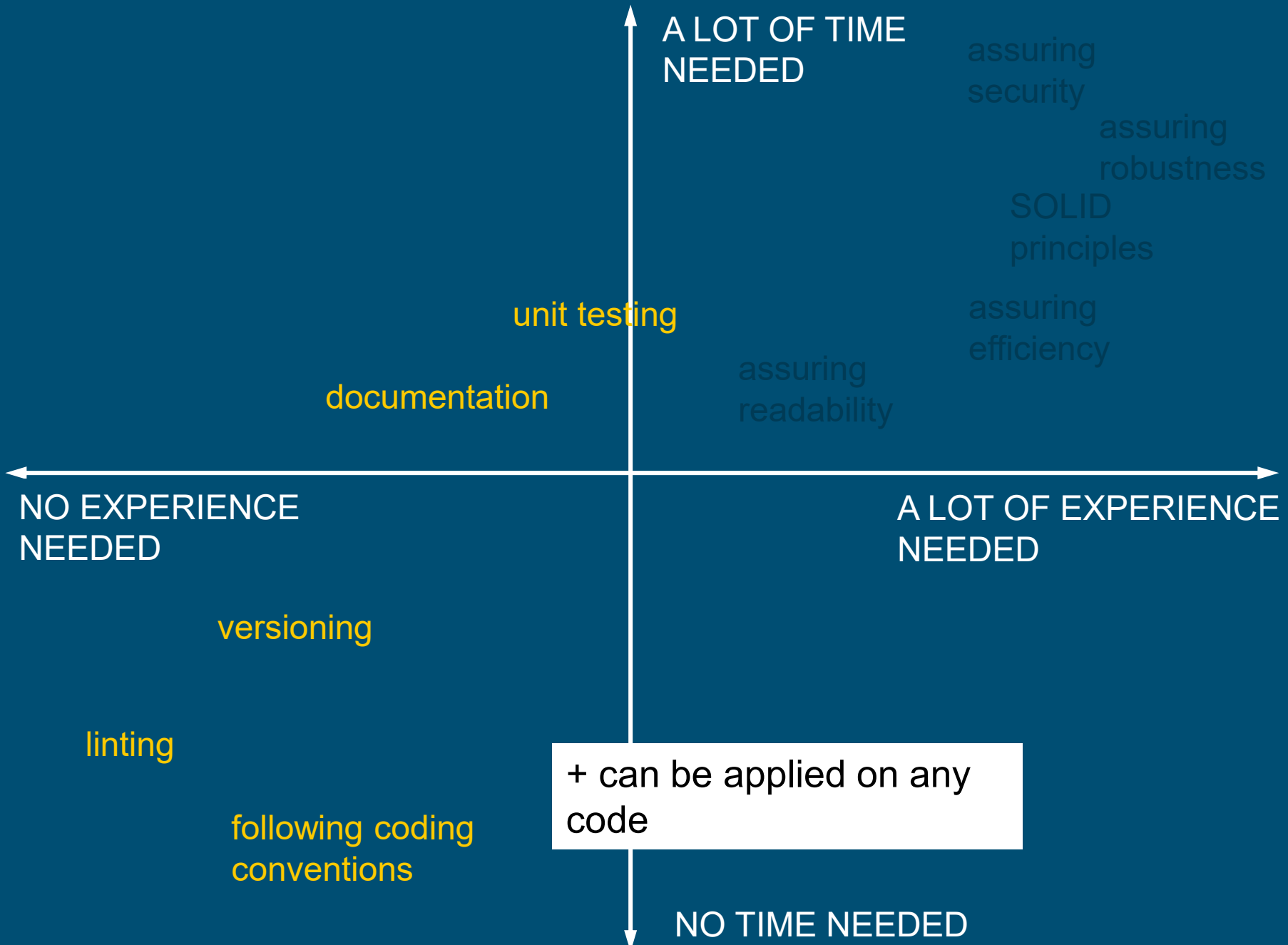
versioning

linting

following coding conventions

+ can be applied on any code

NO TIME NEEDED



A LOT OF TIME NEEDED

assuring security  
assuring robustness

SOLID principles

unit testing

assuring efficiency

documentation

assuring readability

NO EXPERIENCE NEEDED

A LOT OF EXPERIENCE NEEDED

versioning

linting

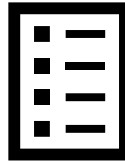
following coding conventions

+ can be applied on any code

NO TIME NEEDED

---

# Today's Goal



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

before coding

## Common Infrastructure and Tools

during coding

### Language Rules

- linting
- autoformatting
- type checking

### Style Rules

- documentation
- naming conventions
- code structure

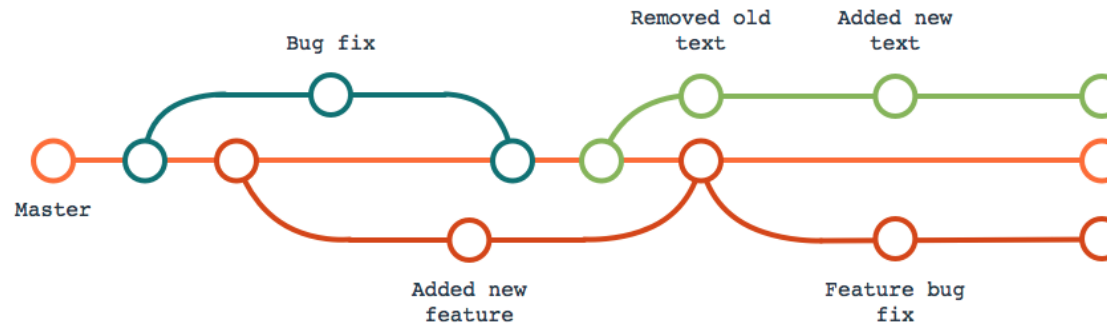
## Unit Testing

# **Common Infrastructure and Tools**

## **Version Control**

# Common Infrastructure and Tools

## Version Control

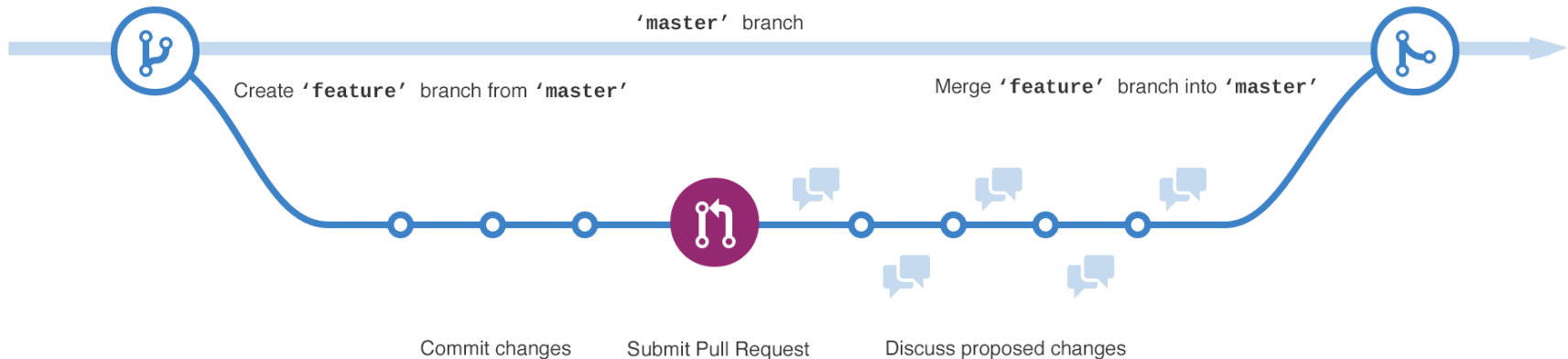


Picture Source: <https://blog.cpanel.com/git-version-control-series-what-is-git/>

- ✓ 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-workflow-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

**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**

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\_ips.inp networks U
  - model.py C:\Users\Lestakova\Documents\em...
  - 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

```
C:\Users\Lestakova\Documents\emergenCITY\WNTR\wntr\sim> hydraulics.py > ...
1 import pandas as pd
2 import numpy as np
3 import scipy.sparse as sparse
4 import math
5 import warnings
6 import logging
7 from wntr.network.model import WaterNetworkModel
8 from wntr.network.base import NodeType, LinkType, LinkStatus
9 from wntr.network.elements import Junction, Tank, Reservoir, Pipe, Pump, HeadPump, PowerPump, PRValve,
10   TCValve, GPValve, PBValve
11 from collections import OrderedDict
12 from wntr.utils.ordered_set import OrderedSet
13 from wntr.sim import aml
14 from wntr.sim.models import constants, var, param, constraint
15 from wntr.sim.models.utils import ModelUpdater
16
17 logger = logging.getLogger(__name__)
18
19
20 def create_hydraulic_model(wn, HW_approx='default'):
21     """
22     Parameters
23     -----
24     wn: WaterNetworkModel
25     mode: str
26     HW_approx: str
27         Specifies which Hazen-Williams headloss approximation to use. Options are 'default' and 'piecew
28         see the WNTR documentation on hydraulics for details.
29     """
```

TERMINAL

```
Microsoft Windows [Version 10.0.19043.2006]
(c) Microsoft Corporation. Alle Rechte vorbehalten.

C:\Users\Lestakova\Documents\emergenCITY\tutorials>
```

# Common Infrastructure and Tools

## Code Editor

- 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

**Agree on and use the same editor.**

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

**Language Rules**  
**Linting**

# Language Rules Linting

```
48
49 #Visualisieren des Netzwerks
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
62
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

- 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

```
48
49 #Visualisieren des Netzwerk
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

**Agree on a linter and lint on save.  
(+) add linting into your CI/CD pipeline.**

# Language Rules Autoformatting

# Language Rules

## Autoformatting

- 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

```
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

**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

- **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

```
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
```



```
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

**Agree on and use a type checker.**

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



# **Style Rules Documentation**

# Style Rules

## Documentation

docstring

comment

```
19
20 def create_hydraulic_model(wn, HW_approx='default'):
21     """
22     Parameters
23     -----
24     wn: WaterNetworkModel
25     mode: str
26     HW_approx: str
27     | Specifies which Hazen-Williams headloss approximation to use. Options are 'default' and 'piecewise'. Please
28     | see the WNTR documentation on hydraulics for details.
29
30     Returns
31     -----
32     m: wntr.aml.Model
33     model_updater: wntr.models.utils.ModelUpdater
34     """
35     m = aml.Model()
36     model_updater = ModelUpdater()
37
38     # Global constants
39     constants.hazen_williams_constants(m)
40     constants.head_pump_constants(m)
41     constants.leak_constants(m)
42     constants.pdd_constants(m)
43
44     param.source_head_param(m, wn)
45     param.expected_demand_param(m, wn)
46     mode = wn.options.hydraulic.demand_model
47     if mode in ['DD', 'DDA']:
48         pass
```

# Style Rules

## Documentation

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

```
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
```



```
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

## Interactive documentation using Sphinx

```
def get_volume(self, level=None):  
    """  
    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.  
    """  
  
    if self.vol_curve is None:  
        A = (np.pi / 4.0 * self.diameter**2) * self.height  
        if level is None:  
            level = self.level  
        vol = A * level  
  
    ...
```

- Simulation results
- Disaster scenarios
- Criticality analysis
- Resilience metrics
- Fragility curves
- Network morphology
- Graphics
- Advanced simulation techniques
- Copyright and license
- Release notes
- Software quality assurance
- User community

### API documentation

#### Subpackages

- wntr.epanet package
- wntr.graphics package
- wntr.metrics package
- wntr.morph package

#### wntr.network package

##### Submodules

- wntr.scenario package
- wntr.sim package
- wntr.utils package

`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:

$Q = \text{discharge\_coeff} * \text{area} * \text{sqrt}(2 * g * h)$

#### 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 ( $P_g + \text{elevation}$ )

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

**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

- 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                   | <code>lower_with_under</code>   |  |
| Modules                    | <code>lower_with_under</code>   | <code>_lower_with_under</code>               |
| Classes                    | <code>CapWords</code>           | <code>_CapWords</code>                       |
| Exceptions                 | <code>CapWords</code>           |  |
| Functions                  | <code>lower_with_under()</code> | <code>_lower_with_under()</code>             |
| Global/Class Constants     | <code>CAPS_WITH_UNDER</code>    | <code>_CAPS_WITH_UNDER</code>                |
| Global/Class Variables     | <code>lower_with_under</code>   | <code>_lower_with_under</code>               |
| Instance Variables         | <code>lower_with_under</code>   | <code>_lower_with_under</code> (protected)   |
| Method Names               | <code>lower_with_under()</code> | <code>_lower_with_under()</code> (protected) |
| Function/Method Parameters | <code>lower_with_under</code>   |  |
| Local Variables            | <code>lower_with_under</code>   |  |

[PEP8 summarized in](#)

[styleguide | Style guides for Google-originated open-source projects](#)

**Style Rules**  
**Code Structure**



# Style Rules

## Code Structure

**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

```
17 def __init__(self):
18     self._before = None
19     self._after = None
20     self._size = None
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

# What to expect

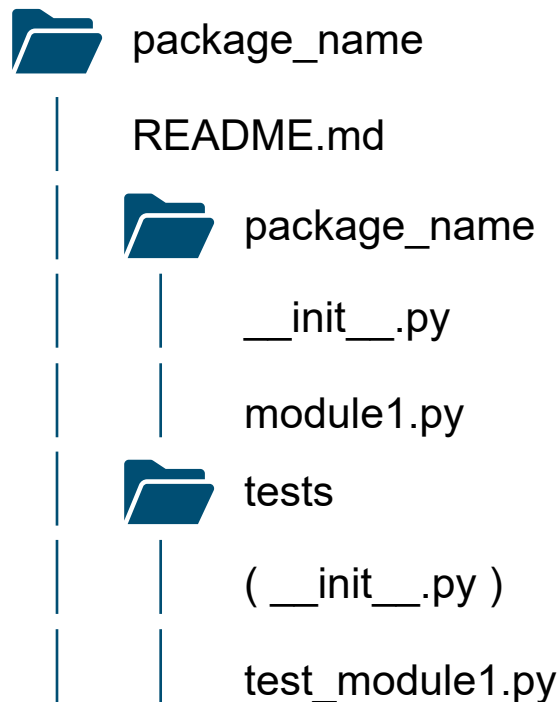
- general introduction to automated testing
- introduction to various tools for testing in python
- metrics for completeness/quality of tests → code coverage
- advanced testing strategies → property-based-testing
- tips for writing testable code

# Why write tests?

- 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

## 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

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



math\_operations

\_\_init\_\_.py

basic.py

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



# Writing tests

## The first test



tests

\_\_init\_\_.py

test\_basic.py

naming convention: test\_\*.py / \*\_test.py



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

naming convention: test\_\*



```
\math_operations$ pytest
```

---

# Writing tests

# Demo

# Writing a module

## Example function



math\_operations

\_\_init\_\_.py

basic.py

```
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



tests

\_\_init\_\_.py

test\_basic.py



```
import pytest

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

---

# Writing tests

# Demo

# Code Coverage

- 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

```
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

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

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

```
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

```
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

- 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

# Demo

# Code Coverage

pytest-cov

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%        |
| <b>Total</b>        | <b>7</b>     | <b>1</b> | <b>0</b> | <b>4</b> | <b>1</b> | <b>82%</b> |

```
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)):
7         raise TypeError(f"'num2' is {type(num2)}; expected float, int")
8     return num1 + num2
```

# Writing tests

customize tests



tests

\_\_init\_\_.py

test\_basic.py



```
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



tests

\_\_init\_\_.py

test\_basic.py



```
@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



tests

\_\_init\_\_.py

test\_basic.py



```
@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



tests

\_\_init\_\_.py

test\_basic.py

```
@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

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



```
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



tests

\_\_init\_\_.py

test\_basic.py



```
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



tests

\_\_init\_\_.py

test\_basic.py



```
@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



tests

\_\_init\_\_.py

test\_basic.py

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

---

# property-based testing

## Demo

# property-based testing

## example – generated inputs



```
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

```
● ● ●  
@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?

- 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?



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



```
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

before coding

Common Infrastructure and Tools

during coding

Language Rules

- linting
- autoformatting
- type checking

Style Rules

- documentation
- naming conventions
- code structure

Unit Testing



Writing tests  
Adding a second test

tests

- \_\_init\_\_.py
- test\_basic.py

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


MASCHINENBAU  
We engineer future


Unit Testing in Python | Daniele Inturri 25.10.2022 10






Python Project Template


TA\_Alex > Python Project Template



**Python Project Template**  ☆ Star 0

Project ID: 77145 




39 Commits  1 Branch  0 Tags  61 KB Project Storage









Light template for Python projects for engineering scientists.

main  python-project-template Find file ↓ Clone ↓

 **Update README.md** 6b423b55 

Leštáková, Michaela authored 23 minutes ago

 README  CI/CD configuration  No license. All rights reserved

| Name  | Last commit                                  | Last update    |
|---|--|----------------|
|  .gitlab/issue_templates | added issue templates                        | 2 hours ago    |
|  .vscode                 | adjusted settings, changed from conda to pip | 54 minutes ago |
|  tests                   | flake8                                       | 27 minutes ago |
|  .gitignore              | flake8                                       | 27 minutes ago |
|  .gitlab-ci.yml         | Update .gitlab-ci.yml file                   | 26 minutes ago |
|  README.md             | Update README.md                             | 23 minutes ago |
|  WORKFLOW.md           | Add new file                                 | 11 months ago  |
|  requirements.txt      | Update requirements.txt                      | 22 minutes ago |

[https://git.rwth-aachen.de/ta\\_alex/python-project-template](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.

**NFDI4ing**

